Appeared in *Proc. of the ACM/IEEE High Performance Computing Architecture Conference* (HPCA'99), Orlando, Florida, January, 1999.

Permutation Development Data Layout (PDDL)

Thomas J.E. Schwarz, S.J.
Jesuit School of Theology
1756 LeRoy Avenue
Berkeley, CA 94709
schwarz@scudc.scu.edu

Jesse Steinberg   Walter A. Burkhard
Gemini Storage Systems Laboratory
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0114
<jsteinbe, burkhard>@cs.ucsd.edu

Abstract

Declustered data organizations in disk arrays (RAIDs) achieve less-intrusive reconstruction of data after a disk failure. We present PDDL, a new data layout for declustered disk arrays. PDDL layouts exist for a large variety of disk array configurations with a distributed spare disk. PDDL declustered disk arrays have excellent run-time performance under light and heavy workloads. PDDL maximizes access parallelism in the most critical circumstances, namely during reconstruction of data on the spare disk. PDDL occurs minimum address translation overhead compared to all other proposed declustering layouts.

1 Introduction

Many applications require high availability and throughput from cost effective storage subsystems. Disk arrays offer higher throughput compared to a single disk since a single, large access is serviced by a number of disks. They store information redundantly so that a single or even multiple disk failure does not leave data inaccessible. By adding a (virtual) spare disk to the ensemble, we increase data availability even more [8]. We propose a novel *data layout,* Permutation Development Data Layout (PDDL), which distributes client data, redundant data and spare blocks throughout the disk array [12]. The layout is implemented without large tables or involved calculations. In addition, its performance is either the best or similar to that of the best of other proposed layouts. In contrast to other layouts, it explicitly provides a virtual spare disk.

2 Disk Array Declustering

Disks arrays store data redundantly. We group a fixed number of consecutive disk blocks into a *stripe unit*. A *stripe* (a.k.a. reliability stripe, reliability group, parity group, or cluster) contains a fixed number of stripe units; in a stripe, all stripe units but one store client data. The remaining stripe unit, the *parity unit,* stores the bit-wise parity of the other stripe units. If we cannot access a stripe unit in a stripe, then we access all the others in the stripe and recalculate the data on the inaccessible stripe unit. For this reason, we position the stripe units of a stripe on different disks. In a RAID Level 5 [11], a stripe contains stripe units on all the disks. If a disk fails, all the other disks must process additional accesses for every read from the failed disk. The surviving disks then carry not only their own load, but also the read load of the failed disk.

In a *declustered* disk array [1, 2, 6, 7, 10], the number of stripe units per stripe is smaller than the total number of disks. Any usable data layout (the assignment of stripe units to stripes) fulfills a number of fairly natural criteria: (1) Each stripe contains the same number k of stripe units. (2) Each disk contains the same number b of stripe units. (3) Single Failure Correction: No two stripe units in a stripe reside on the same disk; otherwise, a single disk failure can cause data loss. (4) Even Reconstruction Load: For any two disks, the number of stripes λ with stripe units on these two disks is independent of the two disks chosen; the workload increase after a disk failure is proportional to λ and evenly distributed accordingly. Properties (1) to (4) characterize a *balanced block design* [5]. We need to impose certain other conditions in order to use balanced block designs for disk array layouts: First, all writes to a single stripe unit also update the parity unit. In order to distribute the write load evenly we demand, that (5) all disks carry the same number of parity units. We frequently insert a virtual spare disk (distributed sparing, [8]) into the disk array. We accomplish this by reserving a set of stripe units, called *spare units*, which will be only used after a disk failure. After a failure, we reconstruct the stripe units including the parity units of the failed disk and place them in this spare space. If no disk has failed, then the spare space on a disk is never accessed, so that the utilization of the disk decreases correspondingly.

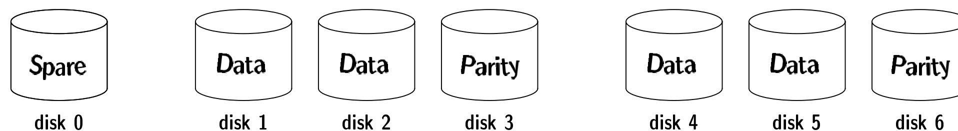

Figure 1. PDDL: Two Virtual RAID Level 4 Stripes with One Virtual Spare Disk

We demand that this decrease be distributed uniformly or, equivalently, that (6) each disk contains the same number of spare units.

Run-time among different data layouts varies considerably. Since disk arrays derive their performance benefits from the increased throughput, an obvious design goal is to maximize read and write throughput. However, not only are read and write operations different since writes need to update parity, but also the operation conditions of the disk array differ depending on whether we work in the fault free mode, whether we have to reconstruct data and place it in the spare space just after a disk failure, or whether we use spare space after the blocks on a failed disk have been recovered.

The data layout mediates between the file system view of storage, typically a very large virtual disk organized as a linear address space of stripe units, and the actual placement of these stripe units on the various disks in the disk array.

PDDL uses almost no storage or calculation to determine its mapping. Recent declustering schemes PRIME of Alvarez, Burkhard, Stockmeyer and Cristian [2] and DATUM of Alvarez, Burkhard, and Cristian [1] also use straightforward, efficient calculation to map stripe units to disks thereby avoiding table lookup. Previous schemes include *Parity Declustering* of Holland and Gibson [6] which uses table lookup to specify balanced incomplete block design together with parity rotation. The *Random Permutation* scheme of Merchant and Yu [9] uses pseudo-random permutations for data layout.

3 PDDL Data Layout

The PDDL approach assumes there are n disks logically partitioned into g virtual RAID Level 4 stripes each containing k stripe units. Figure 1 present the configuration for $n = 7$ disks with $g = 2$ RAID Level 4 stripes, each consisting of $k = 3$ disks. Each stripe consists of dedicated data disks and a dedicated parity disk. We assume that stripe units are accessed via disk logical block addresses (LBA). PDDL numbers the virtual disks of the virtual RAID from 0 to $n - 1$. To address a virtual stripe unit , we need to have a virtual disk number d_v and a virtual LBA b_v. Since we have n physical disks, we number them similarly. We address a physical stripe unit in the same fashion by a physical disk number d_v and a LBA b_v. PDDL provides a one-to-one correspondence between physical and virtual stripe units. This mapping fulfills our seven design goals for a data layout.

The PDDL mapping does not change the LBA. Thus, a physical and a virtual stripe unit address under PDDL have only different physical and virtual disk numbers. PDDL uses a *base permutation* as an initial mapping for disk addresses with LBA 0. For all other LBAs, PDDL adds the LBA to the disk address obtained from the base permutation as shown in Figure 2. This code implements the PDDL mapping for the disk array of Figure 1; the mapping is typical and exemplifies the run-time efficiency possible. The mapping is cyclic and repeats after n LBAs.

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

int virtual2physical( int disk , int stripeUnit )
{
   return ( ( permutation[disk] + stripeUnit ) % 7 ) ;
}
```

Figure 2. PDDL Address Translation

Suppose we need to access stripe unit 10 of the virtual spare disk 0. We would calculate $(0 + 10)\%7 = 3$ and access stripe unit 10 on disk 3. Suppose we must write to stripe unit 20 on disk 4 of the virtual RAID. A disk array write also updates the parity unit, which is located in our case on disk 6 of the virtual RAID. We first apply the base permutation to the disk numbers 4 and 6 and obtain 3 and 5 respectively. We calculate then $(3 + 20)\%7 = 2$ and $(5 + 20)\%7 = 4$. Thus, we access the data of stripe unit 20 of disk 2 and the parity unit 20 on disk 4.

For each virtual LBA there is exactly one spare unit. Accordingly, every disk has the same number of spare units — $1/n^{th}$ of its stripe units. Similarly, there will be the same number of parity units on each disk. This gives us criteria (1), (2), (3), (5), and (6).

Criteria (4) is not true for every base permutation, and finding suitable base permutations is the challenge to utilize the PDDL approach. Sometimes, the best we can do is to use two (or more) base permutations, where the data layout is generated by rotating through the base permutations [12]. In all cases, the PDDL address translation calculation is efficient and the storage requirements minimal.

Number of Stripes	Stripe Width					
	5	6	7	8	9	10
1	1	1	1	1	1	1
2	1	1	2	1	1	?
3	1	1	1'	2	2	1
4	1	1	1	1'	1	1
5	1	1	1'	1	3	2
6	1	1	1	3	6	1
7	1	1	2	5	?	1
8	1	2	4	?	1	?
9	2	2	5	1	?	?
10	1	1	1	?	?	1

Table 1. PDDL data layouts; an apostrophe denotes a solution for a non-prime number of disks obtained or compiled by Furino [5]

4 Existence of PDDL Base Permutations

The PDDL data layouts implement near resolvable incomplete block designs [5]. The way we generate layouts is known as "permutation development." Bose [3] in 1939 gave a general method to create base permutations when n is a prime number. For n composite, the literature [5] contains some base permutations. We have extended this search to include configurations composed of a small number of base permutations. Our results are summarized in Table 1. Each row is for a particular g and every column for a particular k; the entry designates the number of base permutations generating a PDDL data layout. The question mark entries indicate that we do not have a suitable set of base permutations for this configuration.

If the number of disks n is not a prime number but a prime power, then we can created PDDL layouts using calculations within finite fields. This is very attractive for powers of 2 (e.g. for 32 or 64 disks) since we can implement the virtual to physical address translation using only a few XOR and shift operations. Indeed, we believe this is among the fastest address translation schemes possible, if not the fastest. For other prime powers, the resulting implementation is more complicated, but still very fast.

5 Extensions of PDDL

PDDL has the drawback that the number of disks n and the stripe width k have to satisfy $n = 1 \pmod{k}$. We can combine DATUM with PDDL to extend the PDDL's useful domain in a technique referred to as DATUM *wrapping*. Assume that n is slightly larger than $n_0 = gk + 1$. We then use the DATUM data layout with n disks and stripe width n_0. Since n_0 and n are very close, the DATUM mapping is very efficient and remains very small even for large n. We number the DATUM stripes units and treat these numbers as virtual disk numbers in a virtual array with n_0 disks. We then arrange data according to PDDL within this virtual array. In effect, the physical disks are the physical disks of the DATUM layout, the stripes units within DATUM replace the physical disks of PDDL, and the virtual disks of PDDL are the interface abstraction.

We assumed tacitly that there is only one *check* (parity) disk in PDDL. However, for high reliability environments, this does not provide sufficient redundancy. PDDL as well as DATUM, PRIME, Parity Declustering, and Pseudo-Random layouts can be extended to include the extra redundant data.

6 Performance Evaluation

We have tested PDDL as well as DATUM, PRIME, Parity Declustering and RAID Level 5. We used RAIDframe [4] simulating a small disk array with 13 HP 2247 (1.03GB) disks; PDDL can be used for large ensembles of disks as we see in Table 1 but DATUM becomes less efficient for larger ensembles. The other schemes accommodate large ensembles. We tested the data layouts with synthetic workloads consisting of random accesses to a fixed number of contiguous disk blocks. PDDL's response times were at least comparable and often better than all others except DATUM operating at very heavy workloads.

As a result of our experiments, we believe that the intuitively attractive goal of maximum parallelism (one of the design goals for a good declustering scheme in [6]) needs to be modified. While a single large data access benefits from being serviced by as many disks as possible under light workloads, as the workload increases the completion of a single job can be actually faster if it is partitioned over fewer disks. This phenomena has been mentioned recently by Triantifillou and Faloutsos [13]. We introduce the notion of the *footprint,* or, disk working-set, the number of disks accessed by accessing a contiguous set of stripe units in the disk array. The size of the footprint should increase only gradually as the number of contiguous stripe units within the query increases.

7 Comparisons

Distributed sparing dramatically increases reliability and performance [8]. Any of the previous schemes can be modified to include distributed sparing. However, only PDDL distributes the spare space over all n disks of the ensemble thereby incurring the smallest loss of client data space penalty. The other declustering schemes require modification; the most obvious approaches require roughly g times

as much spare space. PDDL uses almost no extra space within its mapping implementation. As Table 1 shows, PDDL schemes exist for a wide variety of disk array parameters. Using DATUM wrapping, we can find PDDL schemes for most commercially interesting dimensions, namely with stripe width k up to 10 and number of disks n up to about 60.

8 Conclusion

PDDL is a new disk array declustering data layout. PDDL layouts exists for a large number of commercially interesting disk array configurations. PDDL uses very little storage and shows minute overhead as it calculates the data layout. The run-time performance of PDDL is very competitive with all other good declustering schemes except for very heavy workloads when DATUM is the clear winner. However, we note that DATUM is most efficient when used with small ensembles. For situations were light to moderate workloads are expected for small to large disk ensembles, PDDL presents an excellent declustering choice.

References

[1] G.A. Alvarez, W.A. Burkhard, F. Cristian: "Tolerating Multiple Failures in RAID Architectures with Optimal Storage and Uniform Declustering", *Proceedings of the 24th Annual ACM/IEEE International Symposium on Computer Architecture,* pp. 62-72, 1997.

[2] G.A. Alvarez, W.A. Burkhard, L.J. Stockmeyer, F. Cristian: "Declustered Disk Array Architectures with Optimal and Near-Optimal Parallelism", *Proceedings of the 25th Annual ACM/IEEE International Symposium on Computer Architecture,* pp. 109-120, 1998.

[3] R.E. Bose: "On the Construction of Balanced Incomplete Block Designs", *Annals of Eugenics,* vol. 9, pp. 353-399, 1939.

[4] W. Courtright, G. Gibson, M. Holland, J. Zelenka: "A Structured Approach to Redundant Disk Array Implementation", *Proceedings of the International Symposium on Performance and Dependability,* pp. 11-20, 1996.

[5] S. Furino, Y. Miao, J. Yin: **Frames and Resolvable Designs: Uses, Constructions, and Existence,** CRC Press, Boca Raton, 1996.

[6] M. Holland, G.A. Gibson: "Parity Declustering for Continuous Operation in Redundant Disk Arrays", *Proceedings, ASPLOS-V,* pp. 23-35, Sept. 1992.

[7] M. Holland, G.A. Gibson, D.P. Sieworuk: "Architectures and Algorithms for On-Line Failure Recovery in Redundant Disk Arrays", *Journal of Parallel and Distributed Databases,* 2, 1994.

[8] J. Menon, D. Mattson: "Distributed Sparing in Disk Arrays", *Proceedings of the COMPCON Conference 1992*, San Francisco, pp. 410-416, 1992.

[9] A. Merchant, P.S. Yu: "Analytic Modelling of Clustered RAID with Mapping Based on Nearly Random Permutation", *IEEE Transactions on Computers,* vol. 45, no. 3, March 1996.

[10] R. Muntz, J. Lui: "Performance Analysis of Disk Arrays under Failure", *Proceedings of the Conference on Very Large Data Bases,* pp. 162-173, 1990.

[11] D.A. Patterson, G.A. Gibson, R.H. Katz: "A Case for Redundant Arrays of Inexpensive Disks (RAID)", *Proceedings, ACM SIGMOD Conference,* pp. 109-116, July 1988.

[12] T.J.E. Schwarz, W.A. Burkhard, J. Steinberg: "Permutation Development Data Layout (PDDL) Disk Array Declustering," *UCSD technical report CS98-584,* April 1998. `www.ucsd.edu/~groups/gemini`

[13] P. Triantafillou, C. Faloutsos: "Overlay Striping and Optimal Parallel I/O in Modern Applications," `http://www.ced.tuc.gr/Research/Reports/HERMES/Reports/htm`, #TR8. 1997.