

STORAGE SYSTEM ARCHITECTURES

Walt Burkhard

Computer Science and Engineering 226

Spring 2010

Soft Reserves Student Center A Room 122

Table of Contents –

1. Disk Drive Modeling	1
2. Response Time Modeling	6
3. RAID	8
4. Information Dispersal	13
5. Reliability Analysis	17
6. Parity Declustering	21
7. Datum Declustering Data Layout	25
8. Prime Declustering Data Layout	31
9. Spares and Permutation Development Data Layout	34
10. EVENODD Data Layout	39

STORAGE SYSTEM ARCHITECTURES

Disk Drive Modeling

Walt Burkhard

Spring 2010

Computer Science and Engineering Department

UCSD

All rights reserved. No part of these notes may be reproduced, stored, or transmitted in any form or means without the permission of the author.

1



Start with your data: one sector.

Add ability to write in slightly different places.



Decide if what you read is actually good or not.



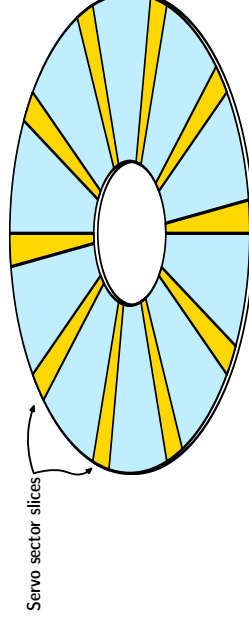
Fix what wasn't read correctly



Decide which sector to read or write

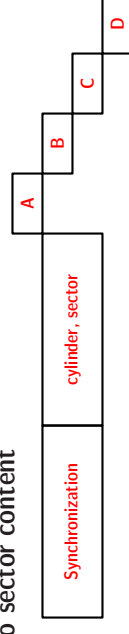


- Your data bytes ... **AF B9 C3** ...
- Randomized to prevent resonating patterns; exclusive-or with ... 01 02 03 Your data bytes now ... **AE BB C0** ...
- Encoded for run-length pattern characteristics; extra bits are added to breakup zero strings; eight bit bytes represented as nine bit values.
... **101111001 001110001 011110111** ...

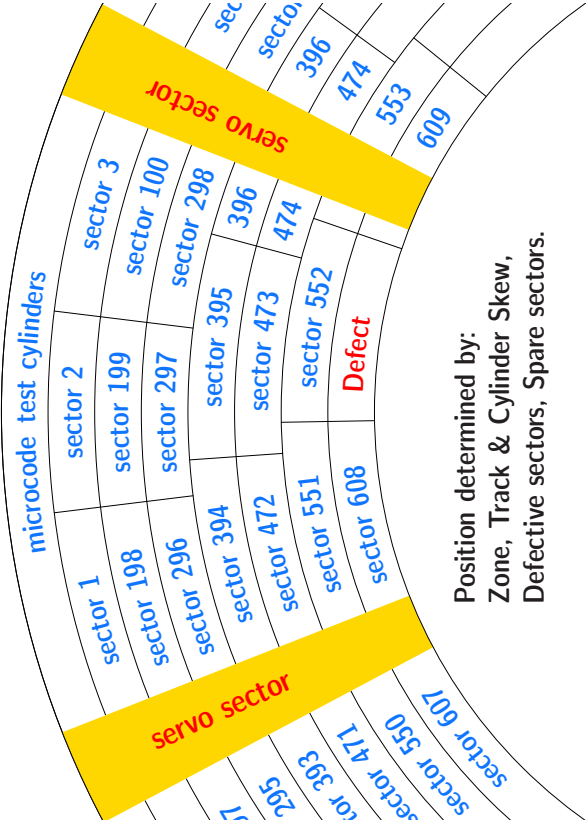


50 - 150 servo sector wedges per platter

- Servo sector content



cylinder, sector written in Gray code readable during seek. Position determined from relative amplitudes of **A,B,C,D** signals.



■ Recording Components

Sector: smallest accessible data object; typically 512 bytes.

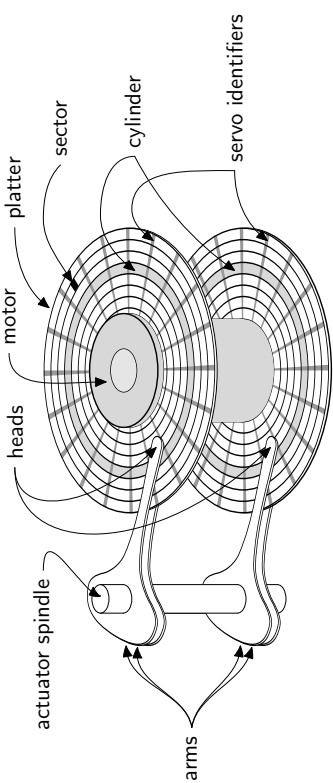
Platter: surface organized to store data in a series of concentric rings referred to as **tracks**. Typically 10,000-20,000 tracks per radial inch; each track contains 200-400 sectors. Bits are spaced at 100-200 nanometers along a track.

Cylinder: stack of tracks with common radius.

Speed: Higher speed implies higher transfer rates; reduces rotational latency. Typically 3,600-15,000 revolutions per minute (rpm).

Head: each surface has a head for writing and/or reading data.

Actuator: rotational lever moves head(s) to the desired cylinder.



Disk Drive Components

■ Data Layout

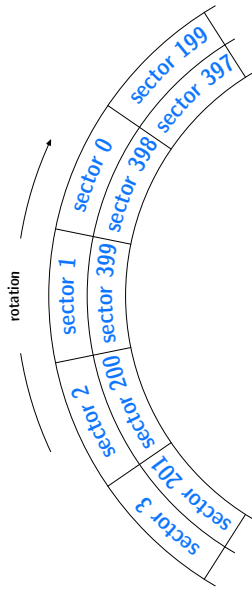
Logical: User data addresses are logical sector addresses within the range 0 through maximum sector minus one.

Physical: Physical sector addresses: cylinder, head, servo-sector location on a disk.

The mapping between logical and physical addresses is accomplished via the disk drive cpu.

skew:

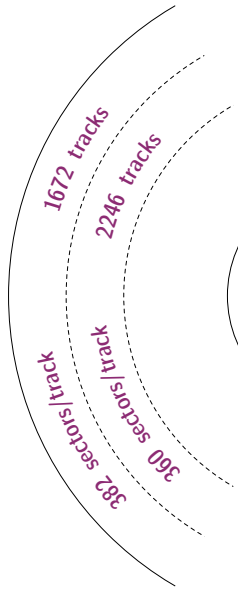
Data are organized to facilitate sequential access. Sequential data on adjacent tracks or cylinders are offset physically to allow the drive time to switch/move the head to the new location so the desired data will then immediately be accessed.



Accessing sectors 199 and then 200 requires the disk head to shift from the outer track to the next track; the is done very quickly, especially for read operations, and sector 200 will move under the head almost immediately in the shift time interval.

zones:

The “longer” tracks contain more sectors; the platter is divided into a number of zones, typically 15-25, allowing more data stored on the “outside” tracks. This variability gives rise to the range of data transfer rates.



The outermost two zones contain 382 and 360 sectors per track; there are 1672 and 2246 tracks in these two zones.

■ Data Access Process

Data objects (sectors) are accessed via a three-phased process consisting of moving the actuator arm/head to the desired track from its current location followed by waiting for the desired sector to rotate under the head and finally transferring the data to (or from) the drive platter.

- seek time:* move head to desired track
- rotational latency:* desired sector rotates under the head
- transfer time:* data transfer time to/from the disk.
- access time:* the sum of these three times.

■ varieties of seek operations

- Very short seek* no more than 4 tracks; essentially settle time: 0.5 ms.
- Short seek time* no more than 2000 tracks. constant acceleration or deceleration; $time \propto \sqrt{distance}$ + settle time.
- Long seek time* proportional to distance plus settle time.

Seek Operation

at most four phases

- speedup* arm accelerated until it reaches half the seek distance or the maximum velocity.
- coast* for long seeks, arms move at constant velocity for a portion of the excursion.
- slowdown* arms brought to rest close to desired track.
- settle* controller adjusts head to the desired track.

typical performance characteristics IBM18LZX

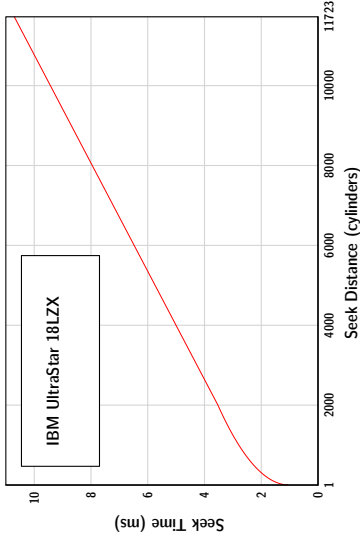
- seek latency:* 4.9 ms
- rotational latency:* 3 ms 10000 rpm

the rotation latency represents 1/2 revolution; the seek latency represents 1/3 of a full excursion.

- transfer rate:* 25 MB/s
- sector size:* 512 bytes

- access times** seek plus rotational latencies plus transfer time
 - time to access 100 bytes: 1.1 through 16.7 ms
 - time to access 1000 bytes: 1.1 through 16.9 ms
 - time to access 5000 bytes: 1.1 through 17.0 ms

Seek Times



$$t_{\text{seek}} = \begin{cases} a + b\sqrt{\text{distance}} & \text{distance} \leq 2000 \\ c + d \text{ distance} & \text{distance} \geq 2000 \end{cases}$$

$$a = 1.010404 \quad b = 0.056256 \quad c = 2.050601 \quad d = 0.000738$$

typical performance characteristics: 2³⁰ records

- AVL trees:** binary nodes
 - minimum height = $\lceil \log_2(2^{30} + 1) \rceil = 31$
 - maximum height = $\lceil 1.44 \log_2(2^{30} + 2) \rceil = 43$
- B trees:** 50-children nodes
 - minimum height = $\lceil \log_{50}(2^{30} + 1) \rceil = 6$
 - maximum height = $\lceil \log_{25}(2^{29} + \frac{1}{2}) \rceil = 7$

- AVL tree nodes:** 100 bytes 7.9 ms per node
- 245 through 340 ms per record access
- B tree nodes:** 5000 bytes 8.2 ms per node
- 49 through 57 ms per record access

- Specifications Quantum Fireball Series circa 1996

formatted capacity (MB)	1082	2168	3255
average seek time (ms)	12	10.5	10.5
data transfer rate (MB/s)	20	20	20
rotational speed (rpm)	4500	4500	4500
field life mtbf (hours)	400000	400000	400000

(400000 hours is more than 45 years.)

rotational latency = $\frac{1}{2} \left(\frac{1}{4500} \right) 60\ 1000 = 6.66\text{ ms}$

access time = 12 + 6.66 + data transfer time ms

- Specifications Seagate Cheetah 10k.6 series circa 2004

formatted capacity (GB)	36	73	147
average seek time r/w (ms)	4.7/5.2	4.7/5.2	4.7/5.2
data transfer rate (MB/s)		15.2 to 29.5	
rotational speed (rpm)	10000	10000	10000
field life mtbf (hours)	1200000	1200000	1200000

rotational latency = $\frac{1}{2} \left(\frac{1}{10000} \right) 60\ 1000 = 3.0\text{ ms}$

access time = 5.2 + 3.0 + data transfer time ms

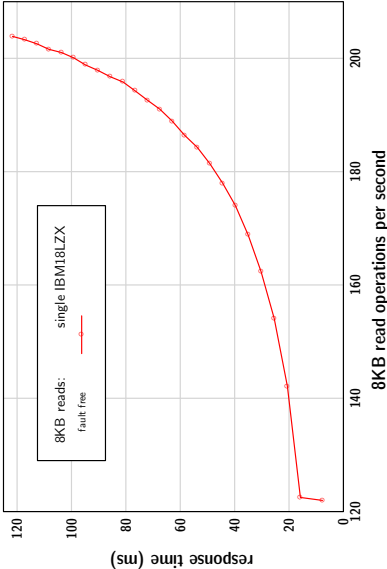
- Specifications IBM UltraStar LZX Series circa 1999

formatted capacity (GB)	9.1	18.3	36.7
average seek time (ms)	4.9	4.9	5.4
data transfer rate (MB/s)		15.2 to 29.5	
rotational speed (rpm)	10000	10000	10000

rotational latency = $\frac{1}{2} \left(\frac{1}{10000} \right) 60\ 1000 = 3.0\text{ ms}$

access time = 5.4 + 3.0 + data transfer time ms

simulation results via Raidframe.



STORAGE SYSTEM ARCHITECTURES

Response Time Modeling

Walt Burkhard

Spring 2010

Computer Science and Engineering Department

UCSD

All rights reserved. No part of these notes may be reproduced, stored, or transmitted in any form or means without the permission of the author.

- **service time T_S** time for the disk/array to perform the operation. Determined by the transfer time, seek latency, and rotational latency.
- **waiting time T_Q** time spent in the disk/array awaiting service.
- **response time T_R** $= T_Q + T_S$.

■ **N** is the number of operations within the disk/array system: the queue and the server.

■ **λ** is the operation arrival rate. This is the workload for the disk/array.

■ **Little's Law** $N = T_R \lambda$

- **basic queueing theory.**

m/m/1 model; first come; first served.

λ is the operation arrival rate

μ is the service rate $= \frac{1}{T_S}$.

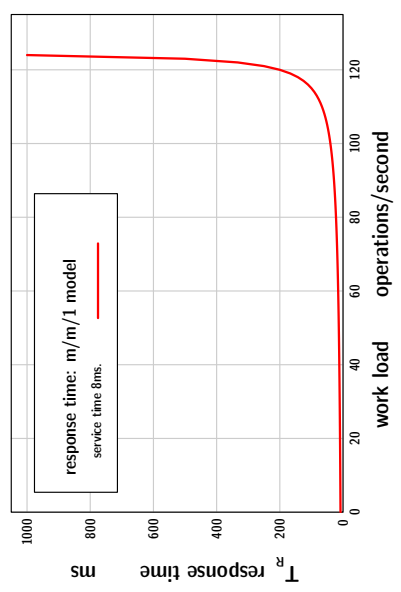
$$\rho = \lambda / \mu$$

- **result** $N = \frac{\rho}{1-\rho}$.

Using Little's law, we have

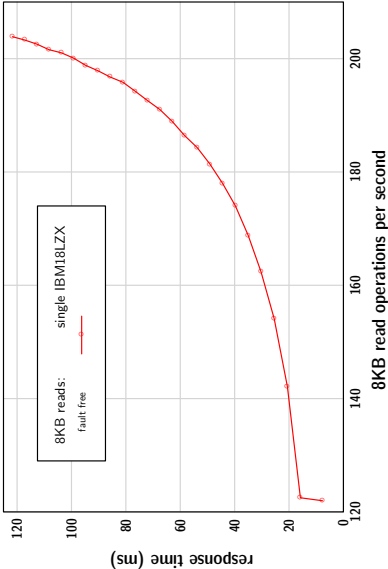
$$T_R = N / \lambda = \frac{1/\mu}{1-\rho} = \frac{1}{\mu-\lambda}.$$

- **Response Time T_R**
8ms service time – queueing theory model.



■ Response Time T_R

single disk with 8ms service time.
simulation results via Raidframe



STORAGE SYSTEM ARCHITECTURES

RAID

Walt Burkhard Spring 2010

Computer Science and Engineering Department UCSD
All rights reserved. No part of these notes may be reproduced, stored, or transmitted in any form or means without the permission of the author.

Redundant Array of Independent Disks RAID

■ SIGMOD 1988: D. Patterson, G. Gibson, R. Katz

■ Disk Data Model

a sequence of r same sized **stripe units** (“scsi” interface) with indices $0,1,\dots,r-1$. Stripe units may be (groups of) sectors, tracks, cylinders, or ...

■ User Data Model

a sequence of same sized data stripe units with indices $0,1,2,\dots$. Data stripe units typically created by file system, database management system, or other (sophisticated) user program.

■ Reliability Groups a.k.a. Stripes

user data is partitioned into fixed sized groups with each group containing additional redundancy information.

■ Disk Array Data Layouts

Data Layouts: map stripe unit indices to disk sectors, tracks, or cylinders ...

n disks r stripe units/disk

k stripe width b number of stripes

g groups $n = k \cdot g$

m user data stripe units/stripe

$c = k - m$ redundant stripe units/stripe

Total number of data stripe units $b(k - c)$.

Total number of parity stripe units $b \cdot c$.

The total number of stripe units within for a completely filled/used disk array is $bk = nr$.

RAID II

$d_0, d_1, d_2, d_3, \dots$ user data
 $\{d_0, d_1, d_2, c_0\}$ stripes
 $\{d_3, d_4, d_5, c_1\}$
 $\{d_6, d_7, d_8, c_2\}$
 $\vdots$ c_i redundant data

data stripe unit indices (user address space)

$a \in DSUI = \{0, 1, 2, \dots, b(k-c)-1\}$

disk: $DSUI \mapsto \{0, 1, 2, \dots, n-1\}$

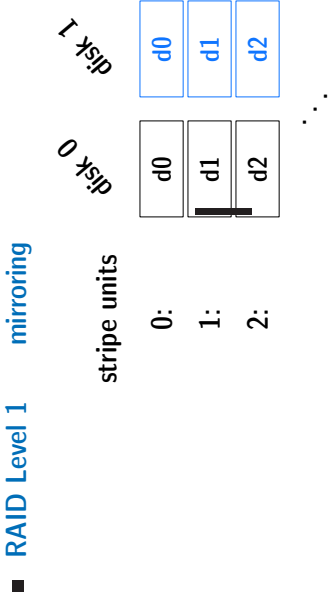
offset: $DSUI \mapsto \{0, 1, 2, \dots, r-1\}$

checkDisk: $DSUI \times \{0, 1, \dots, c-1\} \mapsto \{0, 1, 2, \dots, n-1\}$

checkOffset: $DSUI \times \{0, 1, \dots, c-1\} \mapsto \{0, 1, 2, \dots, r-1\}$

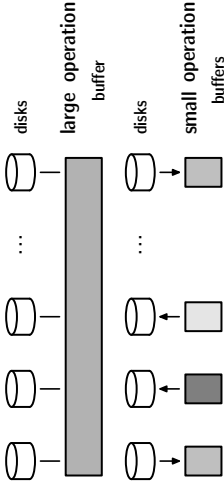
The mapping (disk, offset, checkDisk, checkDisk) is a bijection.

- Layout Taxonomy typically
 - Level 0 no redundancy; just a bunch of disks, JBOD. $m = k = 1$
 - Level 1 mirroring $m = 1, k = 2$
 - Level 2 fine-grained interleaving with ECC error correction $m = 10, k = 14$
 $m = 20, k = 25$
 - Level 3 fine-grained interleaving with dedicated parity disk $k = m + 1$
 - Level 4 stripe unit interleaving; dedicated parity disk $k = m + 1$
 - Level 5 stripe-unit interleaving; distributed parity $k = m + 1$
 - Level 6 Level 5 with additional redundant stripe-units; typically one more $k = m + 2$

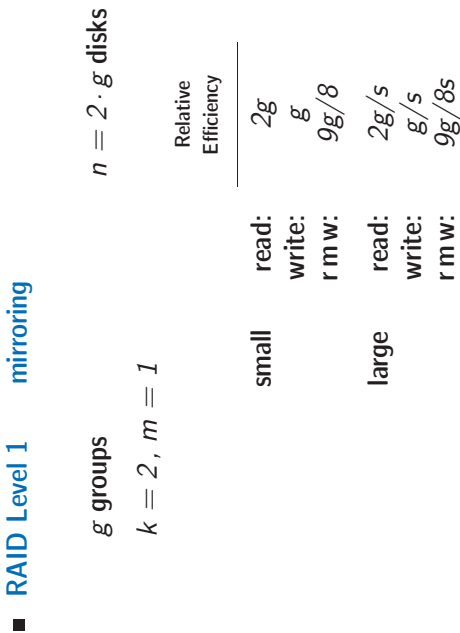


$disk(a) = 0$
 $offset(a) = a$
 $checkDisk(a) = 1$
 $checkOffset(a) = a$

- Workloads
 - Large Operations Parallel read or write accesses one stripe unit from each disk; high data transfer rates obtained.
 - Small Operations Independent read or write accesses one data stripe unit from each disk; high numbers of i/o operations obtained.



- Operations small reads, writes, read-modify-writes (rmw) & large reads, writes, read-modify-writes.
- Relative Efficiency RAID operations / sec.



s slowdown: Within large operations, all disks complete their individual tasks to finish the operation; s is between 1 and 2.

- RAID Level 2 fine-grain interleaving with ECC

disk 0	disk 1	disk 2	disk 3	disk 4	disk 5	disk 6
d0,3:0	d0,3:1	d0,3:2	d0,3:3	c0,3:4	c0,3:5	c0,3:6
d4,7:0	d4,7:1	d4,7:2	d4,7:3	c4,7:4	c4,7:5	c4,7:6
d8,11:0	d8,11:1	d8,11:2	d8,11:3	c8,11:4	c8,11:5	c8,11:6

$disk(a) \in \{ 0, 1, \dots, m-1 \}$
 $offset(a) = a/m$
 $checkDisk(a) \in \{ m, m+1, \dots, k-1 \}$
 $checkOffset(a) = a/m$

- RAID Level 2 fine-grain interleaving with ECC

g groups $n = k \cdot g$ disks
 $k = m + c$ stripe width

This level attempts to provide good performance with less redundant data.

		Relative Efficiency
small	read:	g/s
	write:	$g/2s$
	rmw:	g/s
large	read:	$g \cdot m/s$
	write:	$g \cdot m/s$
	rmw:	$g \cdot m/s$

Level 2 redundancy not needed; disk internal ecc guarantees one bit error in 10^{14} .

- RAID Level 3 fine-grained interleaving, dedicated parity disk

	disk 0	disk 1	disk 2	disk 3	disk 4
0:	d0,3:0	d0,3:1	d0,3:2	d0,3:3	c0,3
1:	d4,7:0	d4,7:1	d4,7:2	d4,7:3	c4,7
2:	d8,11:0	d8,11:1	d8,11:2	d8,11:3	c8,11

$disk(a) \in \{ 0, 1, \dots, m-1 \}$ $offset(a) = a/m$
 $checkDisk(a) = k-1 = m$ $checkOffset(a) = a/m$

- RAID Level 3 fine-grained interleaving; dedicated parity disk

g groups $n = k \cdot g$ disks
 $k = m + 1$ stripe width

This level provides good performance with less redundant data. Level 2 and Level 3 performances are identical with one redundant disk.

		Relative Efficiency
small	read:	g/s
	write:	$g/2s$
	rmw:	g/s
large	read:	$g \cdot m/s$
	write:	$g \cdot m/s$
	rmw:	$g \cdot m/s$

Un-interleave data to improve small operations.

- RAID Level 4 stripe-unit interleaving; dedicated parity disk

	disk 0	disk 1	disk 2	disk 3	disk 4
0:	d0	d1	d2	d3	c0,3
1:	d4	d5	d6	d7	c4,7
2:	d8	d9	d10	d11	c8,11

$disk(a) = a \% m$ $offset(a) = a / m$
 $checkDisk(a) = k - 1 = m$ $checkOffset(a) = a / m$

$c8,11 = d8 \oplus d9 \oplus d10 \oplus d11$ $d9 = c8,11 \oplus d8 \oplus d10 \oplus d11$

- RAID Level 4 stripe-unit interleaving; dedicated parity disk

g groups $n = k \cdot g$ disks
 $k = m + 1$ stripe width

Level 4 provides better small operation performance than Level 3.

	small	read:	write:	rmw:	Relative Efficiency
		$g \cdot m$	$g / 2$	g	
large		$g \cdot m / s$	$g \cdot m / s$	$g \cdot m / s$	

- RAID Level 5 stripe-unit interleaving, distributed parity

Level 4 with distributed parity stripe units.

	disk 0	disk 1	disk 2	disk 3	disk 4
0:	d0	d1	d2	d3	p0,3
1:	d4	d5	d6	p4,7	d7
2:	d8	d9	p8,11	d10	d11
3:	d12	p12,15	d13	d14	d15
4:	p16,19	d16	d17	d18	d19

$P = m - Q \% k$ $R = a \% m$ $Q = a / m$
 $disk(a) = \begin{cases} R & \text{if } R < P \\ R + 1 & \text{otherwise} \end{cases}$ $offset(a) = Q$
 $checkDisk(a) = P$ $checkOffset(a) = Q$

- RAID Level 5 stripe-unit interleaving, distributed parity.

g groups $n = k \cdot g$ disks
 $k = m + 1$ stripe width

Level 5 provides better than Level 4 small operation performance.

	small	read:	write:	rmw:	Relative Efficiency
		$g \cdot k$	$g \cdot k / 4$	$g \cdot k / 2$	
large		$g \cdot m / s$	$g \cdot m / s$	$g \cdot m / s$	

- RAID Level 5 Left Symmetric layout

	disk 0	disk 1	disk 2	disk 3	disk 4
0:	d0	d1	d2	d3	p0,3
1:	d5	d6	d7	p4,7	d4
2:	d10	d11	p8,11	d8	d9
3:	d15	p12,15	d12	d13	d14
4:	p16,19	d16	d17	d18	d19

...

$$P = m - Q \% k \quad Q = a / m$$

$$\begin{aligned} disk(a) &= a \% k & offset(a) &= Q \\ checkDisk(a) &= P & checkOffset(a) &= Q \end{aligned}$$

- RAID Level 6 Level 5 with additional check stripe units.

	disk 0	disk 1	disk 2	disk 3	disk 4	disk 5
0:	d0	d1	d2	d3	c0,3:0	c0,3:1
1:	d4	d5	d6	c4,7:0	c4,7:1	d7
2:	d8	d9	c8,11:0	c8,11:1	d10	d11
3:	d12	c12,15:0	c12,15:1	d13	d14	d15
4:	c16,19:0	c16,19:1	d16	d17	d18	d19

$$P = m - Q \% k \quad R = a \% m \quad Q = a / m$$

$$\begin{aligned} disk(a) &= \begin{cases} R & \text{if } R < P \\ R + c & \text{otherwise} \end{cases} & offset(a) &= Q \\ checkDisk(a) &\in \{ P, P+1, \dots, P+(c-1) \} & checkOffset(a) &= Q \end{aligned}$$

- RAID Level 6 stripe-unit interleaving, distributed check units

$$\begin{aligned} g \text{ groups} & \quad n = k \cdot g \text{ disks} \\ k = m + 2 & \quad \text{typical stripe width} \end{aligned}$$

Level 6 provides better reliability but with performance degradation.

	Relative Efficiency
small	read: $g \cdot k$
	write: $g \cdot k / 6$
	rmw: $g \cdot k / 3$
large	read: $g \cdot m / s$
	write: $g \cdot m / s$
	rmw: $g \cdot m / s$

- RAID Level 6 “Left Symmetric” layout

	disk 0	disk 1	disk 2	disk 3	disk 4	disk 5
0:	d0	d1	d2	d3	c0,3:0	c0,3:1
1:	d6	d7	c4,7:0	c4,7:1	d4	d5
2:	c8,11:0	c8,11:1	d8	d9	d10	d11

...

$$\text{assume } m = c \lambda \ ; \ P = m - (Q \% k) \lambda \quad Q = a / m$$

$$disk(a) = a \% k$$

$$offset(a) = Q$$

$$checkDisk(a) \in \{ P, P+1, \dots, P+(c-1) \} \quad checkOffset(a) = Q$$

STORAGE SYSTEM ARCHITECTURES

Information Dispersal

Walt Burkhard Spring 2010

Computer Science and Engineering Department UCSD

All rights reserved. No part of these notes may be reproduced, stored, or transmitted in any form or means without the permission of the author.

■ Information Dispersal Algorithm Rabin JACM 1989.

Information dispersal is a mechanism providing redundant values allowing protection against multiple disk failures.

m data disks and c check disks.

We wish to reconstruct the m client data values provided we have access to at least m disks available (surviving).

A reliability group is an ensemble of $k = c + m$ disks; containing m disks of client data and c disks of redundant data.

IDA utilizes a linear transformation of the m client data values

$$d_0, d_1, \dots, d_{m-1}$$

where each is a byte or sector value.

The linear transformation M creates $k = m + c$ values.

$$(d_0, d_1, \dots, d_{m-1})M = (e_0, e_1, \dots, e_{m-1}, e_m, \dots, e_{m+c-1})$$

M has m rows and $m + c$ columns.

- **Necessary Property** Any m columns of M are linearly independent.

The necessary property allows us to determine the m client data values within a reliability group provided we are given any m values from the corresponding reliability group.

Suppose we have access to the m values

$$e_{i_0}, e_{i_1}, \dots, e_{i_{m-1}}$$

m columns of $M = \{m_{ij}\}$ create these values

$$(d_0, d_1, \dots, d_{m-1})(m_{i_0}, m_{i_1}, \dots, m_{i_{m-1}}) = (e_{i_0}, e_{i_1}, \dots, e_{i_{m-1}})$$

Since these m columns are linearly independent, we can invert this matrix to solve for

$$(d_0, d_1, \dots, d_{m-1})$$

Issues to Resolve

- How to construct M?
- How to provide RAID-5 like data layout (systematic encoding)?
- Space required to store M as well as encoded values.

Creating M Use Vandermonde's matrix –

$$\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ \alpha_0 & \alpha_1 & \alpha_2 & \dots & \alpha_{k-1} \\ \alpha_0^2 & \alpha_1^2 & \alpha_2^2 & \dots & \alpha_{k-1}^2 \\ \vdots & \vdots & \vdots & \dots & \vdots \\ \alpha_0^{k-1} & \alpha_1^{k-1} & \alpha_2^{k-1} & \dots & \alpha_{k-1}^{k-1} \end{bmatrix}$$

The determinant of this $k \times k$ matrix is

$$\prod_{0 \leq i < j < k} (\alpha_i - \alpha_j)$$

which is non-zero provided $\alpha_i \neq \alpha_j$ for $i \neq j$.

■ Creating M an m row by $m+c$ column matrix.

Choose $m+c$ distinct α values to create M. Any m columns are linearly independent as required.

Example: $m=3$ and $c=2$ with $\alpha=0, 1, 2, 3, 4$ —

$$M = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 \\ 0 & 1 & 4 & 9 & 16 \end{bmatrix}$$

Systematic encoding. M is transformed into a matrix so the m client data values remain unchanged simplifying client data access.

$$(d_0, d_1, \dots, d_{m-1}) \hat{M} = (d_0, d_1, \dots, d_{m-1}, e_m, \dots, e_{m+c-1})$$

$\hat{M} = [I \mid C]$ satisfies the necessary property and consists of an $m \times m$ identity matrix I with an $m \times c$ check matrix C.

Beginning with an appropriate Vandermonde matrix M, let T be the $m \times m$ matrix consisting of the left most m columns of M. Then $\hat{M} = T^{-1}M$.

Example continued

$$T = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \\ 0 & 1 & 4 \end{bmatrix} \quad T^{-1} = \frac{1}{2} \begin{bmatrix} 2 & -3 & 1 \\ 0 & 4 & -2 \\ 0 & -1 & 1 \end{bmatrix}$$

And finally

$$\hat{M} = \begin{bmatrix} 1 & 0 & 0 & 1 & 3 \\ 0 & 1 & 0 & -3 & -8 \\ 0 & 0 & 1 & 3 & 6 \end{bmatrix}.$$

■ **Space Issues** Assume each of m client data values consists of s bytes. The information dispersal transformation is to produce $m + c$ values each expressible within s bytes. For example, client sector (512 bytes) values are encoded as 512 byte values; the redundant values would also be 512 bytes. At first, this seems to be at odds with the Vandermonde matrix transformation approach because of all its large component multipliers. However, finite fields solve this problem.

A field allows us to do “normal” operations such as addition, subtraction, multiplication, and division. The infinite set of reals is familiar. A finite set of elements with these four operations is designated a **finite field**; a finite field containing q elements is denoted $GF(q)$. We will be particularly interested in finite fields containing $q = 2^n$ elements; $q = 256$ is particularly important within many digital applications.

When q is a prime, both addition and multiplication can be implemented via modulo q arithmetic. When q is a power of a prime (e.g. 2^n), we must implement both addition and multiplication for $GF(q)$.

■ **$GF(8)$** For $q = 2^n$, with n -bit representations of the 2^n values, the exclusive-or operation implements **addition**.

For $q = 2^n$, multiplication is defined via a primitive polynomial of degree n within $GF(2)[x]$. Here are some such primitive polynomials – $x^3 + x + 1$, $x^4 + x + 1$, $x^8 + x^4 + x^3 + x^2 + 1$, etc.

The primitive polynomial $p(x) = x^3 + x + 1$ has root α ; so $\alpha^3 + \alpha + 1 \equiv 0$.

label	vector	exponent	polynomial	
0	000		0	
2	010	α	α	successive polynomials computed modulo $\alpha^3 + \alpha + 1$.
4	100	α^2	α^2	
3	011	α^3	$\alpha + 1$	
6	110	α^4	$\alpha^2 + \alpha$	
7	111	α^5	$\alpha^2 + \alpha + 1$	
5	101	α^6	$\alpha^2 + 1$	
1	001	α^7	1	

Multiplication carried out as
 $(a_0 + a_1\alpha + a_2\alpha^2)(b_0 + b_1\alpha + b_2\alpha^2)$ modulo $p(\alpha)$

The addition and multiplication tables show these results with the labels representing the $GF(8)$ values.

×	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	3	1	7	5
3	0	3	6	5	7	4	1	2
4	0	4	3	7	6	2	5	1
5	0	5	1	4	2	7	3	6
6	0	6	7	1	5	3	2	4
7	0	7	5	2	1	6	4	3

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	0	3	2	5	4	7	6
2	2	3	0	1	6	7	4	5
3	3	2	1	0	7	2	5	4
4	4	5	6	7	0	5	2	3
5	5	4	7	2	5	0	3	2
6	6	7	4	5	2	3	0	1
7	7	6	5	4	3	2	1	0

These tables represent an (initial) implementation of the $GF(8)$ operations; of course, the addition table is not required as the exclusive-or operation obtains these values. The multiplication table can be reduced to approximately one-half its q^2 size; this is still too big for very large finite fields, e.g. $GF(2^8)$. We can replace the $GF(q)$ multiplication table with a pair of tables each of size q .

■ **Logarithms** The non-zero elements of $GF(q)$ are represented as powers of α ; the first seven powers of α represent the non-zero elements of $GF(8)$. We use α as the logarithm base; for non-zero β in $GF(q)$

$$\left. \begin{aligned} \log_{\alpha}(\beta) &= k \\ \text{antilog}_{\alpha}(k) &= \beta \end{aligned} \right\} \text{ means } \alpha^k = \beta$$

Multiplication and Division is implemented via –

$$\begin{aligned} xy &= \text{antilog}((\log(x) + \log(y)) \bmod (q-1)) \\ x/y &= \text{antilog}((\log(x) - \log(y)) \bmod (q-1)) \end{aligned}$$

all arithmetic uses ordinary operations (infinite field of reals.)

x	$\log_{\alpha}(x)$	$\text{antilog}_{\alpha}(x)$
---	--------------------	------------------------------

0	*	1
1	7	2
2	1	4
3	3	3
4	2	6
5	6	7
6	4	5
7	5	*

GF(8) tables

Example continued

$$M = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 \\ 0 & 1 & 4 & 5 & 6 \end{bmatrix} \quad \text{Vandermonde matrix} \\ 5 \equiv 3^2 \text{ \& } 6 \equiv 4^2$$

$$T = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \\ 0 & 1 & 4 \end{bmatrix} \quad T^{-1} = \begin{bmatrix} 1 & 4 & 5 \\ 0 & 7 & 6 \\ 0 & 3 & 3 \end{bmatrix}$$

And finally

$$\hat{M} = \begin{bmatrix} 1 & 0 & 0 & 1 & 4 \\ 0 & 1 & 0 & 1 & 3 \\ 0 & 0 & 1 & 1 & 6 \end{bmatrix}.$$

Digital applications typically do arithmetic byte by byte via GF(256) operations; the primitive polynomial $x^8 + x^4 + x^3 + x^2 + 1$ defines multiplication and division.

■ **Updates:** in systematic form.

$$(d_0, d_1, \dots, d_{m-1})M = (e_0, e_1, \dots, e_{m-1}, e_m \dots e_{m+c-1})$$

M = [I | C] consists of an $m \times m$ identity matrix I with an $m \times c$ check matrix C =

$$\begin{bmatrix} c_{0,m} & c_{0,m+1} & c_{0,m+2} & \dots & c_{0,m+c-1} \\ c_{1,m} & c_{1,m+1} & c_{1,m+2} & \dots & c_{1,m+c-1} \\ c_{2,m} & c_{2,m+1} & c_{2,m+2} & \dots & c_{2,m+c-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{m-1,m} & c_{m-1,m+1} & c_{m-1,m+2} & \dots & c_{m-1,m+c-1} \end{bmatrix}$$

e_i is d_i for $0 \leq i < m$ & e_i is $\sum_{j=0}^{m-1} c_{j,i} d_j$ for $m \leq i < m+c$

■ update d_i with d_i^{NEW}

■ **read:** $d_i^{\text{OLD}}, e_j^{\text{OLD}}$ for $m \leq j < m+c$

■ **calculate:** $\Delta = d_i^{\text{OLD}} \oplus d_i^{\text{NEW}}$
 $e_j^{\text{NEW}} = e_j^{\text{OLD}} \oplus \Delta c_{j-m,j}$ for $m \leq j < m+c$

■ **write:** $d_i^{\text{NEW}}, e_j^{\text{NEW}}$ for $m \leq j < m+c$

STORAGE SYSTEM ARCHITECTURES

Reliability Analysis

Walt Burkhard

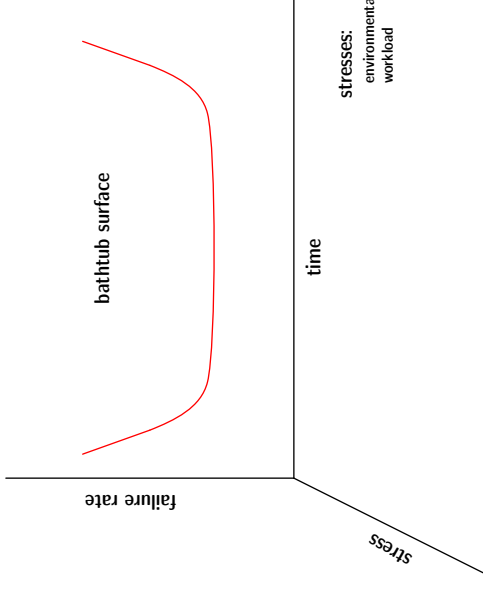
Spring 2010

Computer Science and Engineering Department

UCSD

All rights reserved. No part of these notes may be reproduced, stored, or transmitted in any form or means without the permission of the author.

- failure rate versus time



- First approximation: *constant* failure rate.

Reliability I

Markov Modeling Basics III

- $r(t)$ the *reliability* is the probability the system continues correct operation from the start to time t .

- **More Appropriate Measure**

mean time to failure MTTF. i.e. the expected failure time & mean time to data loss (MTTDL) in storage systems

calculate MTTDL from $r(t)$

$1 - r(t)$ probability of data loss at time no greater than t ; that is, the data loss distribution.

$-\frac{\partial r}{\partial t}$ the data loss density.

$$\text{MTTDL} = - \int_0^\infty t \frac{\partial r}{\partial t} dt = - t r(t) \Big|_0^\infty + \int_0^\infty r(t) dt = \int_0^\infty r(t) dt$$

provides a facility to calculate the reliability function and MTTDL.

states: “description of real world system.” States are mutually disjoint configurations that *cover* the configuration space. We can be in one state at a time. The configuration space must contain enough detail to represent the activity given an input.

states count the number of operational disks.

inputs: “external forces” causing transitions between states.

inputs consist of disk failures.

- Each state i has probability $P_i(t)$ of being there at time t .

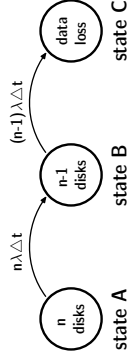
- **disk failure rate:** λ typically 1/1000000 per hour.
- **disk repair rate:** μ typically 2/3 per hour ... or 1/48 per hour over weekends.

Markovian assumptions cast in disk array terminology:

1. A disk fails in time interval Δt with probability $\lambda \Delta t$. A disk is repaired in time interval Δt with probability $\mu \Delta t$.
2. All failure and repair events are independent. Catastrophic failures are not modelled here.
3. The probability of a pair of failure and repair events occurring in an interval Δt is negligible. i.e. $(\lambda \Delta t)^2$ is so small we can ignore it etc.

Disk array reliability function calculation V

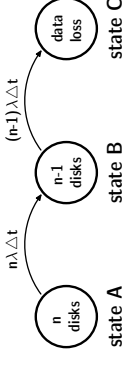
- **Markov model:** n disks & more than one failure results in data loss. (RAID 5 or 1)



- $P_A(t)$, $P_B(t)$, $P_C(t)$ probabilities of being in state A, B, or C at time t . Initially, $P_A(0)$ is 1; $P_B(0)$ and $P_C(0)$ are 0.
- **reliability function (for this model)**

$$r(t) = P_A(t) + P_B(t)$$

Disk array reliability function calculation VI



$$\begin{aligned}
 P_A(t + \Delta t) &= P_A(t)(1 - n\lambda\Delta t) \\
 P_B(t + \Delta t) &= P_A(t)n\lambda\Delta t + P_B(t)(1 - (n-1)\lambda\Delta t) \\
 P_C(t + \Delta t) &= P_B(t)(n-1)\lambda\Delta t + P_C(t)
 \end{aligned}$$

dividing by Δt and taking limits (as Δt goes to zero)

$$\begin{aligned}
 \dot{P}_A(t) &= -n\lambda P_A(t) \\
 \dot{P}_B(t) &= n\lambda P_A(t) - (n-1)\lambda P_B(t) \\
 \dot{P}_C(t) &= (n-1)\lambda P_B(t)
 \end{aligned}$$

using the initial conditions, we get, on the next page —

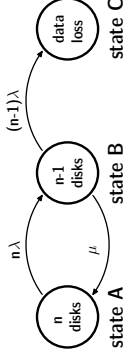
Disk array reliability function calculation VII

$$\begin{aligned}
 P_A(t) &= e^{-n\lambda t} \\
 P_B(t) &= n(e^{-(n-1)\lambda t} - e^{-n\lambda t}) \\
 P_C(t) &= 1 - P_A(t) - P_B(t) \\
 r(t) &= P_A(t) + P_B(t) = \\
 &\quad ne^{-(n-1)\lambda t} - (n-1)e^{-n\lambda t} \\
 \text{MTTDL} &= \int_0^\infty ne^{-(n-1)\lambda t} - (n-1)e^{-n\lambda t} dt \\
 &= \frac{n}{\lambda(n-1)} - \frac{n-1}{\lambda n} = \frac{2n-1}{\lambda n(n-1)}
 \end{aligned}$$

examples: $n = 2$; MTTDL is $\frac{3}{2\lambda}$;

$n = 10$; MTTDL is $\frac{19}{90\lambda}$.

Now we include repairs within our model for Level 1 or 5 layout.



the differential equations are almost as before —

$$\dot{P}_A(t) = -n\lambda P_A(t) + \mu P_B(t)$$

$$\dot{P}_B(t) = n\lambda P_A(t) - ((n-1)\lambda + \mu) P_B(t)$$

$$\dot{P}_C(t) = (n-1)\lambda P_B(t)$$

These equations are not nearly as easy to solve. Consequently, we develop an algebraic approach for the entire MTDDL calculation.

- **More general approach:** there are $n + 1$ states including one *data loss* state. We define a column vector of probabilities

$$P(t) = (P_1(t), P_2(t), \dots, P_n(t))^T$$

where each $P_i(t)$ represents, for states other than the data-loss state, the probability of being in state i at time t . Write the system of n differential equations as follows:

$\dot{P} = M P$ with initial conditions $P(0)$ using an $n \times n$ constant matrix M .

- **calculation formula**

$$\text{MTDDL} = -\bar{I}^{-1} M^{-1} P(0)$$

$$\text{where } \bar{I} = (1, 1, \dots, 1) \quad n\text{-ones}$$

this formula can be verified using the LaPlace transform on the system of differential equations.

Our previous system of differential equations

$$\dot{P}_1(t) = -n\lambda P_1(t) + \mu P_2(t)$$

$$\dot{P}_2(t) = n\lambda P_1(t) - ((n-1)\lambda + \mu) P_2(t)$$

$$\dot{P}_3(t) = (n-1)\lambda P_2(t)$$

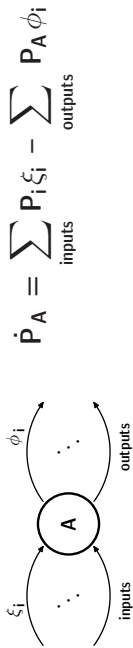
We have the following matrix:

$$M = \begin{bmatrix} -n\lambda & \mu \\ n\lambda & -((n-1)\lambda + \mu) \end{bmatrix}$$

$$M^{-1} = \frac{\begin{bmatrix} -((n-1)\lambda + \mu) & -\mu \\ -n\lambda & -n\lambda \end{bmatrix}}{n(n-1)\lambda^2}$$

$$\text{MTDDL} = -(1, 1) M^{-1} (1, 0)^T = \frac{\mu + (2n-1)\lambda}{n(n-1)\lambda^2}$$

1. Derive state diagram.



2. Write (differential) equations as above to obtain M.

3. Invert M to obtain the MTTDL using

$$\text{MTTDL} = -\bar{\mathbf{I}} \mathbf{M}^{-1} \mathbf{P}(0)$$

Workload: n disks R small reads and W small writes

Small Operation Fault Free Performance:

workload: $R + 4W$ total operations for small accesses.
workload is assumed evenly divided among the disks.

Degraded Mode designates single disk failure

Small Operation Degraded Mode Performance:

Read workload: $2R(n-1)/n$ total operations.
Write workload: $4W + W(n-7)/n$ total operations.

Performance Consequences: Degraded mode is inevitable. To meet the needs of our clients during the degraded mode, we must limit the utility of the system during fault free operation.

Disk failure model:

the disk is operational or it provides no data.
delivery delays (timing) not an issue.

mean time to failure for disk drives has increased due to improved design and manufacturing techniques.

STORAGE SYSTEM ARCHITECTURES

Parity Declustering

Walt Burkhard Spring 2010

Computer Science and Engineering Department UCSD
All rights reserved. No part of these notes may be reproduced, stored, or transmitted in any form or means without the permission of the author.

RAID Level 5: Left Symmetric Data Layout

	disk 0	disk 1	disk 2	disk 3	disk 4
A.0	A.0	A.1	A.2	A.3	B.0
B.1	B.1	B.2	B.3	C.0	C.1
C.2	C.2	C.3	D.0	D.1	D.2
D.3	pE	pD	pC	pB	pA

- Stripes Reliability usable storage ratio
- Parallel reads Writes.
- Degraded Performance (single disk failure)
Stripe-unit accesses – reads or writes. Both increase the workload.
- Improvements Performance, Availability/Reliability, Cost

Declustering improves the degraded mode performance.

Disk Array Declustering

The client data is spread over additional disks while maintaining the storage ratio.

Example: 6 disks with storage ratio 0.8

	disk 0	disk 1	disk 2	disk 3	disk 4	disk 5
A.0	A.0	A.1	A.2	A.3	B.0	B.1
B.2	B.2	B.3	C.0	C.1	C.2	C.3
E.1	E.1	D.0	D.1	D.2	D.3	E.0
F.3	pC	E.2	E.3	F.0	F.1	F.2
		pF	pB	pE	pA	pD

Workload: R small reads and W small writes

Small Operation Fault Free Performance:

workload: R + 4W operations for small accesses.
workload is evenly divided among the disks.

Degraded Mode designates single disk failure

Small Operation Degraded Mode Performance:

Read workload: $R(n + k - 2) / n$ total operations.
Write workload: $4W + W(k - 7) / n$ total operations.

n disks k stripe width
r stripe units/disk b number of stripes

RAID-5 Data Layout Goals

Error Correction stripe units of each stripe mapped to distinct disks.

Distributed Parity each disk contains the same number of parity stripe units.

Distributed Reconstruction the incremental workload is evenly distributed over the surviving disks.

Efficient Mapping run-time efficient requiring little additional memory to represent the data layout.

Parallel Reads contiguous client data mapped to distinct disks as much as possible.

Large Writes contiguous client data mapped to single stripes as much as possible.

Balanced Incomplete Block Designs (BIBD)

(b, n, r, k, λ)

- there are b subsets of { 0, 1, ..., n - 1 } of size k,
- each element in { 0, 1, ..., n - 1 } appears in r subsets, and
- each pair of elements of { 0, 1, ..., n - 1 } is in λ subsets.

conditions:

$n \cdot r = b \cdot k$

$r (k - 1) = \lambda (n - 1)$

	disk 0	disk 1	disk 2	disk 3
A.0	A.0	A.1	pA	pB
B.0	B.0	B.1	C.1	pC
C.0	C.0	D.0	D.1	pD
E.0	E.0	pE	E.1	F.1
F.0	F.0	pF	pG	G.1
G.0	G.0	H.0	pH	H.1
	pI	I.0	I.1	J.1
	pJ	J.0	K.0	K.1
	pK	pL	L.0	L.1

0 1 2

0 1 3

0 2 3

1 2 3

- Our Goals
 - Simple (easy to specify) data layouts.
 - Run-time efficient access calculations.

RAIDframe simulator

<i>Workload</i>	
Access sizes:	8,24,48,72,120,133,168,192,240,288,336KB
Concurrency:	1,2,4,8,19,15,20,25 clients
Alignment:	stripe unit boundaries
random accesses uniformly distributed over address space	
<i>Disks HP 2247</i>	
Capacity:	1.03GB
Geometry:	1981 cylinders, 13 heads, 8 zones
Average seek time:	10 ms
Rotational speed:	5400 rpm
Queueing discipline:	shortest seek time first
<i>Data Layout</i>	
Stripe unit size:	8KB
Number of disks, n :	13
Stripe width, k :	4 stripe units; RAID 5 is 13

- data layout schema 13 disks, stripe width 4.

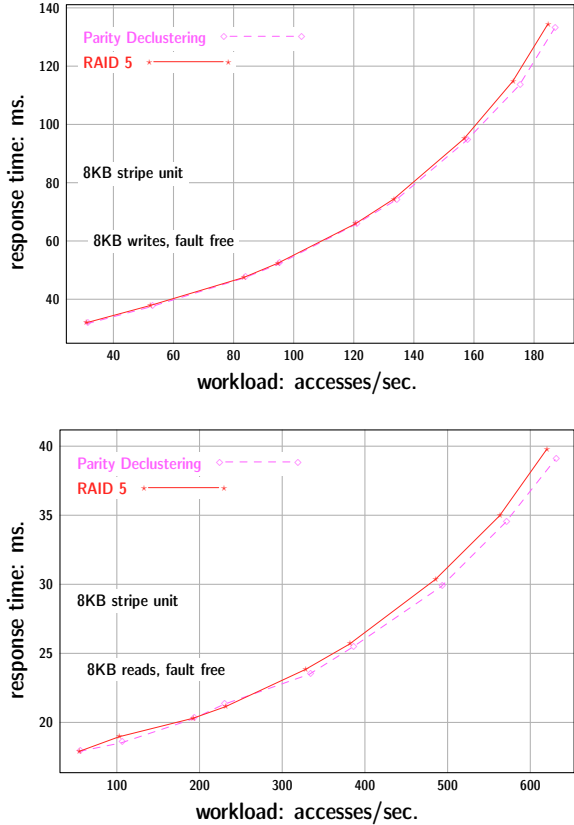
0	1	3	9
1	2	4	10
2	3	5	11
3	4	6	12
0	4	5	7
1	5	6	8
2	6	7	9
3	7	8	10
4	8	9	11
5	9	10	12
0	6	10	11
1	7	11	12
0	2	8	12

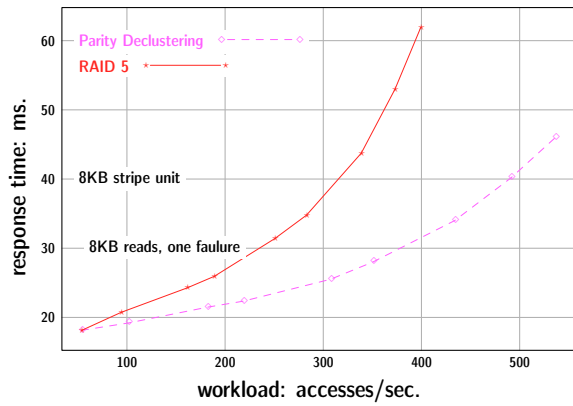
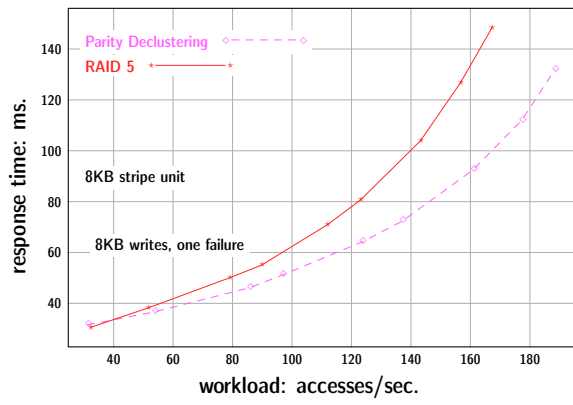
k is 4, n is 13,

b is 13, r is 4,

λ is 1

- Data Layout Goals
 - all satisfied except parallel reads!







STORAGE SYSTEM ARCHITECTURES

Datum Declustering Data Layout

Walt Burkhard Spring 2010

Computer Science and Engineering Department UCSD
All rights reserved. No part of these notes may be reproduced, stored, or transmitted in any form or means without the permission of the author.

- **Small Operation Fault Free Performance:**
the total array workload is $R + 4W$ operations for small reads and writes; each disk gets $1/n^{th}$ the total.
Degraded Mode designates single disk failure
- **Small Operation Degraded Mode Performance:**
Both the reads and writes increase the workloads. The total read load for the surviving $n-1$ disks is $2R \frac{(n-1)}{n}$ operations. The total write load for the survivors is $4W + W \frac{(n-7)}{n}$ operations.
- **Performance Consequences:**
Since the workload increases for each surviving disk in degraded mode, we must utilize less than the full array capability in fault free operation. Otherwise, we will not be able to meet the needs of our clients during the inevitable degraded operation. For read only loads, we can use only one-half the capability.



■ **RAID Level 5: "Left Symmetric" aka "Prime" Data Layout**

	disk 0	disk 1	disk 2	disk 3	disk 4
A.0	A.1	A.2	A.3	B.0	
B.1	B.2	B.3	C.0	C.1	
C.2	C.3	D.0	D.1	D.2	
D.3	E.0	E.1	E.2	E.3	
PE	PD	PC	PB	PA	

- **Access functions** user addresses: $a \quad 0 \leq a < n(r-1)$ (full disks): n disks, r stripe units per disk, $b = r$ stripes

$$p = a/(n(n-1))$$
$$disk(a) = a \bmod n$$
$$offset(a) = a/n + p$$
$$parityDisk(a) = (a/(n-1) + 1)(n-1) \bmod n$$
$$parityOffset(a) = (p+1)n - 1$$

Work Load: R reads and W writes



Declustering improves the degraded mode performance.

- **Disk Array Declustering**
The client data is spread over additional disks; n disks with stripe width k , typically $n > k$.
Example: $n = 6, k = 5$.

	disk 0	disk 1	disk 2	disk 3	disk 4	disk 5
A.0	A.1	A.2	A.3	B.0	B.1	
B.2	B.3	C.0	C.1	C.2	C.3	
E.1	D.0	D.1	D.2	D.3	E.0	
F.3	E.2	E.3	F.0	F.1	F.2	
PC	PF	PB	PE	PA	PD	

Data Layout Goals

- Error Correction
 - stripe units of each stripe mapped to distinct disks.
- Distributed Parity
 - each disk contains the same number of parity stripe units.
- Distributed Reconstruction
 - the incremental workload is evenly distributed over the surviving disks.
- Efficient Mapping
 - run-time efficient requiring little additional memory to represent the data layout.
- Parallel Reads
 - contiguous client data mapped to distinct disks as much as possible.
- Large Writes
 - contiguous client data mapped to single stripes as much as possible.

G. Alvarez, W. Burkhard, & F. Cristian.

Data Layout

n disks
 k stripe width

Complete block designs declustering

Layout calculated via binomial number system.

A complete block design contains all possible k -combinations of n items. There are $\binom{n}{k}$ such combinations. Each combination designates a set of disks for a stripe.

Balanced Incomplete Block Design parameters

$$b = \binom{n}{k} \quad r = \binom{n-1}{k-1} \quad \lambda = \binom{n-2}{k-2}$$

Example: $n = 4, k = 3$

	disk 0	disk 1	disk 2	disk 3
PA	A.0	A.1	B.1	
PB	B.0	C.0	C.1	
PC	PD	D.0	D.1	
E.0	PE	E.1	F.1	
F.0	PF	PG	G.1	
G.0	H.0	PH	H.1	
I.0	I.1	PI	PJ	
J.0	J.1	K.1	PK	
K.0	L.0	L.1	PL	

■ scheme

Layout Parameter Details:

$n = 4, k = 3$. There are 4 stripes per iteration. Each disk contains 3 stripe units per iteration. The parity units are rotated among the columns of the scheme.

k-combinations $\binom{n}{k}$ possible k-combinations.

$$\langle X_0, \dots, X_{k-1} \rangle \text{ assuming } X_{i-1} < X_i \text{ for } 0 < i < k.$$

total order $\succ \langle X_0, \dots, X_{k-1} \rangle \succ \langle Y_0, \dots, Y_{k-1} \rangle$

if there is a $j < k$ such that X_j equals Y_j for $j < i < k$ and $X_j > Y_j$.

mapping: $\langle X_0, \dots, X_{k-1} \rangle$ is in position

$$\binom{X_0}{1} + \binom{X_1}{2} + \dots + \binom{X_{k-1}}{k}$$

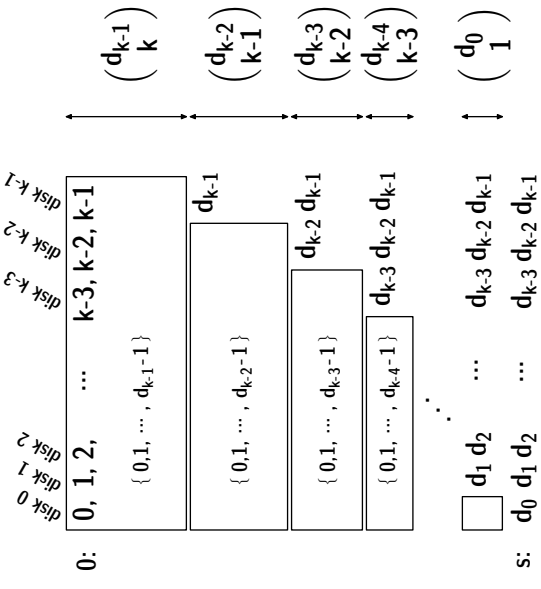
inverse mapping exists since each k-combination is in a unique position according of $\succ$.

- **inverse mapping calculation** $\mapsto$

```
inverseMapping ( int position , int k )
{
    for ( int i = k-1 ; i >= 0 ; i-- ) {
        ll = i ;
        while ( binomial( ll , i+1 ) <= position )
            ll = ll + 1 ;
        c[i] = ll - 1 ;
        position = position - binomial ( c[i] , i+1 ) ;
    }
}
```

- **c array** contains the k-combination associated with position X_i in slot $c[i]$.

- **Position of $\langle d_0, d_1, \dots, d_{k-1} \rangle$**



Binomial Number System

- **Datum Access functions**

n disks r stripe units per disk
 k stripe width b stripes in layout
 $bk = nr$ stripe units. $m = k - 1$
 user addresses: a $0 \leq a < b(k-1)$ (full disks)

$$\text{stripe} = \lfloor a/m \rfloor \bmod \binom{n}{k}$$

$$\text{stripe} \mapsto \{d_0, d_1, \dots, d_{k-1}\}$$

$$\text{iterations} = \lfloor \lfloor a/m \rfloor / \binom{n}{k} \rfloor$$

$$\text{index}P = \text{iterations} \bmod k$$

$$\text{index}D = (a \bmod m + \text{index}P + 1) \bmod k$$

Both **indexP** and **indexD** are indices within the d k -tuple.

- **Binomial number system**

For $0 \leq s < \binom{n}{k}$ with $i \leq d_i < d_{i+1} < n$,

the binomial number system mapping

$$s \mapsto \{d_0, d_1, \dots, d_{k-1}\} \quad s = \binom{d_0}{1} + \binom{d_1}{2} + \dots + \binom{d_{k-1}}{k}$$

is a bijection.

- **Examples:**

$$14 \mapsto \{4, 5\} \quad k = 2$$

$$14 \mapsto \{1, 3, 5\} \quad k = 3$$

$$15 \mapsto \{0, 6\} \quad k = 2$$

$$15 \mapsto \{2, 3, 5\} \quad k = 3$$

■ Datum Access functions

$$\text{dataDisk}(a) = d_{\text{index}D} \quad \text{parityDisk}(a) = d_{\text{index}P}$$

$$\text{dataOffset}(a) = \text{iterations} \times \binom{n-1}{k-1} + \text{offset}(a, \text{index}D)$$

$$\text{parityOffset}(a) = \text{iterations} \times \binom{n-1}{k-1} + \text{offset}(a, \text{index}P)$$

$$\text{offset}(a, i) = \begin{cases} \sum_{j=1}^{k-1} \binom{d_{j-1}}{j} & i = 0 \\ \sum_{j=0}^{i-1} \binom{d_j}{j+1} + \sum_{j=i+1}^{k-1} \binom{d_{j-1}}{j} & 0 < i < k-1 \\ \sum_{j=0}^{k-2} \binom{d_j}{j+1} & i = k-1 \end{cases}$$

■ **Example:** $n = 8, k = 4, a = 354$.

$$\text{stripe} = \lfloor 354 / 3 \rfloor \bmod \binom{8}{4} = 118 \bmod 70 = 48$$

$$\text{stripe} \mapsto \{0, 3, 5, 7\}$$

$$\text{iterations} = \lfloor \lfloor 354 / 3 \rfloor / 70 \rfloor = 1$$

$$\text{index}P = \text{iterations} \bmod k = 1 \bmod 4 = 1$$

$$\begin{aligned} \text{index}D &= (a \bmod m + \text{index}P + 1) \bmod k \\ &= 354 \bmod 3 + 1 + 1 = 2 \end{aligned}$$

$$\text{disk}(354) = d_2 = 5$$

$$\text{parity}(354) = d_1 = 3$$

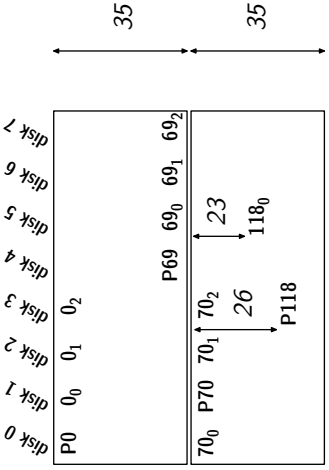
■ $\mapsto$ returns $d[i]$ values.

■ Examples $n = 6, k = 3$:

$$\begin{array}{ll} s = 4 & s = 5 \\ \binom{4}{3} = 4; d[2] \text{ is } 4 & \binom{4}{3} = 4; d[2] \text{ is } 4 \\ \binom{1}{2} = 0; d[1] \text{ is } 1 & \binom{2}{2} = 1; d[1] \text{ is } 2 \\ \binom{0}{1} = 0; d[0] \text{ is } 0 & \binom{0}{1} = 0; d[0] \text{ is } 0 \\ s = 7 & s = 8 \\ \binom{4}{3} = 4; d[2] \text{ is } 4 & \binom{4}{3} = 4; d[2] \text{ is } 4 \\ \binom{3}{2} = 3; d[1] \text{ is } 3 & \binom{3}{2} = 3; d[1] \text{ is } 3 \\ \binom{0}{1} = 0; d[0] \text{ is } 0 & \binom{1}{1} = 1; d[0] \text{ is } 1 \end{array}$$

$$\begin{aligned} \text{offset}(354, 1) &= \sum_{j=0}^0 \binom{d_j}{j+1} + \sum_{j=2}^3 \binom{d_{j-1}}{j} \\ &= \binom{0}{1} + \binom{4}{2} + \binom{6}{3} = 26 \\ \text{offset}(354, 2) &= \sum_{j=0}^1 \binom{d_j}{j+1} + \sum_{j=3}^3 \binom{d_{j-1}}{j} \\ &= \binom{0}{1} + \binom{3}{2} + \binom{6}{3} = 23 \\ \text{diskOffset}(354) &= \text{offset}(354, 2) + \binom{7}{3} \\ &= 23 + 35 = 58 \\ \text{parityOffset}(354) &= \text{offset}(354, 1) + \binom{7}{3} \\ &= 26 + 35 = 61 \end{aligned}$$

- **Example details:** $n = 8$, $k = 4$, $a = 354$.



$$\text{stripe} = \lfloor 354 / 3 \rfloor \bmod \binom{8}{4} = 118 \bmod 70 = 48$$

$$\text{stripe} \mapsto \{0, 3, 5, 7\} \quad \text{indexP} = 1 \quad \text{indexD} = 2$$

$$\text{disk}(354) = d_2 = 5 \quad \text{parity}(354) = d_1 = 3$$

$$\text{diskOffset}(354) = 58 \quad \text{parityOffset}(354) = 61$$

Large Write Optimization

Every stripe contains a contiguous interval of client data.

Parallel Reads

Contiguous client data is not mapped to distinct disks. **We do not meet all the data layout goals.**

- **Degraded Mode Small Operation Performance**

Work Load: R read and W writes

The total read workload for the surviving $n-1$ disks is $R(n+k-2)/n$ operations and the total write workload for the survivors is $4W + W(k-7)/n$ operations.

- **Structural Properties:** n disks & stripe width k

$$\text{Number of stripes per iteration: } \binom{n}{k}$$

$$\text{Number of stripes per period: } k \binom{n}{k}$$

$$\text{Number of stripe units per disk per iteration: } \binom{n-1}{k-1}$$

- **Data Layout Goals revisited**

Error Correction Stripe units distributed over k distinct disks.

Even Parity Distribution Parity stripe units per disk per period: $\binom{n-1}{k-1}$

Even Reconstruction Workload Distribution Number of times a pair of disks occurs together per iteration: $\binom{n-2}{k-2}$

stripe unit **a** maps to stripe **s** $s \mapsto \{d_0, d_1, \dots, d_{k-1}\}$

$\text{offset}(s, i)$ is the number of occurrences of d_i preceding s in this iteration.

0:	0	1	2	$n = 5$, $k = 3$.	Stripe 0 resides	0:	0	0	0
1:	0	1	3	on disks 0, 1, & 2, stripe 1 on	disks 0, 1, & 3, etc.	1:	1	1	0
2:	0	2	3			2:	2	1	1
3:	1	2	3			3:	2	2	2
4:	0	1	4			4:	3	3	0
5:	0	2	4			5:	4	3	1
6:	1	2	4			6:	4	4	2
7:	0	3	4			7:	5	3	3
8:	1	3	4			8:	5	4	4
9:	2	3	4			9:	5	5	5

The offset formula, when $s = \sum_{j=0}^{k-1} \binom{d_j}{j+1}$, for $0 \leq i \leq k-1$

is
$$\text{offset}(s, i) = \sum_{j=0}^{i-1} \binom{d_j}{j+1} + \sum_{j=i+1}^{k-1} \binom{d_j-1}{j}. \tag{1}$$

example: $s = 5$ d_0 is 0, d_1 is 2, d_2 is 4

$$\text{offset}(5, 0) = \sum_{j=1}^2 \binom{d_j-1}{j} = \binom{1}{1} + \binom{3}{2} = 4.$$

$$\text{offset}(5, 1) = \sum_{j=0}^0 \binom{d_j}{j+1} + \sum_{j=2}^2 \binom{d_j-1}{j} = \binom{0}{1} + \binom{3}{2} = 3.$$

$$\text{offset}(5, 2) = \sum_{j=0}^1 \binom{d_j}{j+1} = \binom{0}{1} + \binom{2}{2} = 1.$$

all k combinations in
 $\{0,1, \dots, d_{k-1}-1\}$

all k-1 combinations in
 $\{0,1, \dots, d_{k-2}-1\}$

$\{0,1, \dots, d_1-1\}$

$\textcircled{A}$

$\{0,1, \dots, d_{1-1}-1\}$

d_1

d_{1-1}

d_1

$\vdots$

d_{k-2}

d_{k-1}

d_{k-2}

d_{k-1}

$\vdots$

d_{1+1}

d_{k-1}

d_1

d_{k-1}

d_1

d_{k-1}

$\vdots$

d_{1-1}

d_{k-1}

d_1

d_{k-1}

$\vdots$

d_1

d_{k-1}

$\vdots$

d_1

d_{k-1}

$s: d_0$

d_1

$\dots$

d_{1-1}

d_1

$\dots$

d_{1+1}

$\dots$

d_{k-2}

d_{k-1}

$$\text{offset}(s, i) = \sum_{j=0}^{i-1} \binom{d_j}{j+1} + \sum_{j=i+1}^{k-1} \binom{d_j-1}{j}$$

Beneath A

Above A

STORAGE SYSTEM ARCHITECTURES

Prime Declustering Data Layout

Walt Burkhard Spring 2010

Computer Science and Engineering Department UCSD
All rights reserved. No part of these notes may be reproduced, stored, or transmitted in any form or means without the permission of the author.

	disk 0	disk 1	disk 2	disk 3	disk 4
A.0	A.0	A.1	A.2	B.0	B.1
B.2	B.2	C.0	C.1	C.2	D.0
D.1	D.1	D.2	E.0	E.1	E.2
pE	pE	pB	pD	pA	pC
F.0	F.0	G.0	F.1	G.1	F.2
G.2	G.2	H.2	H.0	I.0	H.1
I.1	I.1	J.1	I.2	J.2	J.0
pJ	pJ	pF	pG	pH	pI

Stride based data layout.

Prime Data Layout Goals

- Almost optimal read of contiguous client stripe units.
- Large write optimization.
- Evenly distributed parity and reconstruction workload.

n disks (prime) k stripe width.

stripe names (x, y) with $0 \leq x < n$ and $1 \leq y < n$.

there are $(n-1)n$ stripes in the layout.

disks in stripe $(x, y) =$

$$\{ d \mid d = (x + iy) \bmod n \quad 0 \leq i < k \}$$

n -block n user addresses with $a \equiv 0 \bmod n$
 $\{ a, a+1, \dots, a+n-1 \}$

stride block $n(k-1)$ user addresses with $a \equiv 0 \bmod n(k-1)$
 $\{ a, a+1, \dots, a+n(k-1)-1 \}$.

G. Alvarez, W. Burkhard, L. Stockmeyer, & F. Cristian.

Existential Result n disks k stripe width r stripe units per disk.

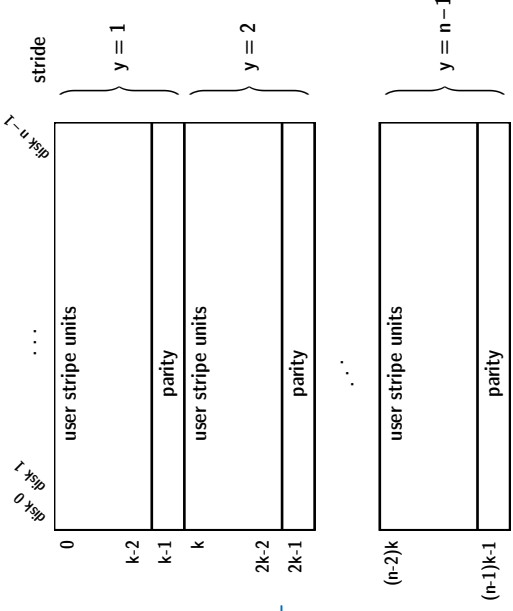
Theorem: There is a data layout satisfying the data layout goals with

parameters n, k , and r if and only if

- a. k divides r , and
- b. $(n-1)$ divides $r(k-1)$,

and one of the following conditions holds –

- i) $k = n$,
- ii) $k = n-1$,
- iii) $k = 2$,
- iv) $k = 3$ & $n = 5$ or $n = 7$,
- v) $k = 4$ & $n = 7$.



Prime
layout
strategy –

Within a stride, each row contains user addresses of an n -block; the $n(k-1)$ user addresses within the stride constitute a stride-block.

Parity stripe units are always in the last row associated with a stride-block; for stripe (x, y) parityDisk is $(y(k-1) + x) \bmod n$

Let p designate the number of complete stride-blocks prior to the current stride-block for user address a

$$p = \left\lfloor \frac{a}{(k-1)n} \right\rfloor \quad (1)$$

Stride numbers are cyclically assigned; the stride number y is

$$y = p \bmod (n-1) + 1 \quad (2)$$

The first user address in stride-block p is $p(k-1)n$.

Within stride-block p , each stripe ψ has its first user address

$$p(k-1)n + \psi(k-1) \quad \text{for } 0 \leq \psi < n.$$

Each stripe contains $k-1$ user addresses for $0 \leq i < k-1$

$$p(k-1)n + \psi(k-1) + i$$

n -block addresses are assigned to a stride-block with stride y –

$$\text{dataDisk}(a) = ay \bmod n. \quad (3)$$

The first user address a within stripe ψ in stride-block p is $p(k-1)n + \psi(k-1)$.

The disk x containing the first user address within stripe ψ is

$$\begin{aligned} x &= \text{dataDisk}(a) = y(p(k-1)n + \psi(k-1)) \bmod n \\ &= \psi y(k-1) \bmod n. \end{aligned}$$

The disk containing parity for stripe (x, y) is

$$\begin{aligned} (x + y(k-1)) \bmod n &= (\psi y(k-1) + y(k-1)) \bmod n \\ &= ((\psi + 1)y(k-1)) \bmod n. \end{aligned}$$

The parity of stripe ψ is on the same disk as the first user address of stripe $\psi + 1$.

■ Calculate ψ

$$a = p(k-1)n + \psi(k-1) + i \quad \text{for } \begin{cases} 0 \leq \psi < n \\ 0 \leq i < k-1 \end{cases}$$

$$\psi = \left\lfloor \frac{a}{(k-1)} \right\rfloor \bmod n$$

$$\text{parityDisk}(a) = (\psi + 1)y(k-1) \bmod n$$

$$= (\left\lfloor \frac{a}{(k-1)} \right\rfloor + 1)y(k-1) \bmod n \quad (4)$$

Offsets:

$$\text{offsetData}(a) = \left\lfloor \frac{a}{n} \right\rfloor + p \quad (5)$$

$$\text{offsetParity}(a) = (p+1)(k-1) \quad (6)$$

Equations 1, 2, 3, 4, 5, & 6 define the Prime data layout.

■ Prime Properties

1. The stripe units of each stripe reside on k disks.

The following three properties hold per layout cycle.

2. Each disk contains $n-1$ parity stripe units.
3. Each disk contains $r = k(n-1)$ stripe units.
4. Each pair of disks occurs in $\lambda = k(k-1)$ stripes.
5. The large write optimization is satisfied: write to $k-1$ stripe units within a stripe without reading anything.

6. Parallel reads almost satisfied: a read of o consecutive stripe units requires each disk supply $\tau(o)$ stripe units – at most one more than optimal.

$$\left\lceil o/n \right\rceil \leq \tau(o) \leq \left\lfloor o/n \right\rfloor + 1$$

STORAGE SYSTEM ARCHITECTURES

Spares and Permutation Development Data Layout

Walt Burkhard

Spring 2010

Computer Science and Engineering Department

UCSD

All rights reserved. No part of these notes may be reproduced, stored, or transmitted in any form or means without the permission of the author.

- Distribute spare space over all the disks. Menon & Mattson 1992

	disk 0	disk 1	disk 2	disk 3	disk 4
0:	d0	d1	d2	parity	spare
1:	d3	d4	parity	spare	d5
2:	d6	parity	spare	d7	d8
3:	parity	spare	d9	d10	d11
4:	spare	d12	d13	d14	parity

- enough space to reconstruct all data of one (failed) disk within spare space.

	disk 0	disk 1	disk 2	disk 3	disk 4
0:	d0	d1	d2	parity	d1
1:	d3	d4	parity	d4	d5
2:	d6	parity	parity	d7	d8
3:	parity	spare	d9	d10	d11
4:	d12	d12	d13	d14	parity

- spare space raises mean repair rate μ .
$$MTTDL = \frac{\mu + (2n-1)\lambda}{n(n-1)\lambda^2}$$
 RAID 5

MTTDL increases linearly with μ . With one disk failure and a reconstructed spare disk, the ensemble continues to tolerate a single failed disk without data loss.

- Additional space improves (raises) repair rate.
How much spare space is required?
How is spare space implemented?

Example: RAID Level 5 with dedicated spare disk.

	disk 0	disk 1	disk 2	disk 3	disk 4
0:	d0	d1	d2	parity	spare
1:	d3	d4	parity	d5	spare
2:	d6	parity	d7	d8	spare
3:	parity	d9	d10	d11	spare

- spare disk not used until failure occurs.
- **Problems:** spare disk may not function properly when needed.
workload distributed over all disks except spare.

- Modern disk drives have extremely low probability of failure; the likelihood of “simultaneous” failures is extremely small.
- How much spare space?** We need enough space to accommodate one failure; the probability of a second failure occurring during the reconstruction for the first is so small we ignore it.
- Include spare space within declustered layout.** We desire to utilize several groups of disks and include space of one spare disk within the ensemble.

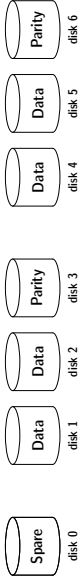
- example continued:** mapping stripe units from virtual disks and offsets to physical disks and offsets.

```
int permutation[] = { 0, 1, 2, 3, 4, 5, 6 } ;  
  
pdisk = ( permutation[vdisk] + voffset ) % 7 ;  
poffset = voffset ;
```

- Data layout schema** created using permutation development where components of each row are one more than in previous row. Rows represent contiguous stripe unit offset on a disk.

0	1	2	3	4	5	6
1	2	3	4	5	6	0
2	3	4	5	6	0	1
3	4	5	6	0	1	2
4	5	6	0	1	2	3
5	6	0	1	2	3	4
6	0	1	2	3	4	5

- PDDL example** seven virtual disks consisting of two RAID level 4 groups of three disks each.



- Declustering Data Layout Goals**

- Single failure correcting.
- Evenly distribute spare and parity stripe units.
- Reconstruction workload evenly divided over surviving disks.
- Efficient mapping.
- Parallel Reads.
- Large Write Optimization.

	disk 0	disk 1	disk 2	disk 3	disk 4	disk 5	disk 6
0:	spare	d0	d1	parity	d2	d3	parity
1:	parity	spare	d4	d5	parity	d6	d7
2:	d11	parity	spare	d8	d9	parity	d10
3:	d14	d15	parity	spare	d12	d13	parity
4:	parity	d18	d19	parity	spare	d16	d17
5:	d21	parity	d22	d23	parity	spare	d20
6:	d26	d27	parity	d24	d25	parity	spare

- Check the Declustering Data Layout Goals.**

goals 1, 2, 4, 6 met. goal 5 is not met within PDDL.

goal 3: this permutation does not evenly divide the incremental reconstruction workload. Consider disk 0 failing — disks 1 and 6 must access five stripe units, disks 2 and 5 access three stripe units, and finally disks 3 and 4 both access only one stripe unit; the reconstruction workload is not evenly balanced.

	disk 0	disk 1	disk 2	disk 3	disk 4	disk 5	disk 6
0:	spare	d0	d1	parity	d2	d3	parity
1:	parity	parity	d4	d5	parity	d6	d7
2:	d1	parity	d11	d8	d9	parity	d10
3:	d4	d15	parity	d14	d12	d13	parity
4:	parity	d18	d19	parity	parity	d16	d17
5:	d1	parity	d22	d23	parity	d21	d20
6:	d6	d27	parity	d24	d25	parity	d26

failed disk

red cells designate access during reconstruction.

the number of red cells per column, count the incremental reconstruction workload per period.

Different permutation: yields better results.

int permutation[] = { 0, 1, 2, 4, 3, 5, 6 } ;

	disk 0	disk 1	disk 2	disk 3	disk 4	disk 5	disk 6
0:	spare	d0	d1	d2	parity	d3	parity
1:	parity	spare	d4	d5	d6	parity	d7
2:	d11	parity	spare	d8	d9	d10	parity
3:	parity	d15	parity	spare	d12	d13	d14
4:	d18	parity	d19	parity	spare	d16	d17
5:	d21	d22	parity	d23	parity	spare	d20
6:	d24	d25	d26	parity	d27	parity	spare

goals 1, 2, 4, 6 met. goal 5 is not met within PDDL.

goal 3: this permutation does evenly divide the incremental reconstruction workload. Consider disk 0 failing — all surviving disks 1 through 6 each access three stripe units; the reconstruction workload is evenly balanced.

	disk 0	disk 1	disk 2	disk 3	disk 4	disk 5	disk 6
0:	spare	d0	d1	d2	parity	d3	parity
1:	parity	parity	d4	d5	d6	parity	d7
2:	d1	parity	d11	d8	d9	d10	parity
3:	parity	d15	parity	parity	d12	d13	d14
4:	d8	parity	d19	parity	d18	d16	d17
5:	d1	d22	parity	d23	parity	d21	d20
6:	d4	d25	d26	parity	d27	parity	d24

failed disk

red cells designate access during reconstruction.

the number of red cells per column, count the incremental reconstruction workload per period. The workload is balanced here.

Where do such permutations come from?

Bose 1939. Let $n = k \cdot g + 1$, the number of disks, be prime and let w be a primitive element. The permutation, containing g groups each of size k ,

$$\begin{aligned}
 &0 ; w^0, w^g, \dots, w^{(k-1)g} ; w^1, w^{g+1}, \dots, w^{(k-1)g+1} ; \\
 &\dots ; w^{g-1}, w^{2g-1}, \dots, w^{kg-1}
 \end{aligned}$$

has even incremental reconstruction workload distribution.

Nearly Resolvable Designs.

■ examples:

n is 7; $w = 2$ is not primitive since

$$w^0 = 1, w^1 = 2, w^2 = 4, \text{ and } w^3 = 1.$$

n is 7; $w = 3$ is primitive since

$$w^0 = 1, w^1 = 3, w^2 = 2, \text{ and } w^3 = 6, w^4 = 4, w^5 = 5.$$

this primitive element provides the permutation for our previous layout.

$$0; 1, 2, 4; 3, 5, 6$$

where the semi-colons only emphasize the stripes. The ordering of columns within a stripe has no effect.

- Combine “complementary” permutation designs: $n = 10 = 3 \cdot 3 + 1$; $k = g = 3$.

$$0 \qquad 1 \ 2 \ 8 \qquad 3 \ 5 \ 7 \qquad 4 \ 6 \ 9$$

Reconstruction workload signature (for disk 0 failing)

$$1 \ 3 \ 2 \ 2 \ 2 \ 2 \ 2 \ 3 \ 1$$

$$0 \qquad 1 \ 2 \ 4 \qquad 3 \ 7 \ 8 \qquad 5 \ 6 \ 9$$

Reconstruction workload signature

$$3 \ 1 \ 2 \ 2 \ 2 \ 2 \ 2 \ 1 \ 3$$

to obtain balanced reconstruction workload.

Bose’s approach works as well for n a power of a prime such as $n = 9$ where we might want to have two groups of size four each. However, the modular arithmetic is replaced by appropriate finite field multiplication when determining the permutation. The permutation development scheme remains the same.

Bose’s approach does not work for other values of n such as $n = 10$ where we might wish to have three groups of size three each.

We consider an alternate approach that has provided data layouts satisfying all data layout goals except for parallel reads.

groups	stripe width							
	5	6	7	8	9			
1	1	1	1	1	1	1		
2		1	1	2	2	1		
3			1	1	2	2	2	
4				1	1	1	1	4
5					1	1	1	2
6						1	2	4
7							1	2
8								2
9								
10								

smallest number of permutations for optimal declustering with one spare disk.

Searching for suitable permutation(s) when n is not power of a prime, can be done using “exhaustive” branch and bound search strategies. Any other systematic approach constitutes a technological breakthrough.



STORAGE SYSTEM ARCHITECTURES

EVENODD Data Layout

Walt Burkhard

Spring 2010

Computer Science and Engineering Department

UCSD

All rights reserved. No part of these notes may be reproduced, stored, or transmitted in any form or means without the permission of the author.

Redundancy Calculations

$$S = \bigoplus_{t=1}^{n-1} a_{(n-1-t) \bmod n, t}$$

For each row r , the redundant values are calculated as follows:

$$a_{r, n} = \bigoplus_{t=0}^{n-1} a_{r, t} \quad a_{r, n+1} = S \oplus \left(\bigoplus_{t=0}^{n-1} a_{(r-t) \bmod n, t} \right)$$

- Disk n contains “horizontal” redundancy and disk $n+1$ contains “diagonal” redundancy.

EVENODD – tolerates double disk failures

EVENODD III



Blaum, Brady, Bruck and Menon. IEEE Trans. on Computers 1995

A redundancy technique tolerating up to two disk failures using only exclusive-OR calculations. EVENODD employs redundant values requiring the space of two additional disks; this is identical to the RAID-6 space requirements.

A single group contains $n + 2$ disks where $n \geq 3$ is prime; the group’s data and redundant values are specified via an $n - 1 \times n + 2$ array of stripe unit values $a_{i,j}$ which designates the i^{th} stripe unit on disk j .

We begin by assuming $a_{n-1, c} = 0$ for $0 \leq c \leq n+1$. This is a notational convenience only.

0	T ₀	U ₁	V ₂	W ₃	X ₄	Y ₅	S ₅	H ₀	T _D ⊕ S _D
1	U ₀	V ₁	W ₂	X ₃	Y ₄	S ₄	T ₅	H ₁	U _D ⊕ S _D
2	V ₀	W ₁	X ₂	Y ₃	S ₃	T ₄	U ₅	H ₂	V _D ⊕ S _D
3	W ₀	X ₁	Y ₂	S ₂	T ₃	U ₄	V ₅	H ₃	W _D ⊕ S _D
4	X ₀	Y ₁	S ₁	T ₂	U ₃	V ₄	W ₅	H ₄	X _D ⊕ S _D
5	Y ₀	S ₀	T ₁	U ₂	V ₃	W ₄	X ₅	H ₅	Y _D ⊕ S _D

There are n diagonal parity values; there are $n-1$ horizontal parity values. The S parity value is stored within each diagonal parity value.

- Example with seven disks ($n = 5$).

$$S = a_{3,1} \oplus a_{2,2} \oplus a_{1,3} \oplus a_{0,4}$$

For each row r , $0 \leq r \leq 3$, the horizontal redundancy is

$$a_{r,5} = a_{r,0} \oplus a_{r,1} \oplus a_{r,2} \oplus a_{r,3} \oplus a_{r,4}.$$

The diagonal redundancies are

$$a_{0,6} = S \oplus a_{0,0} \oplus a_{3,2} \oplus a_{2,3} \oplus a_{1,4}$$

$$a_{1,6} = S \oplus a_{1,0} \oplus a_{0,1} \oplus a_{3,3} \oplus a_{2,4}$$

$$a_{2,6} = S \oplus a_{2,0} \oplus a_{1,1} \oplus a_{0,2} \oplus a_{3,4}$$

$$a_{3,6} = S \oplus a_{3,0} \oplus a_{2,1} \oplus a_{1,2} \oplus a_{0,3}.$$

- Example with seven disks ($n = 5$).

	0	1	2	3	4	5	6
0	00	00	10	10	01		
1	00	01	10	00	10		
2	10	10	10	10	11		
3	00	11	11	01	11		

client data

$$S = 11 \oplus 10 \oplus 00 \oplus 01 = 00$$

	0	1	2	3	4	5	6
0	00	00	10	10	01	01	
1	00	01	10	00	10	01	
2	10	10	10	10	11	11	
3	00	11	11	01	11	10	

horizontal redundancy

diagonal redundancy

- Reconstruction Calculations

A pair of disk failures is accommodated via four cases:

1. both redundancy disks fail,
2. diagonal redundancy disk and client disk fail,
3. horizontal redundancy disk and a client disk fail, and
4. two client disks fail.

- Reconstruction Calculations

Case 1. Redundancy disks n and $n+1$ must be reconstructed. Use both the horizontal and diagonal redundancy equations as was done initially.

Case 2. Client disk is reconstructed using the horizontal redundancy equations and then the diagonal redundancy disk is reconstructed using the diagonal redundancy equations.

0	T ₀	U ₁	V ₂	W ₃	X ₄	Y ₅	S ₅	H ₀	D ₀
1	U ₀	V ₁	W ₂	X ₃	Y ₄	S ₄	T ₅	H ₁	D ₁
2	V ₀	W ₁	X ₂	Y ₃	S ₃	T ₄	U ₅	H ₂	D ₂
3	W ₀	X ₁	Y ₂	S ₂	T ₃	U ₄	V ₅	H ₃	D ₃
4	X ₀	Y ₁	S ₁	T ₂	U ₃	V ₄	W ₅	H ₄	D ₄
5	Y ₀	S ₀	T ₁	U ₂	V ₃	W ₄	X ₅	H ₅	D ₅

reconstruction of disks 3 & 7

$$S = \bigoplus_{t=0}^{n-2} V_t \oplus D_2$$

$$W_3 = D_3 \oplus S \oplus W_0 \oplus W_1 \oplus W_2 \oplus W_4 \oplus W_5$$

$$H_0 = T_0 \oplus U_1 \oplus V_2 \oplus W_3 \oplus X_4 \oplus Y_5 \oplus S_5$$

additional reconstruction: $X_3, H_1, Y_3, H_2, S_2, H_3, T_2, H_4, U_2, H_5$.

■ Reconstruction Calculations

Case 3. Horizontal redundancy disk n and client data disk c must be reconstructed. Since $a_{n-1,t} = 0$ for all t , select diagonal $r \equiv c-1$. Then $r-c \equiv n-1$ and

$$S = a_{(c-1) \bmod n, n+1} \oplus \left(\bigoplus_{t=0}^{n-1} a_{(c-1-t) \bmod n, t} \right).$$

this equation calculates S for $0 \leq c \leq n-1$. The other diagonals each contribute one client data value. Replace r with $k+c$ to obtain for $0 \leq k \leq n-2$

$$a_{k,c} = S \oplus a_{(k+c) \bmod n, n+1} \oplus \left(\bigoplus_{\substack{t=0 \\ t \neq c}}^{n-1} a_{(k+c-t) \bmod n, t} \right).$$

The horizontal redundancy values $a_{r,m}$ can be calculated using the horizontal redundancy equation.

0	T ₀	U ₁	V ₂	W ₃	X ₄	Y ₅	S ₅	H ₀	D ₀
1	U ₀	V ₁	W ₂	X ₃	Y ₄	S ₄	T ₅	H ₁	D ₁
2	V ₀	W ₁	X ₂	Y ₃	S ₃	T ₄	U ₅	H ₂	D ₂
3	W ₀	X ₁	Y ₂	S ₂	T ₃	U ₄	V ₅	H ₃	D ₃
4	X ₀	Y ₁	S ₁	T ₂	U ₃	V ₄	W ₅	H ₄	D ₄
5	Y ₀	S ₀	T ₁	U ₂	V ₃	W ₄	X ₅	H ₅	D ₅

reconstruction of disks 2 & 5

$$S = \bigoplus_{t=0}^{n-2} H_t \oplus \bigoplus_{t=0}^{n-2} D_t$$

$$X_2 = D_4 \oplus S \oplus X_0 \oplus X_1 \oplus X_3 \oplus X_4$$

$$T_4 = H_2 \oplus V_0 \oplus W_1 \oplus X_2 \oplus Y_3 \oplus S_3 \oplus U_5$$

additional reconstruction: $T_1, W_4, W_2, S_4, S_1, V_4, V_2, Y_5, Y_2, U_4$.

■ Reconstruction Calculations

Case 4. Client data disks c_0 and c_1 must be reconstructed.

Calculate S , the horizontal S_r^h and diagonal syndromes S_r^d for $0 \leq r \leq n-1$. Recall $a_{n-1,t}$ is 0 for all t .

$$S = \left(\bigoplus_{t=0}^{n-2} a_{t,n} \right) \oplus \left(\bigoplus_{t=0}^{n-2} a_{t,n+1} \right).$$

$$S_r^h = \bigoplus_{\substack{t=0 \\ t \neq c_0, c_1}}^n a_{r,t} \quad S_r^d = S \oplus a_{r,n+1} \bigoplus_{\substack{t=0 \\ t \neq c_0, c_1}}^{n-1} a_{(r-t) \bmod n, t}$$

S_r^h is $a_{r,c_0} \oplus a_{r,c_1}$ S_r^d is $a_{(r-c_0) \bmod n, c_0} \oplus a_{(r-c_1) \bmod n, c_1}$

Reconstruction Calculations

for $(r = (c_1 - c_0 - 1) \bmod n; r \neq n-1;$

$$r = (r + c_1 - c_0) \bmod n \}$$

$$a_{r, c_0} = S_{(r+c_0) \bmod n}^d \oplus a_{(r+c_0-c_1) \bmod n, c_1}$$

$$a_{r, c_1} = S_r^h \oplus a_{r, c_0}$$

}

r assumes $n-1$ distinct values as required since n is prime.

Reconstruction Example

Disks 1 and 3 have failed.

c_1 is 3 and c_0 is 1.

$$S = 00$$

	0	1	2	3	4	5	6
0	00		10		01	01	11
1	00		10		10	01	10
2	10		10		11	11	10
3	00		11		11	10	10

$$\begin{aligned} S_0^h &= 10 & S_0^d &= 00 \oplus 11 \oplus 00 \oplus 11 \oplus 10 = 10 \\ S_1^h &= 01 & S_1^d &= 00 \oplus 10 \oplus 00 \oplus 11 = 01 \\ S_2^h &= 00 & S_2^d &= 00 \oplus 10 \oplus 10 \oplus 10 \oplus 11 = 01 \\ S_3^h &= 10 & S_3^d &= 00 \oplus 10 \oplus 00 \oplus 10 = 00 \\ S_4^h &= 00 & S_4^d &= 00 \oplus 00 \oplus 10 \oplus 01 = 11 \end{aligned}$$

Reconstruction Example continued

Disks 1 and 3 have failed.

$$r=1 \quad a_{1,1} = S_2^d \oplus a_{4,3} = 01 \quad a_{1,3} = S_1^h \oplus a_{1,1} = 00$$

$$r=3 \quad a_{3,1} = S_4^d \oplus a_{1,3} = 11 \quad a_{3,3} = S_3^h \oplus a_{3,1} = 01$$

$$r=0 \quad a_{0,1} = S_1^d \oplus a_{3,3} = 00 \quad a_{0,3} = S_0^h \oplus a_{0,1} = 10$$

$$r=2 \quad a_{3,1} = S_2^d \oplus a_{0,3} = 10 \quad a_{2,3} = S_2^h \oplus a_{2,1} = 10$$

The reconstructed array –

	0	1	2	3	4	5	6
0	00	00	10	10	01	01	11
1	00	01	10	00	10	01	10
2	10	10	10	10	11	11	10
3	00	11	11	01	11	10	10