SUN BLUEPRINTS

[illegible]

APPROVED

Wide Thin Disk Striping

By Bob Larson - Strategic Applications Engineering

Sun BluePrints™ OnLine - October 2000

Sun microsystems

ENTERPRISE ENGINEERING

http://www.sun.com/blueprints

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303 USA
650 960-1300 fax 650 969-9131

Part No.: 806-6201-10
Revision 01, October 2000

Copyright 2000 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California 94303 U.S.A. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, Sun BluePrints, and Solaris are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. and other countries.

RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227-14(g)(2)(6/87) and FAR 52.227-19(6/87), or DFAR 252.227-7015(b)(6/95) and DFAR 227.7202-3(a).

DOCUMENTATION IS PROVIDED “AS IS” AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 2000 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, Californie 94303 Etats-Unis. Tous droits réservés.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l’utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l’autorisation préalable et écrite de Sun et de ses bailleurs de licence, s’il y en a. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées des systèmes Berkeley BSD licenciés par l’Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d’autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, Sun BluePrints, et Solaris sont des marques de fabrique ou des marques déposées, ou marques de service, de Sun Microsystems, Inc. aux Etats-Unis et dans d’autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d’autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L’interface d’utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d’utilisation visuelle ou graphique pour l’industrie de l’informatique. Sun détient une licence non exclusive de Xerox sur l’interface d’utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l’interface d’utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

CETTE PUBLICATION EST FOURNIE "EN L’ETAT" ET AUCUNE GARANTIE, EXPRESSE OU IMPLICITE, N’EST ACCORDEE, Y COMPRIS DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L’APTITUDE DE LA PUBLICATION A REPONDRE A UNE UTILISATION PARTICULIERE, OU LE FAIT QU’ELLE NE SOIT PAS CONTREFAISANTE DE PRODUIT DE TIERS. CE DENI DE GARANTIE NE S’APPLIQUERAIT PAS, DANS LA MESURE OU IL SERAIT TENU JURIDIQUEMENT NUL ET NON AVENU.

Please
Recycle

Adobe PostScript

Wide Thin Disk Striping

The technique of using stripes to spread data and indexes over many disks is described. This disk layout strategy simplifies performance considerations while achieving reliable and manageable disk farms on large systems. The technique is compared to a carefully hand-balanced layout for disk contention and scalability. Hardware mirroring in conjunction with host level mirroring or a volume manager capable of creating striped mirrors is a key enabler. Detailed considerations that database specialists need to make in order to justify this layout technique are presented. The recommendation to use wide-thin stripes to maximize operational flexibility while minimizing complexity is justified.

Introduction

The unpredictable nature of modern datacenter applications makes it difficult to achieve optimal database layout by putting particular tables on dedicated disks. All too often a bottleneck will occur because Benchmarks sometimes obscure this problem, since so much time can be devoted to fine-tuning disk layout to a particular set of transactions or queries. Sometimes architectures can minimize the inter-node traffic by using carefully tuned disk layout. While the benchmarks may show nice performance, these data layout schemes will have problems dealing with arbitrary join criteria. Data layout for benchmarks may become so specialized that even when running the same query, but with different selection criteria, the IO sub-system becomes extremely unbalanced.

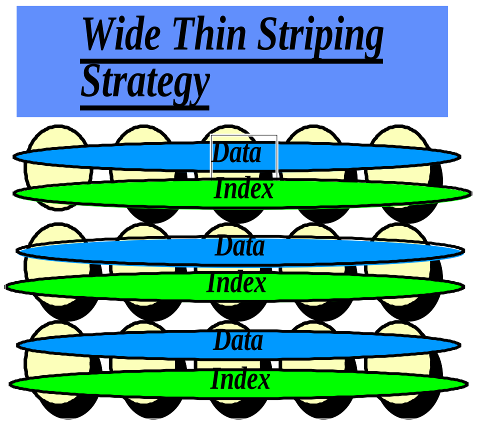
Making use of Pareto's principle is vital to computer system tuning. Pareto is the economist who observed in the 19th century that 80% of production volume usually comes from 20% of the producers. This has been translated into the well known 80/20 rule. For disk layout it predicts that 80% of the accesses are confined to 20% of the data. It provides the basis for the efficiencies gained by using caches. Quite a lot of effort in physical database design goes into arranging data to avoid hot spots. The degree to which the data is localized will be referred to as skew. A pair of numbers attached to skew represents the Pareto ratio.

Another problem encountered with the classic approach is how it complicates the implementation of layered software design. Decisions cannot be made at the application layer without considering the impact on physical layout. Changes in logic (SQL), indexes or tables in the database cannot be made without considering the impact on physical layout. For example, a business directed change in a transaction may change the optimal number of disks devoted to that table.

Depending on the IO channel layout, you may find other factors that should influence layout. In an SMP system, all the CPUs have equally fast access paths to the disks. In this case it is not necessary to match database objects to CPUs through layout. All data-fragments have symmetric paths to each instruction stream.

A challenge for all architectures, SMP, MPP/NUMA, and shared nothing, is delivering good performance when running several queries if they are all going after a small subset of the database. Without wide thin striping, often that data is on a small fraction of the total available disk subsystem. This is actually the normal case predicted by the 80/20 rule

By using wide thin stripes, each major database object (large index or table) is striped over so many disks that hot spotting is impossible. Part of every table and part of every index is on every disk. So no matter which table gets hot, all the disks will be used. This eliminates IO bottlenecks.



Doesn't this increase overhead and introduce head contention by causing accesses to two different portions of the same disk? Often database layout has used this issue as its primary determinate for layout. The thinking is that the database will issue index and data requests at the same time for a particular transaction. But on a larger server, many transactions will be running concurrently anyway. Concurrent access is unavoidable, and should be the primary access pattern for which to prepare. The overall IO performance has many contributing factors that are hard to compare directly. The relative importance is always dependent on the particular application environment. In systems with large CPU counts, the highest priority should be placed on giving the system the capability to support a large numbers of concurrent accesses. Wide thin striping does this by spreading data over many disks and controllers.

The technique of using wide-thin stripes is to make the stripes across all the disk controllers, or at least a large number of controllers. This results in the possibility that the system will bottleneck on the maximum channel rate. This is not a problem; it is the solution when you put it in perspective. The channels (disk controller and FCAL loop) are more expensive than disks, especially considering the precious real estate they use up in server-slots. So getting maximum utilization from the channels is a desirable goal, and its actually quite easy.

The database layout practice of keeping data and index separate is still useful, but using it for a first-order layout rule is a mistake. It is too hard to determine the ratio of IOs going to the data versus the index. Even if you have a system already running, when you make the decision to go to a new machine, you can not be sure the new system will have those same ratios. Suppose the data in a table grows by 8x. This does not necessarily mean the index grows by 8x. The increase in IOs on the index will not scale linearly with the data. The likelihood of changes to columns or addition of indexes makes the inflexibility of classic data layout more untenable.

The Hand-Tuned Disk Layout

If all data had a frequency of access proportional to its size, data distribution would never be an issue. The size of the database object, however, does not drive the frequency of access. Because of the 80/20 rule, the big tables do not necessarily need the most performance. A big history table may have very light IO demands. Such a table can share the same disks with a table that uses little space, but has heavy IO demand.

If one plans ahead, and leaves some spares, the normal monitor and maintenance functions can include fine tuning by splitting tables off a "hot" disk onto two disks. This process becomes too time consuming with large databases that change over time. As the monthly data becomes history, and new data is loaded in, the distribution will have to be adjusted. The query mix may change, leading to some indexes or tables being accessed more heavily than before. The task of database layout and monitoring for IO contention can become a full-time job.

The Wide-Thin Stripe Concept

The crucial idea is to make logical entities at the database level, such as large tablespaces, spread out over large disk sets. As an extreme one could take every disk in the system, and stripe each table over every disk. In practice, it is more practical to break up "all-the-disks" into a few pools (or Veritas disk groups). The idea is to look at the pool like a very large, striped, "virtual-disk". These "virtual-disks" are used to reduce cylinder contention from the high-level contention sets: data and index, and different database instances.

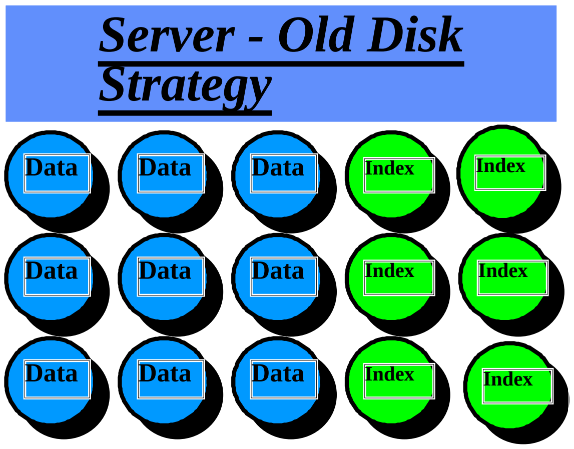
The "virtual-disks" are in turn made up of large numbers of disks that are sliced into small sub-disks and put together into a stripe that spans all the disks in the pool. Each real disk in the virtual-disk is sliced in exactly the same way. This minimizes problems with space management

By pushing the striping interlace to high values, the system can deliver smooth scaling of performance over a range of mixes of small random IO together with large sequential IO.

Cylinder Contention

A quick look at the recent history of disks makes it obvious why we are at a time of changing data layout priorities. The wide-stripe came from a time when seek time was higher than the rotational latency. Typical values for early SCSI disks were 3600 RPM or 16 ms of rotational delay and an average seek of around 30 ms. So total average latency of around 46 milliseconds. The seek portion of total latency is 65% of the total latency.

Modern SCSI disks have rotation rates of 10000 RPM or 6 ms of rotational delay, and average seek times of around 6 milliseconds. The seek portion of total latency is down to 50%. As a result, seek latency has become less important compared to rotational latency.



The early drives did not have track buffer transfers, and thus the average rotational latency was halved, on the assumption that when the IO got to the disk, the sector would be at a random rotational distance from the head. For disks like this, seek time is the dominate component of overall latency, and thus database designers and systems administrators try to minimize "cylinder pinging" by not using lots of small partitions on a drive. Modern SCSI disks have full track buffers and may not allow the read to start until the full track has been read, leading to an effective doubling of the rotational latency. This is considered worthwhile, since many applications will need blocks later in that track on subsequent requests. The drive no longer reads just the requested blocks. It takes the entire track into its track buffer, and then transfers the requested block from the track buffer out through its interface.

Modern SCSI devices are quite different from the disks used at the time that most of the software we run was developed. In the old days, the drives could be made to run faster by paying attention to the geometry of the device. You will see references to these legacy devices scattered throughout RDBMS and OS utilities. The format.dat entries and detailed data about track and heads are an example of this in the operating system. The wide stripes are an example of this in the area of database and stripe layout.

Contention makes the drives operate inefficiently. As an example, consider two tables that are both on one drive. If the access pattern is to get all of one table first, then the other table, the disk only does one long-distance seek and spends most of its time actually transferring data. If both accesses are concurrent, the disk is moving back and forth from one table to the other. But it is a matter of degree, if the disk can spend most of its time transferring data, it is operating efficiently, regardless of whether it is making long seeks or not.

Modern high-performance drives use ZBR (Zone-Based-Recording) to pack more bytes into the longer outer cylinders. This causes the sector to cylinder/track mapping to be a complex function. Sector slipping and track sparing add to the fuzziness of any attempts to accurately map a sector to a spot on the disk. Any perceived disadvantages of this may be overcome by using the track-buffer as a speed matching cache. A simplified view of the device at the system level is preferred.

For most disks, a read of a block not already kept in the track buffer will initiate a read of the whole track. This is a worthwhile readahead since the transfer rate is so much higher than its latency. This way requests for nearby sectors "hit" in the track buffer cache. The track buffer has the benefit of making the details of stripe alignment unnecessary when tuning sequential accesses.

Random versus Sequential IO

If the time for a set of blocks to be transferred is much larger than the time for access, then it is considered sequential IO. Typical access times are around 10 milliseconds. So if the transfers take 100 milliseconds or more, the IO could be considered sequential. Typically, the drive transfers a full track in one revolution, or about 50k Bytes in 8 milliseconds which means 600k Bytes in 100 milliseconds. One model for smooth interplay is to make the RDBMS run full table scans with chunksizes of more than 500 KB chunks, and run the indexed (random) access as regular small block IO.

Optimal Access Patterns on Narrow versus Wide-Thin Layout

Certain access patterns can cause problems for hand-tuned layouts. For example a sequential read, such as a full-table-scan, may happen concurrently with indexed lookup on the same table. The impact of this contention breaks up the efficient sequential IOs of the full-table-scan by constantly moving the heads around to pick up blocks for the indexed lookup. Two concurrent full-table-scans started at slightly different times leads to the same effect. At this point the disk has dramatically lower throughput. The case of multiple concurrent full-table-scans has been tested on a large disk configuration with results shown in the following experiment.

Empirical Comparison of the Two Striping Methods

It is difficult to design an experiment to compare the alternative methods without inherent bias. Increasing the complexity and randomness of the experiment will benefit the wide thin stripe method while increasing regularity and uniformity will benefit the narrow layout. The following results should be part way between, designed to highlight the strengths of each method.

For this experiment, starting with the application level design there are 5 volumes to be laid out on 18 disks. The wide layout has 3 6-way and 2 9-way volumes. The narrow-stripe method leads to 3 4-way and 2 3-way volumes.

The IOs dispatched to these volumes are made with a number of concurrent sequential accesses directed at the devices in the manner of database table scans using the bio benchmark program. The blocksize of the IO is 2048 bytes. Other blocksizes show similar results with a slightly lower IOs/second as blocksize increases.

Table 1 highlights the advantage of wide thin stripes in the case of a highly skewed data distribution. Here, just one of the five volumes has any access at all. So 100% of the accesses are directed at 20% of the data volume. In this case the reads are uniform random over that 2 gigabyte volume. In the "wide" case of the, all 18 disks are busy, but only at 70%, while the "narrow" case has 3 disks, but at 95% busy.

Table 1: High Skew IO/s

Conc	***Wide***	***Narrow***	***diff***
High 8	973	603	61.0%
Med 4	594	423	40.0%
Low 2	334	272	22.0%

The lower busy can be transformed into the capability to support higher levels of concurrent activity. Table 2 highlights the advantage of wider stripe volumes over narrow in this situation. The high concurrency case consists of 8 threads working on one volume (40 total). Low concurrency is using 2 threads per volume, for a total of 10 IO streams. All volumes are the same size, 4G, and use the default stripe interlace of 64K bytes.

Table 2: High Concurrency IO/s

Skew	***Wide***	***Narrow***
Uniform	2925	2828
Light 60/40	2936	2639
Med. 80/20	2341	1766
High 90/10	1924	1344

Table 3: Low Concurrency IO/s

Skew	***Wide***	***Narrow***
Uniform	1267	1270
Light 60/40	1278	1233
Med. 80/20	1154	946
High 90/10	1010	871

It is interesting to note that the low concurrency uniform case is the only place where narrow striping has an advantage. This is the simplest model for data layout, and the easiest to benchmark. We have to be very careful to sure the scaling of the benchmark matches the real system.

Conclusion

Nobody likes surprises in the performance characteristics of a system. Surprises make systems analysts work overtime, and users wait for results. By using large IOs and striping over many disks the performance of the system has been smoothed out. The peaks are not as high, but this is more than made up for because the valleys are not as low. The performance characteristics are more predictable, and therefore easier to get right.

Anticipate the randomizing effects of ad-hoc queries and evolving transaction mixes. Design with a wide-thin striping strategy. The resulting disk performance will be greater than traditional approaches in if not all cases.

Author's Bio: Bob Larson

Bob joined Sun along with the others in the Cray acquisition on July 1, 1996. Bob has worked on high-end world record setting database benchmarks since that time. Bob has specialized in database performance tuning, Low-level disk and network benchmarking, UNIX® systems administration, Kernel process Scheduling and Disk performance and layout. Bob holds a Bachelor's degree in Physics from the University of Chicago.