IBM® DB2® for Linux®, UNIX®, and Windows®

Best practices

Database storage

Aamer Sachedina
Senior Technical Staff Member
DB2 Technology Development

Matthew Huras
DB2 for Linux, UNIX, and Windows
Kernel Chief Architect

Robin Grosman
Manager, DB2 Buffer Pool Services team

Issued: January 2012

Executive summary

In a world with networked and highly virtualized storage, database storage design can seem like a dauntingly complex task for a DBA or system architect to get right.

Poor database storage design can have a significant negative impact on a database server. CPUs are so much faster than physical disks that it is not uncommon to find poorly performing database servers that are significantly I/O bound and underperforming by many times their potential.

The good news is that it is not necessary to get database storage design perfectly right. Understanding the innards of the storage stack and manually tuning the location of database tables and indexes on particular parts of different physical disks is neither generally achievable nor generally maintainable by the average DBA in today's virtualized storage world.

Simplicity is the key to ensuring good database storage design. The basics involve ensuring an adequate number of physical disks to keep the system from becoming I/O bound.

This document provides basic initial suggestions for a healthy database server through easy-to-follow best practices in database storage, including guidelines and recommendations in each of the following areas:

- Physical disks and logical unit numbers (LUNs)
- Stripe and striping
- Transaction logs and data
- File systems versus raw devices
- Redundant Array of Independent Disks (RAID) devices
- Registry variable and configuration parameter settings
- Automatic storage

Note: This paper focuses on best practices for deployments of DB2 for Linux, UNIX, and Windows in typical OLTP environments. Unless specifically mentioned, the recommendations in this paper do not necessarily apply in data warehousing environments or in environments where a DB2 database is used as the underlying database for third-party software.

Introduction to database storage

Storage area networks (SANs) and network-attached storage (NAS) have fundamentally changed the database storage world. A decade or so ago, the word *disk* referred to physical disks with heads and platters. In today's storage world, a disk is often a completely virtual entity that is somewhere on the storage network and that can be a single physical disk, a portion of a physical disk, a RAID array, a part of a RAID array, or part of several RAID arrays. Recent enhancements in file system options, such as direct and concurrent I/O, have almost eliminated any performance advantage that using raw devices had over using file systems.

Although Moore's Law has held for CPU processing power, it does not apply to disk drives. Despite changes in storage communication that were introduced by SAN and NAS, the underlying infrastructure for storing bits remains fundamentally the same. A mechanical spindle rotates a platter of magnetic material, upon which bits of information are encoded using a read/write head on the end of an actuator. Although spindle speeds have increased and caching of data on storage controllers using dynamic random access memory (DRAM) and non-volatile random access memory (NVRAM) has helped, neither of these advancements have kept up with the sheer magnitude of processing power increases over the past decade or so. The result is that relative to CPU processing speeds, disk I/O service times have become much slower. This difference requires an increasingly larger number of physical disks per CPU core to ensure that the system is not I/O bound. Because disk capacities have increased substantially, achieving an appropriate ratio of physical disks per CPU core is often overlooked, therefore leading to disk bottlenecks.

Given the changes in storage, file systems, and CPU processing speeds, the best practices for database storage provisioning and management have also evolved. In the past, a DBA might have been advised to determine which physical disk and which part of each physical disk on which to place individual tables and indexes. In today's virtualized storage world, for the average DBA, yesterday's best practices are impractical.

The best practices presented in this document have been developed with the reality of today's storage systems in mind. For related information about database performance and speed of database operations, refer to the "Best Practices for Physical Database Design" paper. This paper and others are available at the DB2 Best Practices website at http://www.ibm.com/developerworks/data/bestpractices/db2luw/.

Goals of good database storage design

Good database storage design has the following characteristics:

- Predictable I/O and system performance
- Uniform use of available I/O bandwidth and capacities, to avoid physical disk "hot spots"
- Ease of ongoing management, for example, adding new storage
- Ease of problem determination
- High availability through redundancy

The best practices that are presented in this document present straightforward starting configuration that assist in achieving all the goals of good database storage design.

Simplicity in database storage design

> **"Make everything as simple as possible, but not simpler"**
> **– Albert Einstein**

In designing storage for a database, simplicity is your secret weapon. This simplicity sometimes comes at the cost of not choosing the most optimal I/O characteristics for a specific table or table space**.** Consider the situation in which an experienced DBA with excellent storage skills and an unlimited amount of a storage administrator's time carves out storage for a particularly important table or index from specific portions of physical disks. Assume that at the time that it is devised, this approach provides the best-performing solution. However, maintaining the original design objectives (best performance) nearly always results in a system that is more complex to manage and that has unbalanced physical disk utilization. Problem determination is nearly always difficult. Also, the storage bandwidth that was initially considered sufficient for the particularly important table or index might not be sufficient as time goes on and the application grows.

The objective of good database storage design on a dynamic system is to meet all goals as part of the initial design of the system and to continue to meet those goals over time. The simple best practices that are described in this document achieve these goals with virtually no performance sacrifices.

Recipe for database storage success

Think about real physical disks, not just storage space

Physical disks within a disk subsystem are not usually directly accessible by host systems, such as DB2 database servers, and are not directly visible to DBAs. Storage administrators provision units of storage as logical unit numbers[1] (LUNs), which appear to host systems as bona fide SCSI disks. A LUN, however, is a completely virtual entity that can map to any combination of physical disks. An individual LUN can be a single RAID array, a portion of a RAID array, a single physical disk, a portion of a disk, or a *meta* of multiple RAID arrays.

Although the storage world has become more virtualized, data is still normally stored on mechanical disk drives. Regardless of which vendor's storage subsystem you use, the data is located on mechanical disk drives, that is, on rotating physical disk platters. The input/output operations per second (IOPS) bandwidth of storage that you can achieve from a LUN is directly proportional to the number of physical disks that the LUN comprises.

A rule of thumb is that a 15,000 RPM Fibre Channel disk can perform about 200 IOPS. However, the reality is that a physical disk, a RAID array, or even a complete disk subsystem can service I/Os with reasonable times up to a point, and then as IOPS increase, I/O service times degrade. I/O service times are critical to performance for typical random I/O applications. The goal is to avoid situations where I/O service times for one set of disks are good (for example, averaging 10 ms for reads), while for other disks, service times are significantly poorer, for example, averaging 20 ms. This situation would be a disk hot spot and bottleneck, which degrades application performance.

Although we still assume that a typical mechanical disk is similar to the one mentioned above,Solid State Drives (SSDs) are increasingly used. SSDs have the advantage of being entirely electronic, without moving parts. As a result, I/O service times are typically less than a millisecond, instead of up to approximately 10 ms, for typical small random reads from physical disks. SSDs have the disadvantage of costing much more per GB than physical disks, and SSDs typically have less space. However, an SSD can perform over 100 times the IOPS of a physical disk (this number varies somewhat among SSDs). As a result, purchasing a mix of SSDs and physical disks can be less expensive than purchasing physical disks alone, because you require fewer disks to achieve the same IOPS.

Although storage controller caches assist with improving storage performance of I/O-intensive systems, DB2 database systems already cache relevant data in their buffer pools. This makes it less likely that storage controller caches can sufficiently reduce the need for physical disks. In an enterprise database system, there is no substitute for physical disks.

[1] In SCSI storage systems, a LUN is a unique identifier that is used on a SCSI bus that enables it to address up to eight separate devices (such as disk drives), each of which is a logical unit.

Because CPU processing speeds have increased substantially relative to spindle speeds, two good rules of thumb are as follows:

- For OLTP, ensure that there are 15 - 20 dedicated physical disks per CPU core.
- For warehousing, ensure that there are 8 - 10 disks per CPU core.

If you are using SSDs, virtualization, or other sharing methods, convert number of disks to IOPS by using 200 IOPS per disk. For example, for an OLTP workload, the converted value is roughly 3000 - 4000 IOPS per CPU core. You might be able to reduce this number by using I/O techniques such as multidimensional clustering (MDC), materialized query tables (MQTs), and good schema management and design.

It is expected that with the next generation of processors, a larger number of physical disks will be required per processor core for I/O-intensive database servers.

Dealing with highly virtualized storage

As was noted in the introduction of this paper, disk storage is more and more often treated as a generic commodity, where the available storage space is often abstracted from the physical devices on which it is located.

If your enterprise I/O infrastructure requires you to use virtualized storage systems, you must ensure that the virtual LUNs are indeed made up of dedicated physical disks. Here's why: an enterprise system quickly becomes I/O bound if there are too few real disks to keep the CPUs busy. As a DBA who is concerned with database performance, you probably prefer to communicate your storage needs in terms of the number of real disks. However, storage administrators, like much of the rest of the world, often think of storage needs merely in terms of space. And with platter sizes having increased substantially over the last decade or so, the challenge of making an argument for more physical disks per CPU core rather than just more space has become more difficult.

To make matters even more challenging, if you gain access to the number of physical disks that you need to ensure good performance, you might have to justify having too much space. For example, assume that you have a single CPU core with storage on 20 physical disks. This ratio of disks to CPUs should yield sufficient I/O parallelism to provide great performance. However, if each disk device can store 150 GB, you have approximately 3 TB worth of space per CPU core. If you have multiple CPU cores, each with a 1:20 ratio of CPUs to physical disks, you can see how the total amount of storage can quickly add up. Of course, databases vary, and the ratio of physical disks to CPUs for a balanced configuration can also vary.

Configuring enough spindles often requires that you do not fully use all available space on physical disks, but your storage administrators must be careful not to allocate storage such that I/O service times are longer than reasonable. For example, they might be tempted to allocate some of the unused storage in such a deployment as a separate LUN to other applications or processes over which you, the DBA, have no control. However, too many IOPS from competing applications or processes can cause performance for a particular application to degrade.

Now, you could use the unused space yourself as a staging area for online backups or archive logs for your database before backing them up to long-term storage. This can be a good use of the unused space because you control when you perform backups (when you use the devices). You can perform online backups during periods when peak I/O throughput is not required. If you use the same disks for data and backups to maximize space use, consider the risk of a hardware failure on that device. Ensure that you archive your backups in a timely manner to an external backup target, for example, Tivoli® Storage Manager (TSM).

Further complicating the system is that even with dedicated spindles, there are now more things which can impact performance than just the physical disks. Traditional SAN storage controllers have CPU, memory for cache, and other resources, which are shared by all disks. Additional layers of storage virtualization, for example, the San Volume Controller (SVC), AIX® VIOS, and AIX Logical Volume Manager (LVM), further abstract the physical disk. These forms of virtualization can provide desirable functional enhancements, such as the ability to migrate transparently from one set of LUNs to another or the ability for multiple host LPARs to share a Fibre Channel Host Bus Adapter (HBA). However, they do so at the cost of additional complexity of subsystems in the I/O path. Also, they do not reduce the need for real physical disks for I/O-intensive systems.

Have dedicated LUNs and file systems for each non-partitioned DB2 database server or database partition

It is a best practice to have dedicated LUNs for each non-partitioned DB2 database server and for each database partition in a partitioned database. Each LUN should preferably be backed by a separate and equal set of physical disks. Typically, you create LUNs from RAID arrays comprising the same number or a similar number of disks. Because DB2 processes balance I/Os across containers, this method balances I/Os across physical disks.

Dedicating a LUN to a DB2 server or partition prevents the physical disks of that LUN from being used to create other LUNs that could be used for purposes that are likely to interfere with the primary use of those disks. As mentioned in the previous section, though, you should ensure these LUNs are under your control and use them carefully. Also, as mentioned previously, one use of the excess space is as a staging area for backups and archive logs.

You should also create a single file system on each such LUN and dedicate that file system for use by a single DB2 server or database partition. Dedicated LUNs and dedicated file systems per LUN keep the storage layout simple and can assist with problem determination.

For partitioned database systems, you should follow the IBM Balanced Configuration Warehouse practices.

An example of how dedicated LUNs and file systems can help with problem determination is when poor selection of a partitioning key on a table prevents a query from getting the right amount of parallelism. When LUNs and file systems are dedicated to database partitions, this problem becomes evident if you see that one set of LUNs is

busy for much longer than the other LUNs because one partition has all the data that must be processed.

Stripe in two places at most

Storage controllers offer excellent RAID striping directly in the controller firmware. You should design enterprise systems to use the striping that storage controllers provide. A convenient way to do this is to have the storage controller expose an individual RAID array, for example, RAID-5 or RAID-10, as a single LUN. You can then provide one or more such LUNs to DB2 partitions.

If you provide more than one LUN to a DB2 database server or database partition, use DB2 fine-grained striping between containers.

Because two levels of striping are adequate for all systems, avoid using a third level of striping, such as the operating system's logical volume manager (LVM) striping or striping across RAID arrays, as offered by some disk subsystems. LVM striping is beneficial to other middleware, which, unlike DB2 database systems, cannot do their own striping.

Tradeoffs regarding sharing disks for logs and data

In deciding whether to store logs and data on the same disks, consider database resiliency, efficient use of storage, difficulty of designing the data layout, and the ability to optimize performance or service levels independently. Separating logs and data leads to better resiliency and the ability to optimize performance or service levels for each independently. Storing logs and data on the same disks is simple, by assuring that IOs are evenly balanced across the physical disks. Different customers will have different relative values for these goals.. Different workload types place different relative values on these goals.

Separating logs and data can improve database failure resiliency by making the logs more available in two ways. Firstly, because separated logs are striped over fewer disks, there is less chance that a disk containing logs will fail. Secondly, you can give logs a higher degree of failure protection by using a more resilient raid level (see "Choice of Raid Level," later in this paper).

Ways to maintain high resiliency while sharing disks include using a full remote HADR system to protect against site failures and putting primary logs on half of the disks and mirroring them. Putting primary logs on half the disks and mirroring them to the other half balances the I/Os across the physical disks so that storage is used efficiently. However, because write I/Os to the logs are doubled, you need more physical disks because there are more IOPS.

Resiliency tolerance depends on workload and business priorities. For example, data warehouse configurations have some tolerance for lower resiliency, because you can reload data.

A setup with separate logs and data can also outperform a shared storage setup. Firstly, because I/Os to devices holding logs are sequential on real physical disks, actuator seeks are avoided, I/O service time is reduced, and throughput is higher than with random I/O. (I/O becomes random if you also place data tablesspaceson the same disks). A single physical disk can handle approximately 70 MB per second for sequential I/O but can handle perhaps only 250 random IOPS, and if the I/Os are 4 KB in size, that is less than 1 MB per second. Intelligent disk subsystems mitigate this somewhat by coalescing the writes to the disks containing logs. Secondly, you can control resource and I/O service times for each of logs and data independently. For OLTP, fast log response time is often more important than I/O service times for data, which is frequently asynchronous. Note that on hardware with a protected write cache, unless the cache is very dirty, the response time for writes is determined by the time that it takes to write to the shared cache.

Achieving better performance by separating logs and data requires the storage design for data and logs to match their I/O profiles. If you dedicate too many physical disks to logs, service times for data I/Os will theoretically be slower (how much slower depends on the utilization of the physical disks), resulting in slower transaction times. If you dedicate too many physical disks to the data, you risk filling up the write cache and causing writes to the log to significantly slow down write I/O service times. This can significantly increase application transaction time and reduce throughput.

Another advantage of storing logs and data on the same physical disks is that it typically results in a simpler design because there is no need to predict the ratio of log to data I/O. Regardless of workload or changing workload, I/Os are always evenly balanced on physical disks regardless of the ratio of log I/Os to data I/Os. In a warehouse, these factors normally dominate, making sharing the preferred approach, but OLTP systems can also benefit from this approach.

The best approach depends on your specific requirements. For warehouses, the best practice is for logs and data to share disks, because the resiliency offered by RAID is typically sufficient and the even usage of disks is most important. For OLTP, with shorter transactions, separating logs and data is more appealing. If you separate logs and data, the ratio of log I/Os to data I/Os is heavily dependent on the workload, but a good rule of thumb is to allocate 15 - 24% of the spindles for logs and the rest for data.

Use file systems instead of raw devices, one file system per LUN

Direct I/O and concurrent I/O have almost eliminated the need to use raw logical volumes to improve performance. File systems provide manageability that is superior to that of raw devices because you can use a single file system as a container for multiple table spaces.

Each LUN that you provision for a database partition should have a single separate file system. Therefore, each database partition generally has the following file systems:

- One transaction log file system. You create this by using the LUN that you provisioned for that partition's transaction logs
- More than one data file system. You create each file system by using its own separate LUNs that you provided for table space data.

If you are using files in a file system for containers and you have multiple table spaces whereby each table space uses a file in each data file system, it is important to avoid fragmentation of the files. Fragmentation significantly reduces throughput for operations such as full table scans, because fragmentation prevents data from being read sequentially from disk. Creating multiple files in a file system at the same time results in.fragmentation at the file system level. Therefore, you should create or restore files serially rather than in parallel.

Choice of RAID levels

The amount of contiguous data on each physical disk in a RAID array is known as a *strip.* The amount of data that comprises all the strips in a single array is called a *stripe*.

Typical RAID strip sizes are 32 KB, 64 KB, 128 KB, and so forth. A RAID-5 4+1 array has a stripe size equal to four times the strip size. For example, on a system with a 128 kb strip size that uses RAID-5 4+1, the stripe size is 512 kb (128 kb x 4).

You should set the EXTENTSIZE for table spaces to a number of pages to a multiple of an entire RAID stripe. For example, if the page size is 8 KB, an EXTENTSIZE of 64 (512 KB/8 KB) is appropriate.

There are tradeoffs among the various RAID levels:

- RAID 0 offers no redundancy. Data is lost if any physical disk in the array fails, so this level is appropriate only where data loss is acceptable.
- RAID 1 or 10 provides two complete copies of the data. Each application write results in two physical writes to disk. Reads have no additional overhead, unless a disk in the array fails. Up to half the disks in a RAID 1 or 10 array can fail while availability is maintained. However, if the two "wrong" disks fail in a RAID 10 array, you lose access to your data.
- RAID 5 uses a parity scheme whereby an *N*+1 RAID 5 array has *N* disks worth of protected space. Each application write results in four physical I/Os:
  - Reading the data that you are writing over
  - Reading the associated parity on another disk in the array
  - Calculating the new parity
  - Writing the new data and the new parity

  Sequential writes to RAID 5 benefit from full stripe writes; thus, the overhead, rather than being four times, is 1/*N,* where there are *N*+1 physical disks in the array. One

disk can fail while availability is maintained; thus, RAID 5 arrays with more disks are less reliable than ones with fewer disks. RAID 5 arrays are normally limited to no more than 12 disks, and typically fewer.

- RAID 6 uses a double parity scheme whereby an *N*+2 RAID 6 array has *N* disks of protected space. RAID 6 arrays maintain availability if there are up to two physical disk failures. Random writes require six physical I/Os.

The RAID level that you use affects the number of physical disks that you require. To determine the number of physical disks that you need at a minimum, you must know the peak IOPS and the R/W ratio. It is also useful to have statistics such as the disk subsystem cache hit rate, the average I/O size, and the percent of I/Os that are sequential. With this information, you can design an appropriate disk subsystem with a specific RAID level to meet your performance requirements.

Cost is another factor: RAID 10 costs more per GB than RAID 6, which in turn is more expensive than RAID 5. Also, each RAID level differs in IOPS bandwidths for various R/W ratios and has different availability characteristics. Choose a RAID level that meets the combination of your price, performance, and availability requirements.

Disk subsystems that spread I/Os across physical disks

Some disk subsystems, for example, XIV, automatically spread I/Os evenly across all the physical disks. XIV does this by randomly spreading LUNs across the physical disks. The SAN Volume Controller offers striped VDisk LUNs, which are striped across back-end RAID arrays.

Even though spindles are shared there are still benefits to using multiple LUNs rather than a single LUN, due to disk subsystem read-ahead. If a disk subsystem detects that a LUN is being sequentially read, it reads ahead and places the data in the disk subsystem cache to improve I/O service times and throughput. In this way, sequentially read structures benefit from their own LUNs. For databases, the data, temporary storage, and logs should all have their own LUNs. Another benefit to this is that processes that handle I/Os (for example, the disk driver) can operate in parallel.

Dedicated versus shared storage

There are tradeoffs between dedicated and shared storage.

Dedicated disks are physical disks that are dedicated to DB2 storage and do not contain data for other applications, which are often running on other systems. With dedicated storage, each application is allocated sufficient disks for its peak IOPS workload. Dedicated resources provide maximum performance, but at the cost of utilization.

Typically, shared storage offers better price performance and better utilization because the peak IOPS for each application usually do not occur at the same time. Storage is sized for the peak of the sum of the IOPS for all the applications. By sharing storage, there are typically more spindles available for an application that might need the additional physical disks for a spike in IOPS. The shared storage approach is similar to CPU virtualization, whereby applications share a pool of CPUs.

Some disk subsystems offer the ability to set I/O limits for LUNs or hosts. Using these facilities, you can guarantee a specific IOPS bandwidth to critical applications, thus improving their quality of service.

The downside of shared storage is that I/Os for one application (perhaps a poorly designed application that is overusing resources) can interfere with I/Os for more important production workloads. Applications that have unpredictable and potentially very high IOPS workloads are poor candidates for using shared storage because of their potential to interfere with production workloads and the difficulty in sizing the storage.

As IOPS increase for a disk subsystem, average I/O service times usually decrease, although I/O service times are relatively constant up to a point. As I/Os queue up to access the physical disks or other components of the storage subsystem approach a bottleneck situation, I/O service times degrade. Increases in I/O service time typically result in less application bandwidth for a system, longer application transaction times, and longer batch job run times.

Either dedicated storage or shared storage is acceptable if you size the storage to meet performance needs.

Handling growth

There are two kinds of growth to be concerned about:

- Growth in workload demands (IOPS or MB/s)
- Growth in space needed (GB)

Increasing IOPS or MB/s requires the server to use more physical disks. These extra disks might exist within the storage subsystem, or new disks might be required. With some disk subsystems for file systems, you can dynamically add physical disks to a RAID array and then move data so that it is evenly distributed among the spindles. With DB2 commands, you can add new containers to table spaces or add paths to the database if you are using automatic storage on new LUNs. You can then rebalance the data so that old and new data is evenly spread across the containers, and by extension, across the disks underneath.

It is essential to ensure that IOPS do not decrease as the amount of space that you need grows. For example, if you load new data for the next quarter, most of the new writes will be for the same data range, and most of the queries will also access this same recent data. Ways to deal with this situation include the following ones:

- Move your data around in a way similar to that described in the IOPS growth scenario.
- Add enough disks to enable this new data to be accessed with response times and IOPS that are similar to the previous ones (for example, double the number of disks).
- Use disk subsystem dynamic volume expansion.
- Add a new LUN and then a new stripe set to the DB2 table space.

Increasing storage performance might require you to add bandwidth to the storage in addition to disks within the storage..

Setting the table space extent size in virtualized storage environments

Sometimes it is not possible to know which RAID array the file system for a specific DB2 table space container is stored on. This can happen if there are layers of storage virtualization between the LUN that is provisioned to a database server and the storage controller. In such cases, when you create your table spaces, you should set the EXTENTSIZE option??? of the CREATE TABLESPACE statement to a number that is conducive to the efficient big-block I/O that is performed by *prefetchers*. Prefetchers are I/O servers.

A good rule of thumb is to have an extent size of 256 KB. Extent sizes of 128 KB or 512 KB are also good options. The default setting for EXTENTSIZE (32 pages, as specified by the **`dft_extent_sz`** configuration parameter) should provide adequate performance in most cases. For example, if you use 8 KB pages for the database, an EXTENTSIZE of 32 should provide an adequate extent size (8 KB x 32 pages = 256 KB) when you do not have details regarding RAID strip sizes. Even with a 4 KB or 16 KB page size, an EXTENTSIZE of 32 pages yields an extent size in the recommended range. A 4 KB page size yields an extent size of 128 KB, and a 16 KB page size yields an extent size of 512 KB.

Set the DB2_PARALLEL_IO registry variable for IO parallelism

By default, the prefetchers for DB2 for Linux, UNIX, and Windows perform one extent-sized I/O for each table space container during table scans. EXTENTSIZE is thus not only the DB2 unit of striping; it is also the read I/O size that prefetchers use during sequential prefetch. If you set EXTENTSIZE according to the best practice in the previous section, you do not have to set the **`DB2_PARALLEL_IO`** registry variable to ensure that one extent of data spans all the drives within the single RAID array in the file system that each DB2 container uses. The reason is that the database manager automatically prefetches the extent from all physical disks in a container in parallel.

There are ways to set EXTENTSIZE and to design the striping of a DB2 system other than what is presented in this paper, however.

One alternative that is used in some configurations is to set EXTENTSIZE to span continuous data on a single physical disk within each RAID array. That is, set EXTENTSIZE to be the strip size, rather than the stripe size as recommended in the previous section. In such configurations, a single extent-sized I/O per table space container during sequential prefetch is inadequate because it drives only a single physical disk within the RAID array that the file system is based on. Setting the **DB2_PARALLEL_IO** registry variable is required for such systems if you want the database manager to generate multiple extent-sized prefetch requests to drive all of the physical disks for each DB2 table space container in parallel.

You can use the **DB2_PARALLEL_IO** registry variable to explicitly specify the number of physical disks under each container so as to generate the right number of prefetch requests for each container. For example, if each table space container exists on a file system that is backed by a RAID 5 4+1 array, the following **DB2_PARALLEL_IO** registry variable setting is appropriate:

db2set DB2_PARALLEL_IO=*,5

The asterisk (*) indicates that the setting applies to all table spaces. The 5 indicates that under each container are 5 (4 + 1) individual physical disks that should each be driven using an extent-sized read request by a separate prefetcher.

The **DB2_PARALLEL_IO** registry variable setting is taken into account by the algorithms that compute the following items:

- The prefetch size for each table space, when the PREFETCHSIZE option of the CREATE TABLESPACE or ALTER TABLESPACE statement is set to AUTOMATIC

- The number of prefetchers, when the **num_ioservers** configuration parameter is set to `AUTOMATIC`.

For a related recommendation, see "*Avoid manually tune the num_iocleaners, num_ioservers, and prefetchsize configuration parameters*" on page 18.

FILE SYSTEM CACHING clause on CREATE and ALTER TABLESPACE

The NO FILE SYSTEM CACHING clause was added to the CREATE TABLESPACE and ALTER TABLESPACE statements in DB2 Universal Database, Version 8.2. Since Version 9.5, this setting has been the default for newly created databases for those file systems where direct or concurrent I/O is available, such as JFS2, GPFS™, and VxFS.

The NO FILE SYSTEM CACHING clause enables direct or concurrent I/O, whichever is applicable to the operating system on which the DB2 database system runs. Direct or concurrent I/O effectively allows DB2 I/O operations to have raw device–like performance on file systems. Hence, the general DB2 best practice in this area is to rely on the default of NO FILE SYSTEM CACHING.

There are some specific considerations to note regarding table spaces that contain large objects (LOBs). In the past, systems often benefitted from the use of file system caching (by enabling FILE SYSTEM CACHING at the table space level) for table spaces that stored the LOB and LF portion of a table data, as this class of data is not normally cached in a DB2 bufferpool. Recent releases of DB2 have added LOB inlining support, which allows LOB data to be 'inlined' in a data row, and therefore, to be cached in a DB2 bufferpool. Further, we have noted that it is typically the case that the majority of LOBs that can benefit from being cached tend to be of sizes that fit into a DB2 page (i.e. < 32K). As such, LOB inilining is now the preferred method of caching LOB data.

Use DB2 automatic storage to stripe everything everywhere

DB2 automatic storage technology is a simple and effective way to provision storage for a database. With automatic storage, you provide storage to the database as a whole, rather than to individual table spaces. Automatic storage is used by default when you create a database by using the **`CREATE DATABASE`** command.

Consider the following example:

```
DB2 CREATE DATABASE MYDB ON /data1, /data2, /data3 DBPATH ON
/mydbpath
```

This command creates a database with three storage paths: data1, data2, and data3. Each of these three paths is a separate file system, each created using dedicated LUNs. The default table spaces that are created by the DB2 database system (SYSCATSPACE, USERSPACE1, and TEMPSPACE1) each have three containers, one on each of the three storage paths. The **`DBPATH`** parameter is set to another file system (`/mydbpath`), which was created using the LUN that was provisioned for the DB2 transaction logs. DB2 metadata is also on this file system.

Any new table spaces that you create without explicitly providing containers are also created using a stripe-everything-everywhere approach. For example, consider the following DB2 statement:

```
DB2 CREATE TABLESPACE MYTS
```

The table space MYTS will have three containers, one on each of the storage paths.

Although using automatic storage does not preclude you from defining system-managed space (SMS) or database-managed space (DMS) table spaces, in the same database, automatic storage generally eliminates the need for you to define them.

Because the DB2 storage manager uses a simple stripe-everything-everywhere approach, the best practice with automatic storage is to use storage paths, or file systems, that are uniform in capacity. This ensures that parallelism remains uniform. A specific storage

path cannot fill up prematurely, which would cause some parts of some table spaces to have a different striping width than others.

When you add space to a database, try to extend all paths uniformly. That is, increase the capacity of each file system by the same amount. If you cannot extend the storage paths uniformly, add a new set of uniform storage paths by using the ADD STORAGE ON clause of the ALTER DATABASE statement. The new set of storage paths should have the same I/O capabilities as those of the original set. Ideally, add the same number of storage paths that you defined before, at the same time.

For example, to add space to the database MYDB that was created earlier, the best option is to increase the capacity of the file systems /data1, /data2, and /data3 by the same amount. If that is not possible, add a new set of storage paths (file systems) that have the same I/O characteristics as those of the original set, as shown in the following example:

```
DB2 ALTER DATABASE ADD STORAGE ON /data4, /data5, /data6
```

Because the original storage paths are uniform in size, they all fill up together, and table spaces that need additional storage get a new stripe set of containers, one for each of the new storage paths.

Avoid manually tune the num_iocleaners, num_ioservers, and prefetchsize configuration parameters

The default value for these parameters is `AUTOMATIC`. The DB2 database manager does an excellent job in selecting appropriate values for these parameters; therefore, you generally do not have to manually tune them.

Best practices

- Use 15 - 20 dedicated physical disks per CPU core.
- Ensure that you control the use of unused space for your LUNs.
- Dedicate LUNs to non-partitioned DB2 database servers and to database partitions.
- Stripe at most in two places.
- Use file systems instead of raw devices; create one file system per LUN.
- Set your table space EXTENTSIZE to RAID stripe size. If your EXTENTSIZE is the same as the strip size, consider using the **`DB2_PARALLEL_IO`** registry variable to increase prefetch parallelism.
- Use direct or concurrent I/O by using the NO FILE SYSTEM CACHING clause of the CREATE TABLESPACE and ALTER TABLESPACE statements.
- Use DB2 automatic storage for everything-everywhere striping.
- Use `AUTOMATIC` (the default) for the **`num_iocleaners`** **`num_ioservers`**, and **`prefetchsize`** configuration parameters .

Conclusion

Manually mapping individual database tables and indexes to individual physical disks and parts of physical disks is an exercise best left to decades past. The key to taming the complex, virtual, networked storage beast is keeping database storage design as simple as possible.

Although it seems counterintuitive at first, complexity is always best tackled with simplicity rather than with more complexity. While simplicity is not always easy, the best practices that are provided in this document provide an easy-to-follow steps for database storage design success.

Above all, a DBA's time and effort is best spent optimizing the database schema, rather than physical database storage. Optimizing the schema involves using functionality such as MDC and MQTs, creating appropriate indexes, and dropping inappropriate ones. After all, there is no higher throughput or lower latency I/O than one that does not have to be performed at all.

Further reading

- DB2 Best Practices
  http://www.ibm.com/developerworks/db2/bestpractices/

- IBM DB2 9.7 for Linux, UNIX, and Windows Information Center
  http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp

- DB2 9 for Linux, UNIX and Windows manuals
  http://www.ibm.com/support/docview.wss?rs=71&uid=swg27009552

- DB2 Data Warehouse Edition documentation
  http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.dwe.welcome.doc/dwe91docs.html

- IBM Data Warehousing and Business Intelligence
  http://www.ibm.com/software/data/db2bi/

Contributors

Danny F. Arnold
DB2 Technical Evangelist

John Bell
Chief Data Warehousing Solution Architect

Whit Smith
Certified IT Specialist

Linda Snow
Executive IT Specialist

Reuven Stepansky
Senior Managing Specialist
North American Lab Services

Tim Vincent
Chief Architect, DB2 for Linux, UNIX, and Windows

Dan Braden
AIX Advanced Technical Skills

Tom Hart
DB2 Software Developer

Joyce Simmonds
DB2 Information Development

Allan Risk
DB2 Information Development

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

Without limiting the above disclaimers, IBM provides no representations or warranties regarding the accuracy, reliability or serviceability of any information or recommendations provided in this publication, or with respect to any results that may be obtained by the use of the information or observance of any recommendations provided herein. The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any recommendations or techniques herein is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Anyone attempting to adapt these techniques to their own environment do so at their own risk.

This document and the information contained herein may be used solely in connection with the IBM products discussed in this document.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual

results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml

Windows is a trademark of Microsoft Corporation in the United States, other countries, or both.

Intel Xeon is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.