Published in *Proc. of the 25th Annual ACM/IEEE International Symposium on Computer Architecture (ISCA'98)*, Barcelona, Spain, June 1998.

Declustered Disk Array Architectures with Optimal and Near-optimal Parallelism

Guillermo A. Alvarez*° Walter A. Burkhard* Larry J. Stockmeyer◁ Flaviu Cristian°

*Gemini Storage Systems Laboratory
°Dependable Systems Laboratory
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0114 USA
{galvarez,burkhard,flaviu}@cs.ucsd.edu

◁IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099 USA
stock@almaden.ibm.com

Abstract

This paper investigates the placement of data and parity on redundant disk arrays. Declustered organizations have been traditionally used to achieve fast reconstruction of a failed disk's contents. In previous work, Holland and Gibson identified six desirable properties for ideal layouts; however, no declustered layout satisfying all properties has been published in the literature. We present a complete, constructive characterization of the collection of ideal declustered layouts possessing all six properties. Given that ideal layouts exist only for a limited set of configurations, we also present two novel layout families. PRIME and RELPR can tolerate multiple failures in a wide variety of configurations with slight deviations from the ideal. Our simulation studies show that the new layouts provide excellent parallel access performance and reduced incremental loads during degraded operation, when compared with previously published layouts. For large accesses and under high loads, response times for the new layouts are typically smaller than those of previously published declustered layouts by a factor of 2.5.

1 Introduction

A variety of applications, such as on-line transaction processing and scientific computing, require high throughput from cost-effective storage subsystems. Disk arrays offer significant advantages over individual disks. By aggregating multiple disk drives, arrays achieve better transfer bandwidths and potential reductions on seek and rotational latencies due to smaller-sized individual drives. However, the reliability of a disk array decreases linearly with the number of disk drives [15]—data can be irretrievably lost in a matter of days. Redundant information (e.g. parity) must thus be stored in the array along with client data to enable recovery from disk failures.

In this paper we investigate array *layouts*, i.e. mappings of client data and redundant information onto the disks of the array. A popular method is to organize the data into *stripes*. Each stripe contains a constant number of *units*, all of the same size; each unit is mapped to contiguous sectors of the same disk, but different units of the same stripe are mapped to different disks. If, for example, a single failure must be tolerated, all but one of the units in each stripe will be *data units* containing client information. The remaining unit will be a *check unit* containing some form of redundancy for the data units in the same stripe (e.g. their exclusive-or if parity encoding is used). If a disk becomes unavailable, all stripe units mapped to it are recoverable by virtue of the redundancy in the scheme.

In *declustering* organizations (originally suggested by Muntz and Lui [14], and evaluated by Holland and Gibson [11]), each stripe is mapped to just k of the n disks in the array (where $k \leq n$), to achieve significant performance improvements during both degraded operation and on-line disk reconstruction. These layouts rely on balanced incomplete block designs (*BIBDs*) [9], which are known to exist for only some array configurations, and may require the storage of tables with $O(k\binom{n}{k})$ entries. Schwabe and Sutherland [16] introduced BIBD-based layouts for a restricted

Technical Report CS98-576, CSE, UCSD, February 1998.

subset of the possible configurations, as well as perturbations that yield layouts for other cases with a bounded degradation in performance. Alvarez, Burkhard, and Cristian [2] presented layouts that compute on-the-fly mappings based on complete block designs, instead of storing block design tables in memory.

There are six desirable properties of declustered disk array layouts that can tolerate a single failure, originally identified in [11]:

1. Single failure correcting: No two units of the same stripe are mapped to the same disk, to make recovery possible from a single disk crash.

2. Distributed parity: All disks have the same number of check units mapped to them, to balance the accesses to check units during writes or when a failure has occurred.

3. Distributed reconstruction: There is a constant such that, for every pair of disks, that constant number of stripes have units mapped to both disks (to ensure that the accesses to surviving disks during on-line reconstruction are spread evenly).

4. Large write optimization: Each stripe contains a contiguous interval of the client's data, to process a write of all $(k-1)$ data units without pre-reading the prior contents of any disk.

5. Maximal parallelism: Whenever a client requests a read of n contiguous data units, all n disks are accessed in parallel.

6. Efficient mapping: The functions that map client addresses to array locations are efficiently computable, with low time and space requirements.

A layout is *ideal* if it satisfies all six properties. To the best of our knowledge, no previously published declustered layout is ideal. The BIBD-based layouts presented in [11, 16] satisfy the first three properties; and depending on how the client data units are mapped to the stripes, they can be made to satisfy either large write optimization or maximal parallelism, although it was not known how to satisfy both simultaneously. Holland [12] studied the problem of rearranging the placement of client data units to achieve better parallelism while maintaining large write optimization, although he did not achieve maximal parallelism. The time requirement to compute a BIBD-based mapping is reasonably small, although the space requirement depends on the size of the BIBD table, which in turn depends on the values of n and k.

It is natural to wonder whether ideal layouts actually exist. The first result of this paper gives a complete, constructive characterization of the combinations of k and n that allow ideal layouts. With the exception of three special cases, ideal layouts are possible only when $k = n$, $k = n - 1$, or $k = 2$. Moreover, for all cases in which ideal layouts are impossible, we use only the first five properties to prove impossibility. This result gives a formal explanation for the difficulties encountered previously in trying to achieve properties 4 and 5 simultaneously in BIBD-based layouts.

Because ideal layouts are possible only for very limited choices of k and n, it is reasonable to ask what can be achieved if some of the properties hold only approximately. Another contribution of this paper is to present near-ideal layouts that do not require storing any table in controller memory. The novel layouts use different *strides* between consecutive stripe units; that is, if s denotes the stride and unit a in the client's address space is mapped to disk d, then unit $(a + 1)$ will be mapped to disk $(d + s) \bmod n$. This construction was originally presented in [17], and independently rediscovered in [1]. For any prime n and any k, the PRIME family of layouts satisfies all of the properties except maximal parallelism, and moreover, during large reads each disk will read at most one more unit than the optimal. These layouts are based on well-known BIBDs [10, 16]. For general n and k, the RELPR layout family differs from ideal in two ways: there can be an imbalance of at most one unit during large reads as before, and the reconstruction workload is not perfectly uniform for nonprime n. We provide a bound on the reconstruction workload for general n.

Independent of these considerations, there are numerous reasons for considering architectures capable of tolerating multiple concurrent failures [2, 3, 4, 7]. For example, there may be latent sector errors [19] that are discovered when the array already has suffered a failure and thus cannot tolerate another one, or communication failures that render disks inaccessible in network striping approaches [8]. The PRIME and RELPR layouts can tolerate multiple failures, by keeping the necessary number of check units in each stripe. In the case of layouts that tolerate multiple failures, we refer to the distributed parity property as *distributed check information*.

We evaluated the performance of PRIME and RELPR by running a suite of experiments on an accurate disk array simulation tool [6], for a variety of workloads and failure scenarios. The performance of our layouts was compared with RAID-5 [15] and Parity Declustering [11]. The comparisons show that the layouts presented in this paper provide the same performance for small accesses. For large enough accesses, our layouts achieve more than twice the throughput of their competitors under high loads; for smaller accesses, our layouts provide better performance for light loads, and slightly worse performance for high loads.

Section 2 lists our basic definitions and assumptions. Section 3 presents the characterization of the layouts that are ideal. Section 4 introduces our near-optimal layouts PRIME and RELPR. We evaluate the performance of our layouts in Section 5. Finally, we draw some conclusions in Section 6.

2 Definitions

There are four parameters for a declustering layout: the number of disk drives $n \geq 2$, the number of units per stripe $k \leq n$, the number of stripes b, and the number of stripe units per disk r. It follows that both bk and nr count the total number of stripe units, so $bk = nr$. One other parameter is necessary to characterize the level of failure tolerance to be provided: let m denote the number of client data units per stripe. Typically, a minimum of m (data or check) units suffices to reconstruct all units in the stripe. We will occasionally refer to $f = k - m$, the maximum number of failures that can be tolerated simultaneously. In case $m = k-1$ (i.e. $f = 1$), we have the traditional single-disk-failure tolerating layouts. Each stripe consists of m client data units and f check units. In practical situations there is also a need to determine the size of each unit. This problem has been investigated in [5]; since the validity of the formal results in this paper is independent of this parameter, we will not discuss it in what follows.

Each client data unit is named by an address a in the range $0 \leq a < mb$. Each unit within a stripe has an address (s, u), where s is the stripe name with $0 \leq s < b$ and u is the stripe unit name with $0 \leq u < k$. Since the names of the stripe units are arbitrary, we assume the check units are the last units of the stripe, i.e. that the names $(s, k - f)$,...,$(s, k - 1)$ correspond to check units for all s. Each location has also an address within the disk array, (d, p) where d is the name of the disk with $0 \leq d < n$, and p is the offset within the disk with $0 \leq p < r$.

A layout is specified by a function *stripe* mapping client data addresses one-to-one onto stripe addresses of data units, and a function *array* mapping stripe addresses one-to-one onto array addresses:

$$\begin{array}{rcl} \mathit{stripe} : \{0, 1, \ldots, bm - 1\} & \rightarrow & \{0, 1, \ldots, b - 1\} \times \{0, 1, \ldots, m - 1\} \\ \mathit{array} : \{0, 1, \ldots, b - 1\} \times \{0, 1, \ldots, k - 1\} & \rightarrow & \{0, 1, \ldots, n - 1\} \times \{0, 1, \ldots, r - 1\} \end{array}$$

That is, $\mathit{stripe}(a) = (s, u)$ means that client data unit a is mapped to stripe unit (s, u) and $\mathit{array}((s, u)) = (d, p)$ means that stripe unit (s, u) is mapped to array location (d, p). By composing these two functions, we can extend *array* to map client data units to array locations, for example, $\mathit{array}(a) = \mathit{array}(\mathit{stripe}(a))$. It will also be convenient to break *array* into two components, one for the disk name and one for the offset. That is, the functions *disk* and *offset* are such that:

$$\begin{array}{rcl} \mathit{array}(a) & = & (\mathit{disk}(a), \mathit{offset}(a)) \\ \mathit{array}((s, u)) & = & (\mathit{disk}(s, u), \mathit{offset}(s, u)) \end{array}$$

It is straightforward to precisely define the first five desirable properties from Section 1, in terms of the notation introduced above. We refer the interested reader to the Appendix for the details. A layout is *placement-ideal* if it satisfies the first five properties. The "efficient mapping" property is more difficult to define formally than the other five; however, mappings for both the novel layouts presented here and previous declustering layouts [11, 16] can be computed with $O(1)$ arithmetic operations. The previously cited table-based layouts require $O(k\binom{n}{k})$ storage, while our layouts require no tables. Optionally, it may be desirable to store a single table containing at most $(n - 2)$ integers to speed up RELPR's mappings.

A layout with parameters n, k, r for large r is typically obtained by first obtaining a layout n, k, r' for some smaller r' and then *repeating* the layout several times [2, 11]. When r is not a multiple of r', the uniformity properties of the small layout may not be preserved exactly; although the discrepancy is small if r/r' is large.

For positive integers x and y, the notation $x|y$ means that x divides y (i.e., $y = zx$ for some integer z).

3 Ideal Layouts

This section focuses on single-failure-tolerating schemes. Our results regarding placement-ideal layouts follow from a sequence of two lemmas and a theorem. See the Appendix for the proofs of this and the other results in this paper[1].

Lemma 2 *If a layout has the maximal parallelism property, then there is a way to name the disks* $0, 1, \ldots, n-1$ *such that disk*$(a) = a \bmod n$ *for every address* a *of client data.*

A consequence of this lemma is that repeating an ideal layout produces an ideal layout. The next lemma states that, for some naming of the stripe units, client stripe units with addresses in the order $(0, 0), (0, 1), \ldots,$ $(0, k-2), (1, 0), (1, 1), \ldots, (1, k-2), (2, 0), \ldots$ must be mapped to the disks in round-robin order.

Lemma 3 *If a layout has the large write optimization and maximal parallelism properties, then (for some naming of the disks and the stripe units) the mapping of stripe data units to disks is given by disk*$((s, u)) = ((k-1)s+u) \bmod n$ *for all* s *with* $0 \leq s < b$ *and all* u *with* $0 \leq u < k-1$.

Actually, Lemmas 2 and 3 hold in general for $f \geq 1$, with $(k-1)$ replaced by m in Lemma 3. From the two lemmas, the large write optimization and the maximal parallelism properties together determine how stripe data units are mapped to disks. For every placement-ideal layout, this mapping is fixed; the only freedom is in the mapping of check units, and in the offsets within each disk.

Our characterization of when placement-ideal layouts are possible is:

Theorem 5 *There is a placement-ideal layout with parameters* n, k, r *if and only if*

$$k | r \text{ and } (n-1) | (r(k-1))$$

and one of the following holds: i) $k = n$*, ii)* $k = n-1$*, iii)* $k = 2$*, iv)* $k = 3$ *and* $n = 5$ *or* 7*,*
v) $k = 4$ *and* $n = 7$.

For example, the case with $k = n$ is well known: the left-symmetric RAID-5 layout [13] is placement-ideal, with $r = k = n$. For case (v) in the theorem, here is a placement-ideal layout with $n = 7$ and $k = 4$. Each column corresponds to a disk (beginning with disk 0 for the leftmost column), and each row corresponds to an offset within the disks starting from 0 at the top; $Ds.u$ denotes data unit u for stripe s, and Cs denotes its check unit.

D0.0	D0.1	D0.2	D1.0	D1.1	D1.2	D2.0
D2.1	D2.2	D3.0	D3.1	D3.2	D4.0	D4.1
D4.2	D5.0	D5.1	D5.2	D6.0	D6.1	D6.2
C1	C6	C4	C2	C0	C5	C3

Note that, according to Lemma 2, client addresses are mapped to disks in round-robin order. All placement-ideal layouts constructed in Theorem 5 have $r \leq 2n$ rows (which compares quite favorably with previous block-design-based layouts [11, 16]), and admit mappings that require a few arithmetical operations and no table storage. Instead of being intermixed with data units, check units are clustered together at regular intervals (c.f. the last row of the example); this is common to all layouts presented in this paper.

Because the mappings are efficiently computable in all cases for which placement-ideal layouts exist, it can be stated at an informal level that Theorem 5 holds with "ideal" in place of "placement-ideal".

4 Almost Ideal Layouts

We investigate how the situation changes if some of the desirable properties are allowed to hold only approximately. When n is a prime number, the PRIME layouts satisfy five of the six properties, and the parallelism property very close to the optimal. For general n, the RELPR layouts satisfy four of the six properties, with a slight degradation in parallelism and distributed reconstruction. Both layouts can tolerate multiple failures.

[1] The numbering of our results in the body of the paper follows the numbering in the Appendix.

4.1 PRIME

For prime values of n, the PRIME layout family is very close to ideal by relaxing slightly the parallelism property. For a layout L we define the function $\tau_L(o)$, the *parallel read count*, to be the maximum number of data units that any disk must supply when reading any o consecutive units of client data. It is easy to see that, for any access size $1 \leq o \leq bm$, it must be that $\tau_L(o) \geq \lceil o/n \rceil$. Say that a layout L has *optimal parallelism* if $\tau_L(o) = \lceil o/n \rceil$ for all such o. It follows from Lemma 2 that a layout has maximal parallelism as defined in Section 1 if and only if it has optimal parallelism.

The layout is obtained from the ring-based block designs of Schwabe and Sutherland [16]; the new observation is that client data can be mapped to stripes so that the parallel read count is almost optimal. In this construction, each stripe now has a name of the form (x, y) where $0 \leq x, y < n$ and $y \neq 0$. There are $n(n-1)$ stripes in the layout; of course, we can repeat the layout to fill the disks. The stripe with name (x, y) has disks in the set $\{ (x + iy) \bmod n \mid 0 \leq i < k \}$.

To compute our mapping of client data units a to array locations, we first calculate

$$z = \left\lfloor \frac{a}{mn} \right\rfloor . \tag{1}$$

That is, z is such that a lies in the z^{th} iterations of mn consecutive client data unit addresses.

$$y = (z \bmod (n-1)) + 1 \tag{2}$$

$$\textit{disk}(a) = ay \bmod n \tag{3}$$

$$\textit{offset}(a) = \lfloor a/n \rfloor + fz. \tag{4}$$

When tolerating f failures, we have f check units per stripe. The location of check unit i with $0 \leq i < f$, for the stripe that contains client address a is computed as:

$$\textit{check-disk}_i(a) = (((\lfloor a/m \rfloor + 1)m + i) \cdot y) \bmod n \tag{5}$$

$$\textit{check-offset}_i(a) = (z+1)k - f + i \tag{6}$$

As an example, for $n = 5$, $k = 4$ and $f = 2$ we have the following layout (where $Cs.i$ denotes check unit i of stripe s):

D0.0	D0.1	D1.0	D1.1	D2.0
D2.1	D3.0	D3.1	D4.0	D4.1
C4.0	C2.0	C0.0	C3.0	C1.0
C1.1	C4.1	C2.1	C0.1	C3.1
D5.0	D6.1	D5.1	D7.0	D6.0
D7.1	D9.0	D8.0	D9.1	D8.1
C9.0	C8.0	C7.0	C6.0	C5.0
C6.1	C5.1	C9.1	C8.1	C7.1
D10.0	D11.0	D12.0	D10.1	D11.1
D12.1	D13.1	D14.1	D13.0	D14.0
C14.0	C10.0	C11.0	C12.0	C13.0
C11.1	C12.1	C13.1	C14.1	C10.1
D15.0	D17.0	D16.1	D16.0	D15.1
D17.1	D19.1	D19.0	D18.1	D18.0
C19.0	C16.0	C18.0	C15.0	C17.0
C16.1	C18.1	C15.1	C17.1	C19.1

The structure of this double-failure-tolerant instance of PRIME is as follows. The $r = k(n-1) = 16$ rows are subdivided into $n - 1 = 4$ *iterations* of $k = 4$ rows each. In the first iteration we have $y = 1$, i.e. the next data unit is always in the next disk modulo n. Within the second iteration $y = 2$, thus if data unit a is in disk d, then data unit $(a + 1)$ is in disk $(d + 2) \bmod n$. The value of y is constant in each iteration, and can be interpreted as the *stride* between consecutive client data units. In turn, each iteration contains $m = 2$ rows of data units, followed by $f = 2$ rows of check units. Within the group of check rows, the i-th row contains the i-th check unit of every stripe in the iteration. The i-th check unit of stripe s is mapped to the same disk as the i-th unit of stripe $(s + 1)$.

PRIME is very close to an ideal layout; its only deviation from optimality is that, during large reads, some disks may supply at most one data unit more than the optimal. Formally, its properties are:

Theorem 7 *For every prime n, every k with $2 \leq k \leq n$, every f with $0 \leq f < k$ and every r such that $(k(n-1))|r$, the layout* PRIME *is f-failure correcting, has optimally distributed check information, has optimally distributed reconstruction for a single failure, has large write optimization, and has parallel read count $\tau(o) \leq \lceil o/n \rceil + 1$ for all o.*

4.2 RELPR

PRIME may map two units from the same stripe to the same disk if n is not prime. One solution is to restrict all strides to be relatively prime to n. We call this layout RELPR (for *relatively prime*). The number of stripes in the layout is $b = n \cdot \phi(n)$, where ϕ (Euler's totient function) counts the number of positive integers less than n that are relatively prime to n. The mapping is a little more complicated than before when $\gcd(n, m) > 1$.

RELPR is based on a relaxation of the distributed reconstruction property. For layouts that satisfy this property optimally, there is a number λ such that every pair of disks share exactly λ stripes. The number λ is often expressed as a fraction $\alpha = \lambda/r$ of the number of units on a disk. If a layout has optimally distributed reconstruction, then $\alpha_{opt} = (k-1)/(n-1)$. In general, for layouts that do not have optimally distributed reconstruction, Schwabe and Sutherland [16] define the *reconstruction workload* α_L of a layout L to be the maximum, over all disks d and d', of the fraction of disk d' that must be read to reconstruct disk d. We prove that $\alpha_{Relpr}(n,k) \leq (k-1)/\phi(n)$ in the Appendix. Thus, the worst-case deviation from the optimal is

$$\frac{\alpha_{Relpr}(n,k)}{\alpha_{opt}(n,k)} \leq \frac{(n-1)}{\phi(n)}. \tag{7}$$

As described in Section 5.3, the actual deviation from optimal is significantly smaller than this bound for many values of n and k. Let $Y = \{\, y_0, y_1, \ldots, y_{\phi(n)-1} \mid$
$0 < y_l < n \wedge \gcd(y_l, n) = 1 \text{ for } 0 \leq l < \phi(n) \,\}$, and $g = \gcd(n, m)$. The client data unit with address a is mapped as follows:

$$\begin{aligned}
z &= \left\lfloor \frac{a}{mn} \right\rfloor && (8)\\
l &= z \bmod \phi(n) && (9)\\
j &= \left\lfloor \frac{a}{m \cdot (n/g)} \right\rfloor \bmod g && (10)\\
\textit{disk}(a) &= (a + j) y_l \bmod n && (11)\\
\textit{offset}(a) &= \lfloor a/n \rfloor + fz && (12)
\end{aligned}$$

Check unit i with $0 \leq i < f$ is at:

$$\begin{aligned}
\textit{check-disk}_i(a) &= (((\lfloor a/m \rfloor + 1)m + i + j) \\
&\quad \cdot y_l) \bmod n && (13)\\
\textit{check-offset}_i(a) &= (z+1)\,k - f + i && (14)
\end{aligned}$$

These formulas reduce to PRIME's location functions when n is prime. We show an example for $n = 6$, $k = 3$, $f = 1$:

D0.0	D0.1	D1.0	D1.1	D2.0	D2.1
D5.1	D3.0	D3.1	D4.0	D4.1	D5.0
C2	C5	C0	C3	C1	C4
D6.0	D8.1	D8.0	D7.1	D7.0	D6.1
D11.1	D11.0	D10.1	D10.0	D9.1	D9.0
C8	C10	C7	C9	C6	C11

RELPR's structure is similar to PRIME's. However, there are only $\phi(n)$ iterations in the layout in this case (when n is a prime number, $\phi(n) = n - 1$ as in the case of PRIME). So the strides that can be used are the elements of Y.

For this layout, we have $r = k \cdot \phi(n)$. RELPR deviates from the ideal in two ways: it can have parallel read count of at most one more than the optimal, and the reconstruction workload is not uniformly distributed.

Theorem 9 *For every n, every k with $2 \leq k \leq n$, every f with $0 \leq f < k$, and every r such that $(k \cdot \phi(n))|r$, the layout* RELPR *is f-failure correcting, has optimally distributed check information, has large write optimization, has reconstruction workload bounded above by $(k-1)/\phi(n)$ for a single failure, and has parallel read count $\tau(o) \leq \lceil o/n \rceil + 1$ for all o.*

5 Performance

We compare the performance of our layouts with Parity Declustering [11] and RAID-5 [15]. Parity Declustering was chosen as a typical representative of BIBD-based layouts; RAID-5 satisfies the maximal parallelism optimally, even though it is not a declustered layout.

Workload	
Access size:	Fixed at 8, 48, 120, 192 KB
Concurrency:	From 1 to 20 simulated clients
Alignment:	8 KB (stripe unit boundary)
Random accesses uniformly distributed over all data	
Array	
Stripe unit:	8 KB
Data layouts:	PRIME
	Parity Declustering
	Left-symmetric RAID-5
Number of disks, n:	13
Stripe width, k:	Prime, Parity Declustering: 4 units
	RAID-5: 13 units
Disk (HP 2247)	
Capacity:	1.03 GB
Geometry:	1981 cylinders, 13 heads, 8 zones
Average seek time:	10 ms.
Rotational speed:	5400 RPM (11.12 ms./rev.)
Head scheduling:	SSTF on 20-request queue

Table 1: Simulation parameters.

Our experiments were run on RAIDframe [6], a disk array architecture testbed for run-time performance evaluation. RAIDframe and its predecessor Raidsim have been used a number of times in the literature [2, 6, 11]. Table 1 shows the parameters for the array and disk simulator. In each experiment, the workload serviced by the array consists of interleaved streams of fixed-size *logical accesses* of the same type (read or write), originated by a fixed number of simulated clients. Workloads are synthetic: each simulated client generates an independent logical access starting at a random location with uniform distribution, blocks until the array services it, and immediately repeats the cycle. Logical accesses span an integer number of stripe units, and are aligned to a stripe unit's boundary. Traces or synthetic workloads with a more realistic access mix would be a better predictor of the performance of the arrays in a real situation. However, our goal is to analyze the strengths and weaknesses of the three examined architectures with respect to each other. Several subtle interactions and tradeoffs would have been considerably more difficult to understand with less homogeneous workloads.

Logical accesses are translated by the array controller into sets of *physical accesses* on the individual disks. Our simulated arrays always have 13 disks. Usable storage capacity varies according to the layout: RAID-5 uses 7.7% of the disks for parity, while PRIME and Parity declustering have a parity overhead of 25% in our configuration. RAID-5 and Parity Declustering can tolerate one failure by definition; the configuration of PRIME used in our experiments can tolerate a single failure as well. Each disk controller performs dynamic request reordering, following the shortest-seek-time-first (SSTF) policy to choose the next request to be serviced from the disk's queue.

Our principal performance metric is *average response time*: the average time elapsed from the moment a client requests a logical access, to the moment the array completes the access. Given that both the number of clients and the size of logical accesses are fixed during each experiment, throughput can be easily calculated from the response times reported here: Throughput = Request_Size * Number_of_Clients / Response_Time. Experiments run until the measured average response time is within 2% of the true average with 95% confidence. Additional experiments for non-prime values of n showed that the only differences in response times between PRIME and RELPR are due to the different array sizes—as predicted by our theoretical results. Therefore, we discuss the performance of PRIME in the remainder of this section, as it is very similar to RELPR's.

For a given logical access l, we define the *working set size* as the number of disks that have to perform at least one physical access in order to process l. Figure 1 shows the average working set size for the three layouts, for four different access sizes: 1, 6, 15, and 24 consecutive stripe units. For each access size we consider reads and writes separately, and for each access type we consider both failure-free and degraded modes (when one disk has crashed and has not been reconstructed). RAID-5 satisfies the maximal parallelism property optimally; as discussed in Sections 4.1 and 4.2, PRIME has a maximum deviation of one from the optimal and Parity Declustering does not even satisfy the relaxed

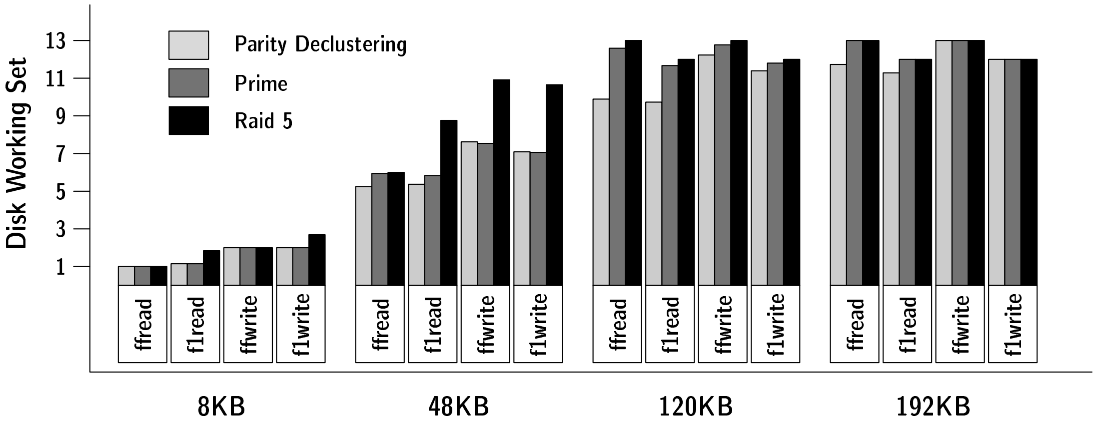

Figure 1: Average working set sizes. Computed by averaging the working set sizes for logical accesses starting at every possible offset in the array. The prefix "ff" denotes failure-free mode, while the prefix "f1" means that a single disk has crashed.

constraint. This can be observed in Figure 1: for failure-free reads when no parity unit must be accessed, working sets for RAID-5 are larger than for PRIME. In turn, working sets for PRIME are larger than for Parity Declustering. This is also true for the other three cases. RAID-5 peaks out at the maximum possible values (13 disks for failure-free mode, 12 disks for degraded mode) for smaller access sizes than the other layouts. In fact, Parity Declustering does not reach these maximum values for any read size in the figure. Since the average number of physical accesses per logical access is the same for any declustered layout with the same values of n and k, layouts with smaller working sets map more accesses to each disk. Accesses mapped to the same disk are serviced serially in some order; therefore, larger working sets imply fewer serialized requests.

5.1 Reads

Figure 2 shows the response times for failure-free read accesses. In the 8KB case, performance is very similar for the three layouts. However, we observe the following behavior for larger accesses: Parity Declustering is slower than Prime when the array is lightly loaded, but as load increases the curves eventually cross and Parity Declustering becomes faster. For 48KB, Parity Declustering is up to 27% slower than Prime for low and medium access rates, but Prime becomes 7.5% slower than Parity Declustering for the highest load.

This effect occurs for all failure-free reads smaller than 120KB. To understand it, we need to look at the head positioning overhead in detail—in particular, seek overhead. Figure 3 shows the average number of seeks per logical access measured during our simulations. Seek counts are largely independent of the request arrival rate. Parity declustering results in fewer seeks than PRIME for access sizes smaller than 120KB; from that point on, PRIME has a smaller seek overhead, and the difference increases with the access size. Therefore, the point after which PRIME is doing fewer seeks than Parity Declustering is the same point after which response time curves do not cross anymore.

To analyze this seek count inversion in greater detail, we classify seeks into two varieties. *Local seeks* occur between physical accesses caused by the same logical access, while *non-local seeks* occur between physical accesses caused by different logical accesses. We define a *locality* as the set of disk addresses of all physical accesses caused by the same logical access and mapped to the same disk. Because the starting addresses of logical accesses are uniformly distributed over the address space of the array, most switches between different localities require a non-local seek in our experiments. Figure 4 refines the seek counts for the failure-free case. Local seeks are further categorized according to whether there was a head switch to another surface within the same cylinder (incurring a resettling time of 0.8 ms. in our simulation) or if the head ensemble was actually moved to the adjacent cylinder (2.9 ms.). All local seeks belong to one of the two categories in the experiments discussed here.

Even though they were measured independently, the numbers of non-local seeks in Figure 4 and the working set sizes from Figure 1 are equal. This is because non-local seeks occur when, and only when, a disk has to service the first physical access for a particular logical access. Because of the SSTF head scheduling policy, disks seldom leave a locality before all its accesses are finished. Therefore, the layout with the larger working set (PRIME) incurs more

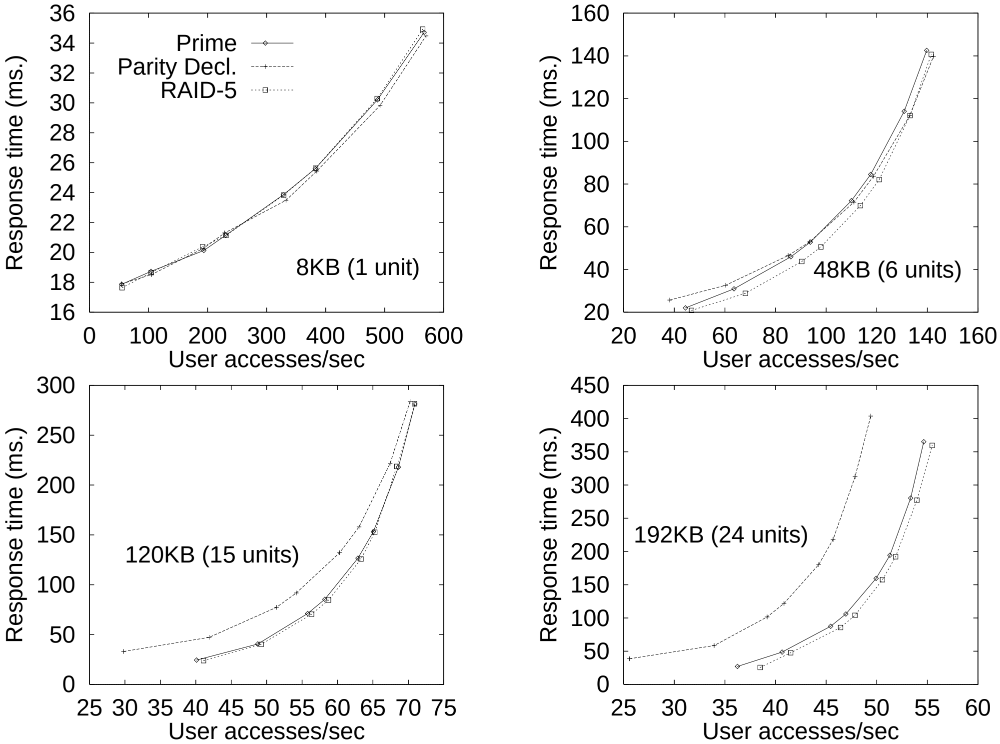

Figure 2: Read response times: Failure-free mode.

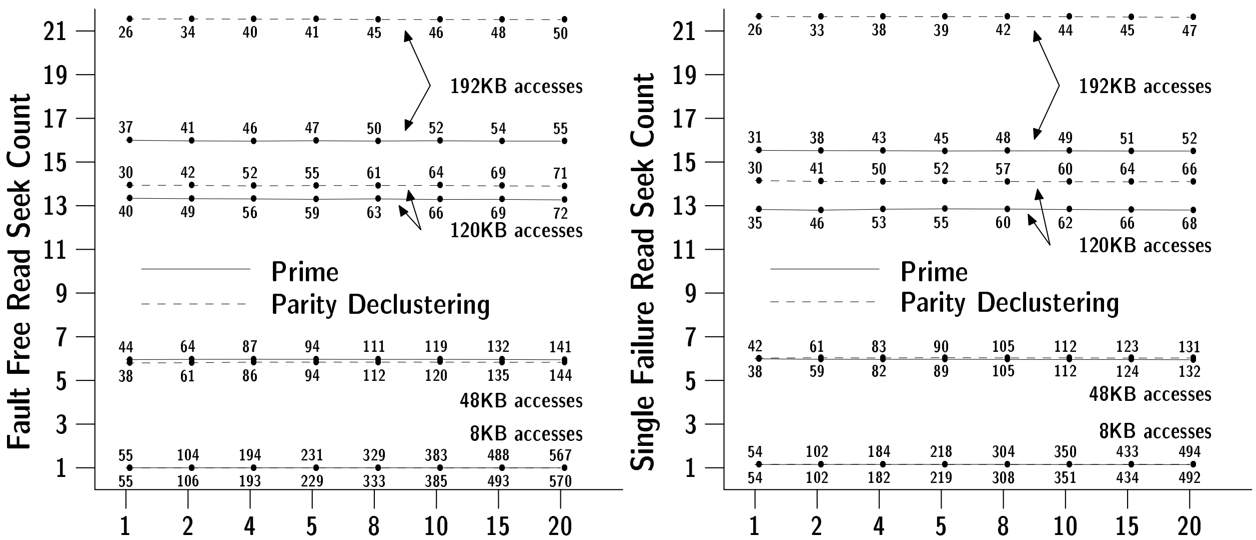

Figure 3: Read seek counts. These plots show the average number of seeks per logical access, for both failure-free and degraded modes, as a function of the number of clients issuing concurrent requests to the array. Each data point is labeled with the average number of logical accesses per second sustained during each experiment.

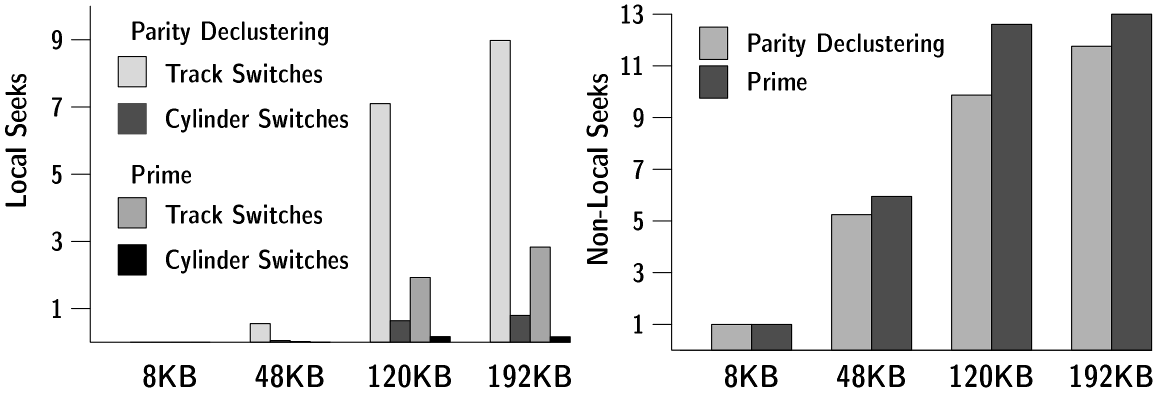

Figure 4: Seek types for failure-free reads.

non-local seeks during the execution. However, the number of local seeks grows much faster with access size for Parity Declustering than for PRIME. The total seek count is the sum of the number of local and non-local seeks, and 120 KB is precisely the smallest access size for which the difference in the number of local seeks more than offsets the non-local seek disadvantage of PRIME.

At 8KB per stripe unit, the actual data transfer time is negligible compared with the head positioning overhead. Therefore, the layout incurring the smaller number of seeks has smaller response times. There is an exception to this rule: when the array is experiencing a light load, disks are relatively idle and can respond to access requests within a short time interval. In this case, a larger working set implies better performance because it takes better advantage of the available bandwidth by performing more parallel accesses. As the load increases, it becomes increasingly more costly to involve all the disks that must cooperate to service a logical access, and the seek count determines the response times. We see in our results that PRIME has better performance than Parity Declustering for light and moderate loads, even in the cases where Parity Declustering is doing fewer seeks. For accesses of 120KB or larger, PRIME has superior performance for every load, as it has a (sometimes considerably) smaller seek count in addition to larger working sets. For accesses of 192KB, Parity Declustering is 178% slower than PRIME for one client; the difference is 166% for the maximum of 20 clients.

Similar arguments can be made for degraded-mode reads. The only differences in the relative performance of PRIME and Parity Declustering are quantitative. The same considerations apply to RAID-5. Compared to PRIME, RAID-5 has better performance in failure-free mode because it satisfies the maximum parallelism property optimally. However, when considering degraded-mode operation, RAID-5's performance degrades significantly. In non-declustered layouts, every time a read is attempted from a failed disk, it is necessary to read units from all $(n-1)$ disks to reconstruct the missing data. Therefore, the workload on the surviving disks roughly doubles during degraded-mode operation. In the same situation, declustered layouts only require $(k-1)$ reads; this improves both response times and the highest load that the array can sustain. To summarize, PRIME provides the best overall performance for reads, except for smaller access sizes under high loads. By using read-ahead techniques such as the prefetching policies discussed in [18], PRIME can deliver very significant performance improvements.

5.2 Writes

Figure 6 shows the response times for failure-free writes. Again, response times are very similar for the three layouts in the 8KB case. For all larger accesses, PRIME shows better performance than Parity Declustering—and the difference increases with the access size. Figure 7 shows the average number of seeks per logical access measured during our simulations. PRIME's working sets are larger or at least equal to Parity Declustering's. Unlike in the read case, seek counts are always larger for Parity Declustering. Since both effects favor PRIME, its response times are substantially better for all loads—and the difference increases with access size. For 120KB accesses, Parity Declustering is 28% slower under light loads and 64% under high loads; for 192KB accesses, the differences are 66% and 181%, respectively. PRIME can process a much higher write load using the same hardware.

RAID-5 has much higher response times than the declustering layouts for 48KB accesses. This is because, even though all three layouts satisfy the large write optimization, the stripe size is 13 for RAID-5, compared with 4 for

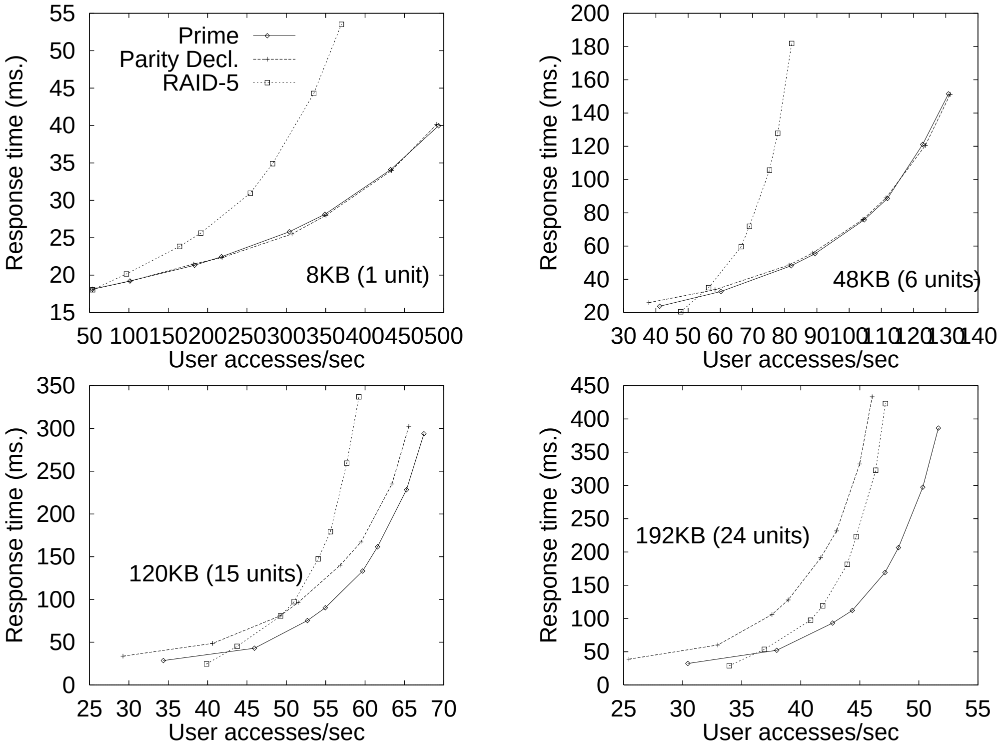

Figure 5: Read response times: Degraded mode.

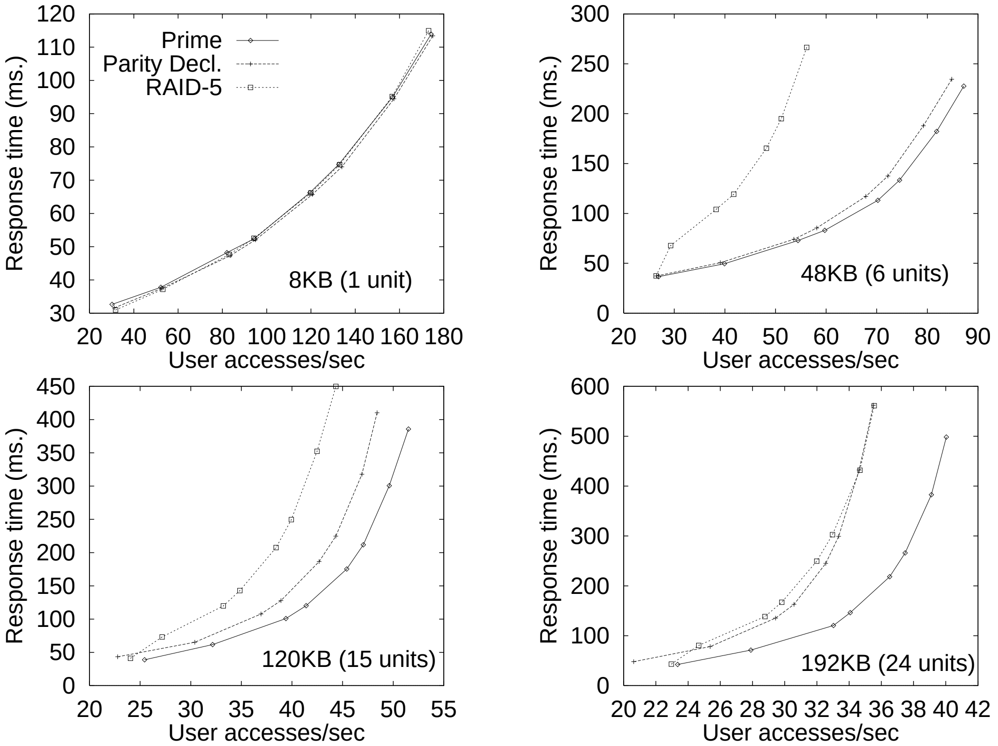

Figure 6: Write response times: Failure-free mode.

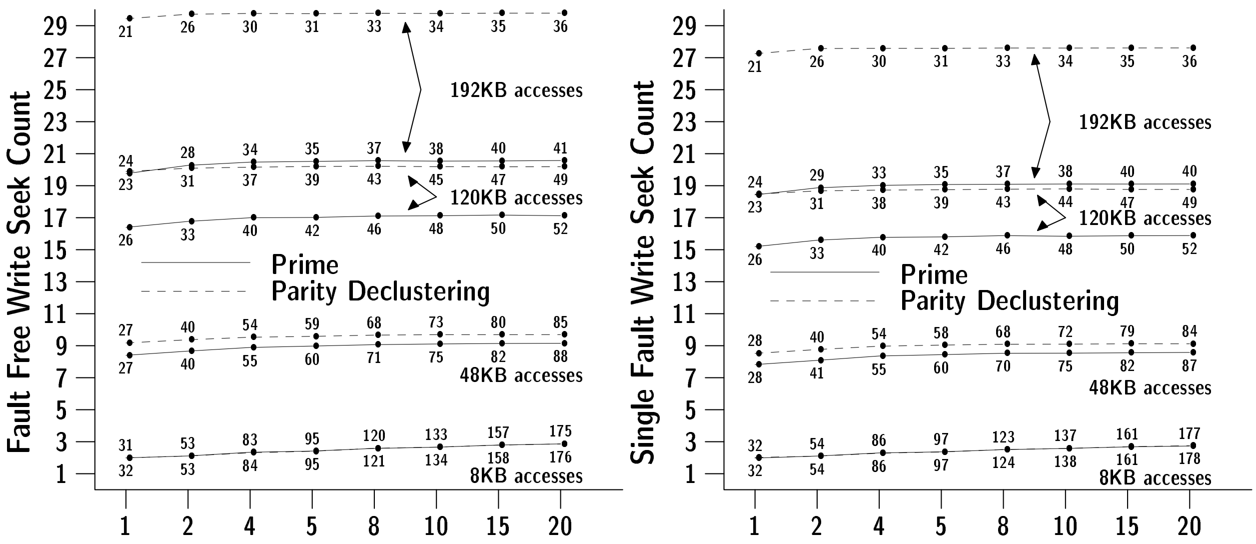

Figure 7: Write seek counts.

PRIME and Parity Declustering. Therefore, writes to a whole stripe will occur much more often for the declustered layouts than for RAID-5. In fact, RAID-5 is implementing all its writes as "small writes" in the 48KB case—that is, reading the old values of data and parity, using them to compute the new parity, and writing the new data and parity. It is easy to see that this requires at least twice as many physical accesses than full-stripe writes. This effect is less pronounced for larger access sizes, but PRIME has a substantial performance advantage in all our experiments.

Regarding degraded mode, response times for PRIME and Parity Declustering are slightly better than in the failure-free case (by a very small margin). This is a well-known phenomenon: when performing large writes in degraded mode, the disks actually do *less* work in most cases, because they do not have to write to the failed disk. The RAID-5 response times degrade with respect to failure-free mode, more significantly for smaller access sizes. For 8KB and 48KB accesses, all logical writes span less than half a stripe, so RAID-5 would ideally implement them as small writes. However, when one of the disks containing modified data has crashed, every logical write must be implemented as a "large write"—that is, reading the data units that will *not* change instead of the ones that will. Therefore, while large writes never occur for RAID-5 in failure-free mode, they are quite common in degraded mode. Moreover, since no logical write involves more than half the data units of the stripe, implementing it as a large write results in more physical reads. The average number of physical accesses per logical write is therefore higher in degraded mode. This effect is less pronounced for larger access sizes, when some large writes do occur even in failure-free mode.

5.3 Reconstruction Access Distribution

We evaluated the real deviation from the optimal reconstruction access distribution for a series of examples. Figure 9 shows the results of calculating it for a variety of n's and k's and comparing it with the optimal value $(k-1)/(n-1)$. The table contains data for three "target values" of the reconstruction workload, .25, .5 and .75. That is, for a given target value α_{targ}, the values of n and k are chosen so that $\alpha_{opt}(n,k) = (k-1)/(n-1)$ is as close as possible to, but never more than, α_{targ}. For a given n, this is accomplished by setting $k = \lfloor \alpha_{targ}(n-1) \rfloor + 1$. The measure of how well RELPR meets the target for a given n (and derived k) is the ratio of the actual calculated reconstruction workload $\alpha_{Relpr}(n,k)$ to the optimal value $(k-1)/(n-1)$.

The bound 7 is often not very tight. For example, if $n = 32$, then $(n-1)/\phi(n) = 31/16 \approx 1.94$; although, the actual value of $\alpha_{Relpr}(n,k)/\alpha_{opt}(n,k)$ is less than 1.3 for all $k \geq 3$, and less than 1.04 for all $k \geq 15$. We observe that as either n or k increases, generally speaking, the deviation from the ideal diminishes as well. As expected, the deviations from the ideal are larger when $\phi(n)$ is smaller relative to n.

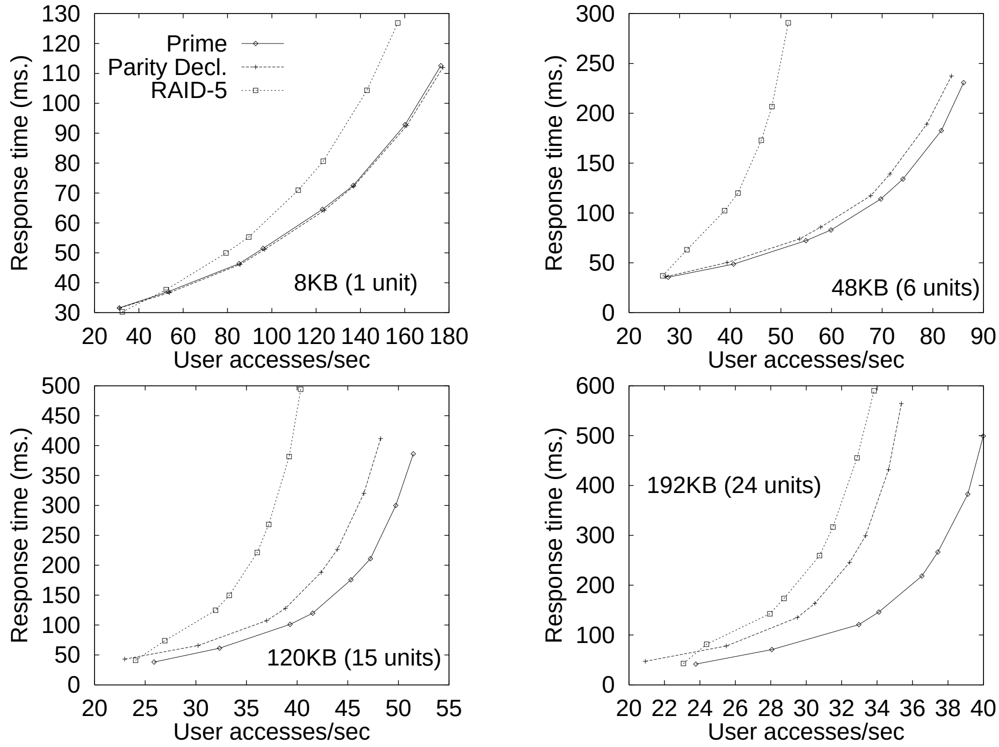

Figure 8: Write response times: Degraded mode

6 Conclusions

In this paper we have characterized the declustering disk arrays that admit an ideal layout satisfying all six criteria identified by Holland and Gibson. We actually constructed the layouts during our existential proof. To the best of our knowledge, no previously published layout satisfies all criteria. Unfortunately, optimal solutions turned out to be possible only for a very restricted set of configurations: except for three special cases, ideal layouts are only possible for very high declustering ratios or for a version of declustered mirroring.

By relaxing some of the initial constraints, a much larger variety of configurations can be accommodated. If an imbalance of one request during large reads is tolerated, the PRIME layouts can satisfy all other conditions optimally for a prime number of disks. For the general case, the RELPR layouts satisfy four of the conditions, relaxing the parallelism as above and having a slightly suboptimal distribution of reconstruction accesses.

We evaluated the performance of the new layouts on a simulation testbed under a variety of access sizes and failure scenarios. The results show that the new layouts have considerably better performance than previously-existing alternatives for large accesses, where throughput is typically improved by a factor of approximately 2.5. For read accesses smaller than 120KB, our layouts have a slight performance disadvantage under heavy loads. PRIME and RELPR also outperform their competitors during degraded-mode operation when a failure has occurred. Non-declustered layouts like RAID-5 perform well for reads in failure-free mode, but there is a significant performance degradation in response times for writes and for degraded-mode operation. Moreover, the duration of online reconstruction of a failed disk is substantially longer for RAID-5, resulting in a more significant degradation of the quality of service and in longer time intervals of vulnerability to unrecoverable failure [11].

Acknowledgements: We thank Jai Menon for helpful discussions on issues of data layout for disk arrays, William Courtright for his help with RAIDframe, and John Hobby for creating MetaPost.

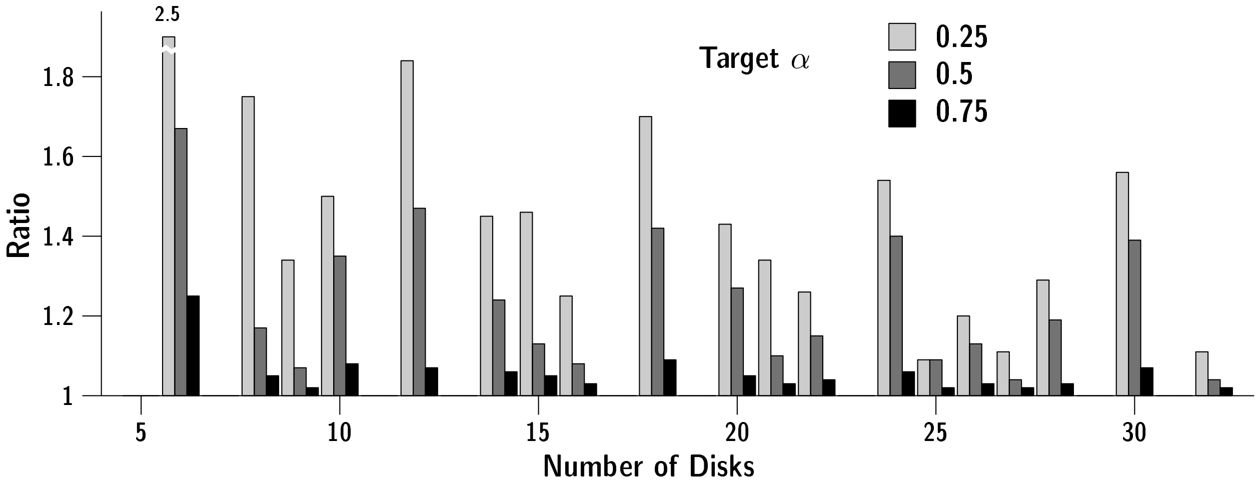

Figure 9: The ratio $\alpha_{Relpr}/\alpha_{opt}$ for three target values of α

References

[1] G. Alvarez and W. Burkhard. Stride-based layouts for disk array declustering. Technical report CS97-549, UCSD–CSE, May 1997.

[2] G. Alvarez, W. Burkhard, and F. Cristian. Tolerating multiple failures in RAID architectures with optimal storage and uniform declustering. In *Proc. of the International Symposium on Computer Architecture*, pages 62–72, 1997.

[3] M. Blaum, J. Brady, J. Bruck, and J. Menon. Evenodd: An efficient scheme for tolerating double disk failures in RAID architectures. In *Proc. of the International Symposium on Computer Architecture*, pages 245–54, 1994.

[4] W. Burkhard and J. Menon. Disk array storage system reliability. In *Proc. of the International Symposium on Fault-tolerant Computing*, pages 432–41, 1993.

[5] P. Chen and E. Lee. Striping in a RAID level 5 disk array. In *Proc. of ACM SIGMETRICS*, pages 136–45, 1995.

[6] W. Courtright, G. Gibson, M. Holland, and J. Zelenka. A structured approach to redundant disk array implementation. In *Proc. of the International Symposium on Performance and Dependability*, pages 11–20, 1996.

[7] G. Gibson, L. Hellerstein, R. Karp, R. Katz, and D. Patterson. Coding techniques for handling failures in large disk arrays. In *Proc. of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 123–32, 1989.

[8] G. Gibson, D. Nagle, K. Amiri, F. Chang, E. Feinberg, H. Gobioff, C. Lee, B. Ozceri, E. Riedel, D. Rochberg, and J. Zelenka. File server scaling with network-attached secure disks. In *Proc. of ACM SIGMETRICS*, pages 272–84, 1997.

[9] M. Hall. *Combinatorial Theory*. Wiley, New York, 1986.

[10] H. Hanani. Balanced Incomplete Block Designs and Related Designs. *Discrete Mathematics*, 1975.

[11] M. Holland and G. Gibson. Parity declustering for continuous operation on redundant disk arrays. In *Proc. of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 23–35, 1992.

[12] Mark Calvin Holland. *On-Line Data Reconstruction In Redundant Disk Arrays*. PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1994.

[13] E. Lee and R. Katz. Performance consequences of parity placement in disk arrays. In *Proc. of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 190–99, 1991.

[14] R. Muntz and J. Lui. Performance analysis of disk arrays under failure. In *Proceedings of the 16th VLDB Conference*, pages 162–73, 1990.

[15] D. Patterson, G. Gibson, and R. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proc. of the ACM SIGMOD International Conference on Management of Data*, pages 109–16, 1988.

[16] E. Schwabe and I. Sutherland. Improved parity-declustered layouts for disk arrays. In *Proc. of the Symposium on Parallel Algorithms and Architectures*, pages 76–84, 1994.

[17] L. Stockmeyer. Parallelism in parity-declustered layouts for disk arrays. Technical Report RJ 9915 (87295), IBM Almaden Research Center, November 1994.

[18] A. Tomkins, H. Patterson, and G. Gibson. Informed multi-process prefetching and caching. In *Proc. of ACM SIGMETRICS*, pages 100–14, 1997.

[19] X. Wu, J. Li, and H. Kameda. Reliability analysis of disk array organizations by considering uncorrectable bit errors. In *Proc. of the 16th Symposium on Reliable Distributed Systems*, pages 2–9, October 1997.

Appendix: Definitions and Proofs

A Formal Definition of Placement-Ideal Layouts

These are five of the six desirable properties of layouts described by Holland and Gibson [11]. For some of the properties, we also give measures of the "goodness" of a layout for cases where the property is not optimally satisfied; two of these measures were defined in [16].

1. *Failure Correcting.* In order to be able to reconstruct the contents of up to f failed disks, it is necessary that no two units on the same stripe are mapped to the same disk. That is, for every stripe s $(0 \leq s \leq b-1)$ and every u and u' with $0 \leq u < u' < k$, $\mathit{disk}((s,u)) \neq \mathit{disk}((s,u'))$.

2. *Distributed Check Information.* A layout has *optimally distributed check information* if every disk contains the same number of check units. That is, there is a number ρ_{opt} such that, for every disk d $(0 \leq d < n)$, there are exactly ρ_{opt} stripes s such that there exists an i with $0 \leq i < f$ such that $\mathit{disk}((s, k-f+i)) = d$. Since there are a total of bf check units, if they are optimally distributed we must have $\rho_{opt} = bf/n$ $(= rf/k)$, so n must divide bf (so k must divide rf). In general, the *check overhead* of a layout is defined in [16] to be the maximum, over all disks d, of the fraction of the locations on disk d that contain check units. Check information is optimally distributed iff the check overhead is f/k.

3. *Distributed Reconstruction.* If disk d fails then, for every other disk d', the number of units that must be read from disk d' to reconstruct the contents of disk d is the number of stripes that contain units mapped to both d and d'. Disks d and d' *share* stripe s if there are u and u' with $0 \leq u, u' \leq k-1$ such that $\mathit{disk}((s,u)) = d$ and $\mathit{disk}((s,u')) = d'$. A (single-failure-correcting) layout has *optimally distributed reconstruction* if there is a number λ_{opt} such that, for every pair of disks d and d' $(0 \leq d < d' \leq n-1)$, disks d and d' share exactly λ_{opt} stripes. Assuming that the layout is single-failure-correcting, it is known that $\lambda_{opt} = r(k-1)/(n-1)$. (To see this, note that r stripes have some unit mapped to disk d, so $r(k-1)$ units of these stripes are mapped to other disks. Therefore, on the average, each of the $n-1$ other disks share $r(k-1)/(n-1)$ stripes with disk d. If the number of stripes shared is the same for every pair of disks, each disk d' with $d' \neq d$ must share exactly the average number with disk d.) The number λ_{opt} is often expressed as a fraction $\alpha_{opt} = (k-1)/(n-1)$ of the number r of units on a disk, and called the *declustering ratio* [11]. In general, the *reconstruction workload* of a layout is defined in [16] to be the maximum over all pairs of disks d and d' $(0 \leq d < d' \leq n-1)$ of $\lambda(d,d')/r$, where $\lambda(d,d')$ is the number of stripes s such that d and d' share stripe s. In analogy with α_{opt}, we denote the reconstruction workload of a layout L by α_L. So L has optimally distributed reconstruction iff $\alpha_L = (k-1)/(n-1)$.

4. *Large Write Optimization.* A layout has the *large write optimization* property if the space of client data addresses is broken into b "stripe intervals", each stripe interval containing $m = k - f$ consecutive addresses, such that each stripe interval is mapped to the same stripe. Since stripe names are arbitrary and the names of the data units in a stripe are arbitrary, we can define a layout to have this property if, for every client data address a,

$$\mathit{stripe}(a) = (\lfloor a/m \rfloor,\ a \bmod m). \tag{15}$$

   That is, the stripe intervals of client data are mapped to stripes in order of increasing stripe number, and the units within a stripe interval are mapped to the stripe in order of increasing unit number within the stripe.

5. *Maximal Parallelism.* A layout has *maximal parallelism* if the addresses of client data in any interval $\{t, t+1, \ldots, t+n-1\}$ of n consecutive addresses are mapped to n different disks, so that these n units of data can be read in parallel. That is, for every starting point t with $0 \leq t \leq bm-n$ and every i and i' with $0 \leq i < i' \leq n-1$, we have $\mathit{disk}(t+i) \neq \mathit{disk}(t+i')$. In general, for a layout L we define the function $\tau_L(o)$, the *parallel read time*, to be the maximum number of time units needed to read in parallel any o consecutive units of client data. That is, $\tau_L(o)$ is the maximum, over all disks d and all t with $0 \leq t \leq bm-o$, of the number of i's with $0 \leq i \leq o-1$ and $\mathit{disk}(t+i) = d$. For any o between 1 and the number bm of client data units, it must be that $\tau_L(o) \geq \lceil o/n \rceil$. Say that a layout L has *optimal parallelism* if $\tau_L(o) = \lceil o/n \rceil$ for all o. It follows from Lemma 2 that a layout has maximal parallelism iff it has optimal parallelism.

A layout is *placement-ideal* if it satisfies all five of these properties optimally. Note that the first three properties concern only the mapping of stripes to disks, the fourth concerns only the mapping of client data to stripes, and the fifth concerns only the (implied) mapping of client data to disks. Large write optimization can always be achieved if we are willing to ignore parallelism. Moreover, if the layout has optimally distributed check information, it is easy to see that optimal parallelism can always be achieved if we are willing to ignore large write optimization.

Since the distributed reconstruction property is only defined for layouts that tolerate a single failure, our characterization of ideal layouts (done in the first part of this appendix) will only be applicable to layouts tolerating a single failure as well. However, the PRIME and RELPR almost-ideal layouts can tolerate an arbitrary number $f < k$ of failures as shown in Sections 4.1 and 4.2. For simplicity and to unify this presentation, we will only discuss the case $f = 1$ in the remaining part of this appendix. The proofs for PRIME and RELPR can be generalized in a straightforward manner to arbitrary values of f. Following common usage, we will speak of "parity" instead of "check" for the most usual variant of $f = 1$.

B Possibility and Impossibility of Placement-Ideal Layouts

First, we review some simple definitions and facts from number theory. For positive integers x and y, the notation $x|y$ means that x divides y (i.e., $y = zx$ for some integer z). The greatest common divisor of x and y, denoted $\gcd(x, y)$, is the largest integer g such that $g|x$ and $g|y$. Integers x and y are *relatively prime* if $\gcd(x, y) = 1$. The following elementary fact will be used often.

Fact 1 *If y and n are relatively prime and i and j are integers with $1 \leq i - j \leq n - 1$, then $iy \not\equiv jy \bmod n$.*

It follows from this fact, for example, that if y and n are relatively prime, then the residues $iy \bmod n$ for $t \leq i \leq t + n - 1$ are all distinct, for any integer t. In this section we completely determine the combinations of parameters n, k, r for which placement-ideal layouts are possible. Our main focus is on determining the combinations of n and k that allow placement-ideal layouts. Regarding r, as noted in the discussions of distributed parity and distributed reconstruction, if there is a placement-ideal layout with parameters n, k, r, then $k|r$ and $(n-1)|(r(k-1))$; we call these the *divisibility conditions*. For the combinations of n and k that allow placement-ideal layouts, we show that there is a placement-ideal layout for any r meeting the divisibility conditions (in particular, any r that is a multiple of $k(n-1)$). For the combinations of n and k that do not allow placement-ideal layouts, for no r is there a placement-ideal layout with parameters n, k, r.

We begin by showing that the large write optimization property and the maximal parallelism property together determine how stripe data units are mapped to disks. The first step is to show that the maximal parallelism property determines how client data is mapped to disks, up to a renaming of the disks.

Lemma 2 *If a layout has the maximal parallelism property, then there is a way to name the disks $0, 1, \ldots, n-1$ such that disk$(a) = a \bmod n$ for every address a of client data.*

Proof. By maximal parallelism, the first n units of client data, (with addresses $0, 1, \ldots, n-1$) are mapped to different disks. Rename the disks, if necessary, so that *disk*$(a) = a$ for $0 \leq a \leq n-1$. Consider the following statement:

$$\textit{disk}(a) = a \bmod n \text{ for all } a \text{ with } 0 \leq a \leq A. \tag{16}$$

The rest of the proof is by induction on A. The basis $A = n-1$ holds by the way the disks are renamed. Assume that (16) holds for some A with $n - 1 \leq A \leq b(k-1) - 2$. We prove that it holds for $A + 1$. By maximal parallelism, the n addresses a for $A - n + 1 \leq a \leq A$ must be mapped to n different disks, and the n addresses a for $A - n + 2 \leq a \leq A + 1$ must be mapped to n different disks. So we must have *disk*$(A+1) =$ *disk*$(A-n+1)$. Since *disk*$(A-n+1) = A - n + 1 \bmod n$ by induction, and $A + 1 \equiv A - n + 1 \bmod n$, this completes the proof. □

Lemma 2 and our convention (15) for mapping client data to stripes determine how stripe data units are mapped to disks. The sequence of stripe data units with addresses in the order $(0,0), (0,1), \ldots, (0,k-2), (1,0), (1,1), \ldots, (1,k-2), (2,0) \ldots$ must be mapped to the disks in round-robin order. This fact is stated in the next lemma, which is key to our proofs of impossibility of placement-ideal layouts.

Lemma 3 *If a layout has the large write optimization and maximal parallelism properties, then (for some naming of the disks and the stripe units) the mapping of stripe data units to disks is given by disk$((s,u)) = ((k-1)s + u) \bmod n$ for all s and all u with $0 \leq u < k-1$.*

Thus, in a placement-ideal layout, the mapping of stripe data units to disks is fixed; the only freedom is in the mapping of stripe parity units.

Another consequence of Lemma 2 is that repeating a placement-ideal layout produces a placement-ideal layout. This fact is used in constructions showing possibility of placement-ideal layouts.

Lemma 4 *If L' is a placement-ideal layout and if L is obtained by repeating L' any number of times, then L is placement-ideal.*

Proof. Clearly, L is single failure correcting and has large write optimization. It is easy to see that the parity overhead and the reconstruction workload of L are identical to those of L', so if L' is optimal in one of these measures then L is also. It remains only to consider optimal parallelism. Let L' have parameters n, k, r' and let $b' = r'n/k$ be the number of stripes. Since L' has optimally distributed parity, n divides b'. So n divides $b'(k-1)$, the number of units of client data mapped by L'. So for each copy of L', the last unit of client data mapped by the copy has address congruent to $(n-1) \bmod n$, and the first unit mapped by the copy has address congruent to $0 \bmod n$. It follows that *disk*$(a) = a \bmod n$ for the entire layout L, so L has optimal parallelism. □

We can now state and prove the characterization of when placement-ideal layouts are possible.

Theorem 5 *There is a placement-ideal layout with parameters n, k, r iff*

$$k|r \ \text{ and } \ (n-1)|(r(k-1))$$

and one of the following holds:

1. $k = n$,
2. $k = n-1$,
3. $k = 2$,
4. $k = 3$ *and* $n = 5$ *or* 7,
5. $k = 4$ *and* $n = 7$.

Proof. We begin by disposing of the case $k = n$, which is well known. If $k = n = r'$, the left-symmetric RAID5 layout [15] is placement-ideal. For any r with $k|r$ (i.e., $r'|r$), a placement-ideal layout is obtained by repeating this layout r/r' times.

In the rest of the proof, we assume that $k < n$. The proof is broken into several cases. In some of the cases, we let the integer $q = r/k$, so

$$r = qk, \ \ b = qn, \ \text{ and } \ \lambda_{opt} = \frac{qk(k-1)}{n-1}.$$

Of the $r = qk$ locations on each disk, $q(k-1)$ hold data units and q hold parity units. We also use the following terminology. We say that stripe s *starts at disk* d if *disk*$((s,0)) = d$. If stripe s starts at disk d and if the layout is placement-ideal, then by Lemma 3 the data units of stripe s are mapped to disks $d, d+1, \ldots, d+k-2 \bmod n$. We say that disks d and d' *share stripe s in data-data* if there are data units of stripe s mapped to d and to d'. Disks d and d' *share stripe s in parity-data* if a data unit of stripe s is mapped to one of d or d' and the parity unit of stripe s is mapped to the other one.

Case 1. $k \geq 3$.

Case 1.1. $\gcd(k-1, n) > 1$.

Case 1.1.1. $k = n-1$.

In this case, placement-ideal layouts are possible. Since $\gcd(k-1,n) > 1$ and $k-1 = n-2$, it must be that $\gcd(k-1,n) = 2$, and n and $k-1$ are both even. We describe a placement-ideal layout L' with parameters n, k, r' with $r' = k$. So there are $b' = n$ stripes, and $\lambda_{opt} = k-1$. The construction is illustrated in Figure 10 for $n = 6$. (In this figure and subsequent ones in this proof, we do not write the names of data units in stripes since they are determined by Lemma 3. Ds represents a data unit in stripe s, and Ps represents the parity unit.) In general, the mapping of stripe data units to disks in layout L' is determined by Lemma 3. For each k with $0 \leq k \leq (n/2) - 1$, there are two stripes that start at disk $2k$. (For every odd d, there is no stripe that starts at disk d.) Say that disks d are

D0	D0	D0	D0	D1	D1
D1	D1	D2	D2	D2	D2
D3	D3	D3	D3	D4	D4
D4	D4	D5	D5	D5	D5
P2	P5	P1	P4	P0	P3

Figure 10: A placement-ideal layout with $n = 6$ and $k = 5$.

D0	D0	D0	D1	D1
D1	D2	D2	D2	D3
D3	D3	D4	D4	D4
P4	P1	P3	P0	P2

Figure 11: A placement-ideal layout with $n = 5$ and $k = 4$.

$d+1$ are an *even pair* if d is even. For each even pair $d, d+1$, there are exactly two stripes that have no data unit mapped to either d or $d+1$, namely, the two stripes that start at disk $(d+2) \bmod n$. One of the parity units of these two stripes is mapped to d and the other parity unit is mapped to $d+1$ (in either order). It is not hard to see that (i) if d and d' belong to the same even pair then d and d' share $k-1$ stripes in data-data and share no stripes in parity-data; and (ii) if d and d' do not belong to the same even pair, then d and d' share $k-3$ stripes in data-data and share 2 stripes in parity-data, for a total of $k-1$. Therefore, L' is placement-ideal.

For any r with $k|r$ (i.e., $r'|r$), the layout L' can be repeated to obtain a placement-ideal layout with parameters n, k, r.

Case 1.1.2. $k \leq n-2$.

In this case, placement-ideal layouts are impossible. Suppose that a placement-ideal layout L with parameters n, k, r exists. Let $q = r/k = b/n$. Let $\gcd(k-1, n) = g \geq 2$. By Lemma 3, for every i with $0 \leq i < n/g$, there are gq stripes that start at disk ig. We first argue that disks 0 and 1 share (in data-data) every stripe that starts at disk 0, or disk $n-g$, or disk $n-2g$, or ..., or disk $n-((k-1)/g-1)g$. To see this, it is enough to show it for the smallest non-zero starting point $d = n-((k-1)/g-1)g$. If a stripe s starts at this disk d, then the data unit $(s, k-2)$ is mapped to disk $d' = (d+(k-2)) \bmod n$. Substituting the value of d into the expression for d', a simple calculation shows that $d' = g-1 \geq 1$, so disks 0 and 1 share stripe s. So the number of stripes shared by disks 0 and 1 is at least $((k-1)/g)gq = q(k-1)$. But since $k < n-1$,

$$q(k-1) > \frac{qk(k-1)}{n-1} = \lambda_{opt},$$

so L does not have optimally distributed reconstruction.

Case 1.2. $\gcd(k-1, n) = 1$.

It follows from Lemma 3 and Fact 1 in this case that, if the layout is placement-ideal, then for every disk d ($0 \leq d \leq n-1$) there are q stripes that start at disk d.

Case 1.2.1. $k = n-1$.

In this case placement-ideal layouts exist. We describe a placement-ideal layout with parameters n, k, r' where $r' = k$ (and $b' = n$). Stripe unit (s, u) is mapped to disk $((k-1)s+u) \bmod n$ for $0 \leq s \leq n-1$ and $0 \leq u \leq k-1$. The layout is illustrated in Figure 11 for $n = 5$. Since $\gcd(k-1, n) = 1$, it follows from Fact 1 that each disk holds exactly one parity unit, so parity is optimally distributed. Note that for each stripe s, there is exactly one disk d that does not contain a unit of stripe s (namely, $d = ((k-1)s-1) \bmod n$); call this d the "hole" of stripe s. For any two disks d and d', the two disks share every stripe s such that neither d nor d' is the hole of s. The number of such stripes

D0	D0	D1	D1	D2
D2	D3	D3	D4	D4
P4	P2	P0	P3	P1
D5	D5	D6	D6	D7
D7	D8	D8	D9	D9
P6	P9	P7	P5	P8

Figure 12: A placement-ideal layout with $n = 5$ and $k = 3$.

D0	D0	D1	D1	D2	D2	D3
D3	D4	D4	D5	D5	D6	D6
P2	P6	P3	P0	P4	P1	P5

Figure 13: A placement-ideal layout with $n = 7$ and $k = 3$.

is $n - 2 = k - 1$ regardless of the choice of d and d', so the layout has optimally distributed reconstruction. For any r with $k|r$, this layout is repeated r/k times.

Case 1.2.2. $k = n - 2$.

We first show that in this case there are no placement-ideal layouts if $n \geq 6$. Suppose that L is a placement-ideal layout with parameters n, k, r where $k = n - 2$ and $n \geq 6$ (so $k \geq 4$). Let $q = r/k = b/n$. We claim that disks 0 and 3 share at most $q(n - 6)$ stripes in data-data. A stripe that starts at disk 1, 2, or 3 has no data unit mapped to disk 0. A stripe that starts at disk 4, 5, or 6 $\bmod n$ has no data unit mapped to disk 3. Recall, as noted above, that in every case with $\gcd(k - 1, n) = 1$ there are q stripes that start at disk d, for every d. So disks 0 and 3 do not share in data-data any of the $6q$ stripes that start at disks 1 through 6 $\bmod n$. Since there are a total of qn stripes, this proves the claim that disks 0 and 3 share at most $q(n - 6)$ stripes in data-data. Since any disk contains exactly q parity units, disks 0 and 3 contain together $2q$ parity units, so they can share at most $2q$ stripes in parity-data. Therefore, disks 0 and 3 can share at most $q(n - 6) + 2q = q(n - 4)$ stripes. But

$$\lambda_{opt} = \frac{qk(k-1)}{n-1} = \frac{q(n-2)(n-3)}{n-1} > q(n-4),$$

so L does not have optimally distributed reconstruction.

Since $k \geq 3$ and $k = n - 2$, and since we have handled the case $n \geq 6$, the only other possibility under the current case is $n = 5$ and $k = 3$. This is one of the special cases where a placement-ideal layout is possible. A placement-ideal layout with $n = 5$, $k = 3$, and $r' = 6$ is shown in Figure 12. The divisibility conditions in this case are $3|r$ and $4|(2r)$. These together imply that $6|r$, so the layout of Figure 12 can be repeated.

Case 1.2.3. $k \leq n - 3$.

By an argument similar to ones that have already been given, it is not hard to see that, in any placement-ideal layout, disks 0 and 1 share at least $q(k - 2)$ stripes in data-data. Specifically, disks 0 and 1 share in data-data every stripe that starts at disk $-i \bmod n$ for $0 \leq i \leq k - 3$. We first show that placement-ideal layouts are impossible in this case if $k \geq 5$. Suppose a placement-ideal layout L does exist. Now

$$\lambda_{opt} = \frac{qk(k-1)}{n-1} \leq \frac{qk(k-1)}{k+2} < q(k-2).$$

The first inequality follows from $k \leq n - 3$, and the second (strict) inequality can be verified by calculation using $k \geq 5$. So L does not have optimally distributed reconstruction.

We next show that placement-ideal layouts are impossible in this case if $n \geq 8$ and either $k = 3$ or $k = 4$. This is true because it can be verified by calculation that, in these cases, $\lambda_{opt} = (qk(k - 1))/(n - 1) < q(k - 2)$.

D0	D0	D0	D1	D1	D1	D2
D2	D2	D3	D3	D3	D4	D4
D4	D5	D5	D5	D6	D6	D6
P1	P6	P4	P2	P0	P5	P3

Figure 14: A placement-ideal layout with $n = 7$ and $k = 4$.

Since $k \geq 3$ and $k \leq n - 3$, and since we have shown impossibility if $n \geq 8$, the only remaining cases are $(n, k) = (7, 3)$, $(7, 4)$, $(6, 3)$. The case $(n, k) = (6, 3)$ was shown impossible in Case 1.1.2, since $\gcd(2, 6) = 2$. Figure 13 shows a placement-ideal layout for $n = 7$ and $k = r' = 3$. Figure 14 shows a placement-ideal layout for $n = 7$ and $k = r' = 4$. These can be repeated for any r with $k|r$.

Case 2. $k = 2$.

In this case, placement-ideal layouts are possible for any n. We consider two subcases depending on whether n is odd or even. The subcases are very similar and both are based on the complete block design with $k = 2$.

First assume that n is odd. For every pair of disks d, d' with $0 \leq d < d' \leq n - 1$, there is a stripe mapped to disks d and d'. The number of stripes is $b' = n(n-1)/2$, so $r' = 2b'/n = n - 1$. This layout has optimally distributed reconstruction, since every two disks share one stripe. To assign the parity units, give each stripe a name of the form (x, y) where $0 \leq x < n$ and $1 \leq y \leq (n-1)/2$. The stripe with name (x, y) contains the two disks x and $(x + y) \bmod n$. This naming gives one copy of the complete block design with $k = 2$. To see this let d, d' be any two disk names with $d < d'$. If $d' - d \leq (n-1)/2$ then taking $x = d$ and $y = d' - d$ names the stripe. If $d' - d > (n-1)/2$ then take $x = d'$ and $y = d - d' + n$. For each stripe (x, y), let disk x contain the data unit and disk $(x + y) \bmod n$ contain the parity unit. Parity is optimally distributed because, for each fixed y, the parity disk $(x + y) \bmod n$ hits every disk exactly once as x ranges from 0 to $n - 1$. Since each disk contains the same number of data units (namely, $(n-1)/2$) and since each stripe contains only one data unit, it is clear that we can map client data units to stripe data units to achieve optimal parallelism. Large write optimization holds trivially, so this layout is placement-ideal. The divisibility conditions for $k = 2$ are $2|r$ and $(n-1)|r$, and this layout can be repeated for any r with $(n-1)|r$.

The case n even is very similar. The only difference is that we use two copies of the complete block design to produce a placement-ideal layout with $b' = n(n-1)$ and $r' = 2(n-1)$. Parity units are assigned as above, where now the names are (x, y) for $0 \leq x < n$ and $1 \leq y < n$. If n is even, the divisibility conditions $2|r$ and $(n-1)|r$ together imply that $(2(n-1))|r$, so the layout can be repeated for any r satisfying the divisibility conditions.

This completes the proof of the theorem. $\square$

For all of the cases in which placement-ideal layouts have been shown to exist, it should be clear (except possibly in the case $k = 2$) that the mapping functions can be efficiently computed. For the case $k = 2$, the mapping functions are similar to those described for the PRIME layouts in Section 4.1. (In this case, it is not needed that n is prime.) A difference here is that, since only one unit of each stripe contains client data, the mapping of client data addresses to disks can be *disk*$(a) = a \bmod n$ for all a, giving optimal parallelism. In the case n even, the mapping functions are

$$\begin{aligned}
z &= \lfloor a/n \rfloor \\
\textit{disk}(a) &= a \bmod n \\
\textit{offset}(a) &= 2z \\
y &= (z \bmod (n-1)) + 1 \\
\textit{check-disk}(a) &= (a + y) \bmod n \\
\textit{check-offset}(a) &= 2z + 1.
\end{aligned}$$

If n is odd, the only difference is that fewer values for y are used (that is, $1 \leq y \leq (n-1)/2$), so $(n-1)$ should be replaced by $(n-1)/2$ in the computation of y.

Because the mappings are efficiently computable in all cases for which placement-ideal layouts exist, we can state (at an informal level) that Theorem 5 holds with "ideal" in place of "placement-ideal".

C Almost Ideal Layouts

Since Theorem 5 shows that ideal layouts are possible only for very limited choices of n and k, it is reasonable to investigate how the situation changes if some of the properties are required to hold only approximately. In Section C.1, it is shown that for any prime n and any k, there are layouts that optimally satisfy all five of the properties, except that the parallel read time can be one time unit more than optimal. In Section C.2, we suggest a layout for general n and k. Here the layouts differ from ideal in two ways: the parallel read time can be one time unit more than optimal, and the reconstruction workload is not optimal if n is not prime.

The following definition and result are used to give an upper bound on the parallel read time of these layouts. Define a *n-block* of client data addresses to be any interval $\{t, t+1, \dots, t+n-1\}$ of n consecutive addresses such that n divides the starting point t. Say that a layout L has *maximal parallelism on n-blocks* if the addresses in any n-block are mapped to n different disks. (This is a weaker property than having maximal parallelism.)

Lemma 6 *If a layout L has maximal parallelism on n-blocks, then its parallel read time satisfies $\tau_L(o) \le \lceil o/n \rceil + 1$ for all o.*

Proof. Consider an arbitrary interval $I = \{\, t, t+1, \dots, t+o-1 \,\}$ of o consecutive client data addresses, for an arbitrary o and t. Let $\tau_L(I)$ be the maximum, over all disks d, of the number of $a \in I$ with $\mathit{disk}(a) = d$. If I is contained in a single n-block, then $\tau_L(I) = 1$ and $\lceil o/n \rceil = 1$. If I is not contained in a single n-block, then let n_1, n_2 with $0 \le n_1, n_2 < n$ and j with $j \ge 0$ be such that I consists of n_1 addresses of some n-block, followed by all the addresses in the next j n-blocks, followed by n_2 addresses of the next n-block. So $o = n_1 + jn + n_2$. If $n_1 = n_2 = 0$, then $\tau_L(I) = j$ and $\lceil o/n \rceil = j$. If at least one of n_1, n_2 is greater than zero, then $\tau_L(I) \le j+2$ since I spans at most $j+2$ n-blocks, and $\lceil o/n \rceil \ge \lceil (jn+1)/n \rceil = j+1$. In every case, $\tau_L(I) \le \lceil o/n \rceil + 1$. $\square$

C.1 Almost Ideal Layouts for Prime n

The layout, which we call PRIME, is obtained from one of the *ring-based block designs* of Schwabe and Sutherland [16]. In the basic layout (which is repeated to obtain larger layouts) each stripe has a name of the form (x, y) where $0 \le x, y \le n-1$ and $y \ne 0$. Thus, there are $b = n(n-1)$ stripes. The stripe with name (x, y) has its units mapped to disks in the set

$$\{\, (x+iy) \bmod n \;\mid\; 0 \le i \le k-1 \,\}.$$

It follows from Theorem 1 of [16] that this mapping of stripes to disks gives a layout that is single failure correcting and has optimally distributed reconstruction. For completeness, we repeat the arguments here, since they are simple in this special case, and since some of the arguments apply to the layout in the next section.

First, since n is prime, n is relatively prime to every y with $1 \le y \le n-1$; it then follows from Fact 1 that this mapping never maps two units of the same stripe to the same disk, because the residues $iy \bmod n$ are distinct for $0 \le i \le k-1$.

To see that the stripe units can be mapped one-to-one onto the disk locations, it is enough to note that, for each disk name d_0, there are exactly $r = bk/n = k(n-1)$ stripe units mapped to disk d_0. This follows since, for each of the k values of i with $0 \le i \le k-1$ and each of the $n-1$ values of y with $1 \le y \le n-1$, there is exactly one value of x with $d_0 = (x+iy) \bmod n$.

To show that the layout has optimally distributed reconstruction we must show that every pair of disks share $\lambda_{opt} = r(k-1)/(n-1) = k(k-1)$ stripes. Fix any two disks d and d'. The disks share the stripe with name (x, y) iff there are i, i' with $0 \le i, i' \le k-1$ and $i \ne i'$ such that

$$d = (x+iy) \bmod n \quad \text{and} \quad d' = (x+i'y) \bmod n.$$

Subtracting these two equations gives $(d-d') = (i-i')y \bmod n$. Since n is prime, the integers modulo n are a field. Since $i - i' \ne 0$, $(i-i')$ has a multiplicative inverse in this field, so for each pair (i, i') there is exactly one y with $(d-d') = (i-i')y \bmod n$, namely, $y = (d-d')(i-i')^{-1} \bmod n$. Moreover, fixing i and y fixes x, namely, $x = (d-iy) \bmod n$. Therefore, for each of the $k(k-1)$ pairs (i, i'), there is exactly one stripe (x, y) such that disks d and d' share stripe (x, y). Moreover, different pairs (i, i') yield different stripes (x, y).

We take the unit mapped to disk $(x+(k-1)y) \bmod n$ to be the parity unit of stripe (x, y). Parity is optimally distributed since, fixing any value for y and letting x range from 0 to $n-1$, the value $(x+(k-1)y) \bmod n$ hits every disk exactly once. In the case of multiple-fault-tolerance with $f > 1$, the check units of stripe (x, y) are the

D0.0	D0.1	D0.2	D1.0	D1.1
D1.2	D2.0	D2.1	D2.2	D3.0
D3.1	D3.2	D4.0	D4.1	D4.2
P4	P1	P3	P0	P2
D5.0	D6.0	D5.1	D6.1	D5.2
D6.2	D7.2	D7.0	D8.0	D7.1
D8.1	D9.1	D8.2	D9.2	D9.0
P9	P5	P6	P7	P8

Figure 15: Illustrating the mapping of stripes to array locations for the PRIME layout.

units mapped to disks $(x + iy) \bmod n$ for $k - f \leq i \leq k - 1$. The same argument shows that check information is optimally distributed in this case because, fixing y and i, the value $(x + iy) \bmod n$ hits every disk exactly once as x ranges from 0 to $n - 1$.

To complete the description of the layout, we must describe how client data units are mapped to stripe units and how stripe units are mapped to disk offsets. Since we already have a convention (15) for mapping client data to stripes, we are, in effect, describing how the stripe unit names of the form $((x, y), i)$ with $0 \leq x \leq n - 1$, $1 \leq y \leq n - 1$, and $0 \leq i \leq k - 1$, correspond to stripe unit names of the form (s, u) with $0 \leq s \leq n(n - 1) - 1$ and $0 \leq u \leq k - 1$; although, it is more natural to think in terms of mapping client data units to stripes, using stripe names of the form (x, y). For the record, however, the correspondence is given by:

$$y = \lfloor s/n \rfloor + 1, \quad x = sy(k - 1) \bmod n, \quad i = u.$$

We now explain what this means in terms of mapping client data units to stripe units, and thus to disks. Recall that the client data addresses are divided into *stripe intervals* of length $k - 1$ each. The first n stripe intervals, and therefore the first $(k - 1)n$ units of client data, are mapped to stripes with names (x, y) where $y = 1$. (Since the following description is useful for other y's, we describe it in terms of a "general" y instead of the specific $y = 1$.) The first stripe interval is mapped to stripe $(0, y)$, so data address a is mapped to disk $ay \bmod n$ for $0 \leq a \leq k - 2$. The next stripe interval is mapped to stripe $((k - 1)y \bmod n, y)$, so the mapping of address a to disk $ay \bmod n$ holds for this stripe interval ($k - 1 \leq a \leq 2(k - 1) - 1$) as well. Following stripe intervals are mapped to stripes

$$(2(k - 1)y \bmod n, y), (3(k - 1)y \bmod n, y), (4(k - 1)y \bmod n, y), \ldots$$

until n stripe intervals have been mapped. It is easy to see that the mapping of data address a to disk $ay \bmod n$ holds for all $0 \leq a < (k - 1)n$. At this point, $(k - 1)n$ stripe data units have been mapped into. Assign these $(k - 1)n$ data units to the first $k - 1$ offsets (filling up lower numbered offsets first), and assign the parity units of the stripes (x, y) ($0 \leq x \leq n - 1$) to the next offset. This pattern is repeated with increasing values of y; i.e., the next $(k - 1)n$ client data units are mapped as above with $y = 2$, the next $(k - 1)n$ are mapped with $y = 3$, and so on. The mapping is illustrated in Figure 15 for $n = 5$, $k = 4$, and $y = 1, 2$, so stripes 0 through 9 are being mapped to offsets 0 through 7.

The mapping functions from client addresses to array locations have been listed in the body of the paper. Since the mapping of client addresses to disks within any n-block is a mapping of the form $ay \bmod n$ for some y, it follows from Fact 1 that the layout has maximal parallelism on n-blocks. Using Lemma 6, this shows the following.

Theorem 7 *For every prime n, every k with $2 \leq k \leq n$, and every r such that $(k(n - 1))|r$, the layout* PRIME *is single failure correcting, has optimally distributed parity, has optimally distributed reconstruction, has large write optimization, and has parallel read time $\tau(o) \leq \lceil o/n \rceil + 1$ for all o.*

C.2 Less Ideal Layouts for General n and k

The layout PRIME does not work for general (non-prime) n since it might not be single failure correcting. A solution, which we investigate in this section, is to restrict the y's to be relatively prime to n. Following the definitions in

Section 4.2, we consider the layout with stripes defined as in the previous section, but restricted to those with names (x, y) where $0 \le x \le n-1$ and $y \in Y$. We call this layout RELPR (for *Relatively Prime*). The number of stripes is $b = n \cdot \phi(n)$. The number of units per disk is $r = bk/n = k \cdot \phi(n)$. Again, for each stripe (x, y), we let the unit mapped to disk $(x + (k-1)y) \bmod n$ be the parity unit. The layout RELPR is single failure correcting, has optimally distributed parity, and each disk contains r stripe units; the arguments are identical to those given in the previous section for PRIME.

To map client data to stripes, we again divide the space of client data addresses into blocks of length $(k-1)n$, where each block contains n stripe intervals, and all such stripe intervals are mapped to stripes (x, y) with a fixed y and $0 \le x \le n-1$. The mapping is a little more complicated than before if $\gcd(k-1, n) > 1$.

Fix some $y \in Y$, and let $g = \gcd(k-1, n)$. The n stripe intervals that are mapped to stripes (x, y) for this fixed y are mapped to stripes with the x's chosen in the following order, where all numbers in this list are reduced modulo n:

$$\begin{array}{ccccc} 0, & (k-1)y, & 2(k-1)y, & \cdots & (\frac{n}{g}-1)(k-1)y, \\ y, & y+(k-1)y, & y+2(k-1)y, & \cdots & y+(\frac{n}{g}-1)(k-1)y, \\ 2y, & 2y+(k-1)y, & 2y+2(k-1)y, & \cdots & 2y+(\frac{n}{g}-1)(k-1)y, \\ \vdots & \vdots & \vdots & & \vdots \\ (g-1)y, & (g-1)y+(k-1)y, & (g-1)y+2(k-1)y, & \cdots & (g-1)y+(\frac{n}{g}-1)(k-1)y \end{array}$$

There are a total of n numbers in this list, that is, g "rows" with n/g numbers in each row. We must show that if x and x' are different elements of this list, then $x \not\equiv x' \bmod n$ (we need this so that every stripe (x, y) for $0 \le x \le n-1$ is used). Suppose for contradiction that $x \equiv x' \bmod n$. There are two cases. First, suppose that x and x' belong to the same row of the list. Then n divides $(j-j')(k-1)y$ for some j, j' with $0 \le j' < j < n/g$. Since n and y are relatively prime, this implies that n divides $(j-j')(k-1)$. Since $g = \gcd(k-1, n)$, it follows that n/g divides $(j-j')((k-1)/g)$ and that n/g and $(k-1)/g$ are relatively prime. So n/g divides $(j-j')$, which is a contradiction because $1 \le j-j' < n/g$. The second case is that x and x' belong to different rows of the list. Now g divides $x - x'$, since g divides n, and n divides $x - x'$ by assumption. For $0 \le i \le g-1$, every element in the ith row of the list is congruent to $iy \bmod g$, since g divides $k-1$. So we have that g divides $(i-i')y$ for some i, i' with $0 \le i' < i < g$. Since $g|n$ and $\gcd(y, n) = 1$, it follows that $\gcd(y, g) = 1$, so g divides $(i-i')$, which is a contradiction because $1 \le i - i' < g$.

For $0 \le i \le g-1$, the number of elements in the ith row of the list is n/g. So the number of client data units mapped using x's from the ith row is $(n/g)(k-1)$. Since $g|(k-1)$, this number of data units is a multiple of n. Moreover, the same function, $(iy + ay) \bmod n$, is used for mapping all of these data units to disks. Since y and n are relatively prime, it follows from Fact 1 that this mapping of client data to disks has maximal parallelism on n-blocks. So the parallel read time of the layout RELPR is, by Lemma 6, at most one time unit more than optimal. By placing the parity units at every kth offset as was done for the layout PRIME, the mapping of client data to array locations for the layout RELPR is calculated using the formulas presented in Section 4.2.

It remains to consider the reconstruction workload of RELPR. The best general upper bound on α_{Relpr} that we have been able to prove is:

$$\alpha_{Relpr}(n, k) \le \frac{(k-1)}{\phi(n)}. \tag{17}$$

Therefore, in the worst case,

$$\frac{\alpha_{Relpr}(n, k)}{\alpha_{opt}(n, k)} \le \frac{(n-1)}{\phi(n)}.$$

Even though the bound $(n-1)/\phi(n)$ may not be good in many cases, the calculations of actual reconstruction workloads does suggest that the "bad" n's, i.e., the n's that have large reconstruction workloads, are those for which $\phi(n)$ is small compared to n. This is also consistent with the fact that, if n is prime, then RELPR is the same as PRIME, $\phi(n) = n-1$ (as large as possible compared to n), and $\alpha_{Relpr}(n, k) = \alpha_{opt}(n, k)$ (as good as possible). Since $r = k \cdot \phi(n)$ for the basic RELPR layout, the bound (17) is immediate from the following lemma.

Lemma 8 *For the* RELPR *layout with parameters n, k, r where $r = k \cdot \phi(n)$, any two disks share at most $k(k-1)$ stripes.*

Proof. Fix any two disks d and d' with $d \ne d'$. If disks d and d' share stripe (x, y) then $d = (x + iy) \bmod n$ and $d' = (x + i'y) \bmod n$ for some i, i' with $0 \le i, i' \le k-1$ and $i \ne i'$. Since i and y determine x, we can bound the number of shared stripes by bounding the number of solutions (i, i', y) to this pair of congruences. Letting $\Delta = d - d'$ and $\delta = i - i'$,

$$\Delta = \delta y \bmod n. \tag{18}$$

We bound the number of solutions (i, i', y) for $\delta \geq 1$. By symmetry, we get an upper bound on the number of solutions for all δ by doubling the upper bound for the case $\delta \geq 1$.

We first observe that, for a fixed δ, if (18) has any solution y then $\gcd(\Delta, n) = \gcd(\delta, n)$. This is true because, if there is a solution y, then $\Delta - \delta y = zn$ for some integer z. Therefore, if w is any integer that divides both n and δ, then w also divides Δ. So $\gcd(\delta, n)$ divides $\gcd(\Delta, n)$. Also, if w divides both n and Δ, then w divides δy. Note that $\gcd(w, y) = 1$: for if $u \geq 2$ and u divides both w and y, then u divides both n and y, which contradicts the fact that $\gcd(y, n) = 1$. Therefore, w divides δ. So $\gcd(\Delta, n)$ divides $\gcd(\delta, n)$. This establishes the observation that $\gcd(\Delta, n) = \gcd(\delta, n)$. In particular, if (18) has any solution y, then $\gcd(\Delta, n)$ must divide δ. It is known that (18) has at most $\gcd(\delta, n)$ solutions y. Summarizing, for a fixed δ, if (18) has any solution y, then $\gcd(\Delta, n)$ divides δ, and there are at most $\gcd(\Delta, n)$ solutions y.

If (18) has t solutions y, again for a fixed value of δ, then this produces $t(k - \delta)$ solutions (i, i', y), since $0 \leq i' < i \leq k - 1$ and $i - i' = \delta$. From this, it is not hard to see that the total number of solutions (i, i', y) is maximized if $\gcd(\Delta, n) = 1$ and (18) has one solution y for each value of δ. For example, if $\gcd(\Delta, n) = 3$, then there are no solutions y for $\delta = 1, 2$, and at most 3 solutions y for $\delta = 3$, thus producing at most $3(k - 3)$ solutions (i, i', y) for $\delta = 1, 2, 3$. On the other hand, if there is one solution y for each of $\delta = 1, 2, 3$, this produces a larger number $(k - 1) + (k - 2) + (k - 3)$ of solutions (i, i', y).

If there is one solution y for each value of $\delta \geq 1$, the total number of solutions (i, i', y) produced is $1 + 2 + \cdots + (k - 1) = k(k - 1)/2$. Doubling this bound to include $\delta \leq -1$ gives the bound claimed in the lemma. $\Box$

Properties of the layout RELPR are summarized in the following.

Theorem 9 *For every n, every k with $2 \leq k \leq n$, and every r such that $(k \cdot \phi(n)) | r$, the layout* RELPR *is single failure correcting, has optimally distributed parity, has large write optimization, has reconstruction workload bounded above by $(k - 1)/\phi(n)$, and has parallel read time $\tau(o) \leq \lceil o/n \rceil + 1$.*