

STORAGE SYSTEM ARCHITECTURES

Spares and Permutation Development Data Layout

Walt Burkhard

Fall 2005

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.

- 34 -

Spare Space I

- 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.

Spare Space II

- 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.

Spare Space III

	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

failed disk

- spare space raises mean repair rate μ .

$$\text{MTTDL} = \frac{\mu + (2n-1)\lambda}{n(n-1)\lambda^2} \quad \text{RAID 5}$$

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.

Spare Space IV

- 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.

Permutation Development Data Layout VI

- **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

Permutation Development Data Layout V

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



- **Declustering Data Layout Goals**

1. Single failure correcting.
2. Evenly distribute spare and parity stripe units.
3. Reconstruction workload evenly divided over surviving disks.
4. Efficient mapping.
5. Parallel Reads.
6. Large Write Optimization.

Permutation Development Data Layout VII

	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.

Permutation Development Data Layout VIII

	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.

Permutation Development Data Layout X

	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.

Permutation Development Data Layout IX

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.

Permutation Development Data Layout XI

■ 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 ,

$$0 ; w^0, w^g, \dots, w^{(k-1)g} ; w^1, w^{g+1}, \dots, w^{(k-1)g+1} ; \\ \dots ; w^{g-1}, w^{2g-1}, \dots, w^{k \cdot g - 1}$$

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: $n = 10 = 3*3 + 1$; $k = g = 3.$

0 1 2 8 3 5 7 4 6 9

Reconstruction workload signature (for disk 0 failing)

1 3 2 2 2 2 2 3 1

0 1 2 4 3 7 8 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
2	1	1	2	2	1
3	1	1	2	2	2
4	1	1	1	1	4
5	1	1	1	2	3
6	1	2	2	4	6
7	1	2	2	5	?
8	2	2	4	?	?
9	2	2	5	?	?
10	2	3	?	?	?

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.