

Tuning I/O Subsystem: A Key Component in RDBMS Performance Tuning

Hitesh Kumar SHARMA¹, Christalin NELSON. S², Dr. Sanjeev Kumar SINGH³

¹Assistant Professor (SS), University of Petroleum & Energy Studies

²Assistant Professor (SG), University of Petroleum & Energy Studies

³Associate Professor, Galgotia University Noida

hkshitesh@gmail.com, cnelson@ddn.upes.ac.in, sksingh8@gmail.com

Abstract: *In a computer system, the fastest storage component is the CPU cache, followed by the system memory. I/O to disk is thousands of times slower than an access to memory. This fact is the key for why you try to make effective use of memory whenever possible and defer I/Os whenever you can. The majority of the user response time is actually spent waiting for a disk I/O to occur. By making good use of caches in memory and reducing I/O overhead, you can optimize performance. The goal is to retrieve data from memory whenever you can and to use the CPU for other activities whenever you have to wait for I/Os. This paper examines ways to optimize the performance of the system by taking advantage of caching and effective use of the system's CPUs.*

Keywords: *Tuning, I/O, RDBMS.*

1 Introduction

I/O is probably one of the most common problems facing RDBMS users. In many cases, the performance of the system is entirely limited by disk I/O. In some cases, the system actually becomes idle waiting for disk requests to complete. We say that these systems are *I/O bound* or *disk bound*. Disks have certain inherent limitations that cannot be overcome. Therefore, the way to deal with disk I/O issues is to understand the limitations of the disks and design your system with these limitations in mind. Knowing the performance characteristics of your disks can help you in the design stage. Optimizing your system for I/O should happen during the design stage. Different types of systems have different I/O patterns and require different I/O designs. Once the system is built, you should first tune for memory and then tune for disk I/O. The reason you tune in this order is to make sure that you are not dealing with excessive cache misses, which cause additional I/Os. The strategy for tuning disk I/O is to keep all drives within their physical limits. Doing so

reduces queuing time and thus increases performance. In your system, you may find that some disks process many more I/Os per second than other disks. These disks are called "hot spots." Try to reduce hot spots whenever possible. Hot spots occur whenever there is a lot of contention on a single disk or set of disks.

2. Understanding Disk Contention

Disk contention occurs whenever the physical limitations of a disk drive are reached and other processes have to wait. Disk drives are mechanical and have a physical limitation on both disk seeks per second and throughput. If you exceed these limitations, you have no choice but to wait. You can find out if you are exceeding these limits both through Oracle's file I/O statistics and through operating system statistics. Although the Oracle statistics give you an accurate picture of how many I/Os have taken place for a particular data file, they may not accurately represent the entire disk because other activity outside of Oracle may be incurring disk I/Os. Remember that you must correlate the Oracle data file to the physical disk on which it resides.

Information about disk accesses is kept in the dynamic performance table V\$FILESTAT.

Important information in this table is listed in the following columns:

- **PHYRDS:** The number of physical reads done to the data file.
- **PHYWRTS:** The number of physical writes done to the data file.

The information in V\$FILESTAT is referenced by file number. The dynamic performance table V\$DATAFILE contains a reference to this number as well as other useful information such as this:

- **NAME:** The name of the data file.
- **STATUS:** The type of file and its current status.
- **BYTES:** The size of the data file.

Together, the V\$FILESTAT and V\$DATAFILE tables can give you an idea of the I/O usage of your data files. Use the following query to get this information:

```
SQL> SELECT substr(name,1,40),
 2 phylds, phywrts, status, bytes
 3 FROM v$datafile df, v$filestat fs
 4 WHERE df.file# = fs.file#;
SUBSTR(NAME,1,40)      PHYRDS
PHYWRTS STATUS BYTES
-----
C:\UTIL\ORAWIN\DBS\wdbsys.ora 221
7 SYSTEM 10485760
C:\UTIL\ORAWIN\DBS\wdbuser.ora 0 0
ONLINE 3145728
C:\UTIL\ORAWIN\DBS\wdbrrs.ora 2 0
ONLINE 3145728
C:\UTIL\ORAWIN\DBS\wdbtemp.ora 0
0 ONLINE 2097152
```

The total I/O for each data file is the sum of the physical reads and physical writes. It is important to make sure that these I/Os don't exceed the physical limitations

of any one disk. I/O throughput problems to one disk may slow down the entire system depending on what data is on that disk. It is particularly important to make sure that I/O rates are not exceeded on the disk drives.

3. Identifying Disk Contention Problems

To identify disk contention problems, you must analyze the I/O rates of each disk drive in the system. If you are using individual disks or disk arrays, the analysis process is slightly different. For individual disk drives, simply invoke your operating system or third-party tools and check the number of I/Os per second on an individual disk basis. This process gives you an accurate representation of the I/O rates on each drive. A general rule of thumb is not to exceed 50 I/Os per second per drive with random access, or 100 I/Os per second per drive with sequential access. If you are experiencing a disk I/O problem, you may see excessive idle CPU cycles and poor response times. For a disk array, also invoke your operating system or third-party tools and check the same items specifically the number of I/Os per second per disk. The entire disk array appears as one disk. For most popular disk arrays on the market today, it is accurate to simply divide the I/O rate by the number of disks to get the I/Os per second per disk rate. The next step in identifying a disk contention problem is to determine the I/O profile for your disk. It is sufficient to split this into two major categories: sequential and random I/O. Here is what to look for:

- **Sequential I/O.** In sequential I/O, data is written or read from the disk in order, so very little head movement occurs. Access to the redo log files is always sequential.
- **Random I/O.** Random I/O occurs when data is accessed in different places on the disk, causing head movement. Access to data files is almost always random. For database loads, access is sequential; in most other cases (especially OLTP), the

access patterns are almost always random.

With sequential I/O, the disk can operate at a much higher rate than it can with random I/O. If *any* random I/O is being done on a disk, the disk is considered to be accessed in a random fashion. Even if you have two separate processes that access data in a sequential manner, the I/O pattern is random.

With random I/O, there is not only access to the disk but a large amount of head movement, which reduces the performance of the disks.

Finally, check these rates against the recommended I/O rates for your disk drives. Here are some good guidelines:

- **Sequential I/O.** A typical SCSI-II disk drive can support approximately 100 to 150 sequential I/Os per second.
- **Random I/O.** A typical SCSI-II disk drive can support approximately 50 to 60 random I/Os per second.

4. Solving Disk Contention Problems

There are a few rules of thumb you should follow in solving disk contention problems:

- **Isolate sequential I/Os.** Because sequential I/Os can occur at a much higher rate, isolating them lets you run these drives much faster.
- **Spread out random I/Os as much as possible.** You can do this by striping table data through Oracle striping, OS striping, or hardware striping.
- **Separate data and indexes.** By separating a heavily used table from its index, you allow a query to a table to access data and indexes on separate disks simultaneously.

- **Eliminate non-Oracle disk I/O from disks that contain database files.** Any other disk I/Os slow down Oracle access to these disks.

The following sections look at each of these solutions and determine how they can be accomplished.

4.1 Isolate Sequential I/Os

Isolating sequential I/Os allows you to drive sequentially accessed disks at a much higher rate than randomly accessed disks. Isolating sequential I/Os can be accomplished by simply putting the Oracle redo log files on separate disks. Be sure to put each redo log file on its own disk—especially the mirrored log file (if you are mirroring with Oracle). If you are mirroring with OS or hardware mirroring, the redo log files will already be on separate volumes. Although each log file is written sequentially, having the mirror on the same disk causes the disk to seek between the two log files between writes, thus degrading performance. It is important to protect redo log files against system failures by mirroring them. You can do this through Oracle itself, or by using OS or hardware fault tolerance features.

4.2 Spread Out Random I/Os

By the very nature of random I/Os, accesses are to vastly different places in the Oracle data files. This pattern makes it easy for random I/O problems to be alleviated by simply adding more disks to the system and spreading the Oracle tables across these disks. You can do this by striping the data across multiple drives or (depending on your configuration) by simply putting tables on different drives. *Striping* is the act of transparently dividing the contents of a large data source into smaller sources. Striping can be done through Oracle, the OS, or through hardware disk arrays.

4.3 Oracle Striping

Oracle striping involves dividing a table's data into small pieces and further dividing these pieces among different data files.

Oracle striping is done at the tablespace level with the CREATE TABLESPACE command. To create a striped tablespace, use a command similar to this one:

```
SQL> CREATE TABLESPACE
mytablespace
2 DATAFILE 'file1.dbf' SIZE 500K,
3 'file2.dbf' SIZE 500K,
4 'file3.dbf' SIZE 500K,
5 'file4.dbf' SIZE 500K;
Tablespace created.
```

To complete this task, you must then create a table within this tablespace with four extents. This creates the table across all four data files, which (hopefully) are each on their own disk.

Create the table with a command like this one:

```
SQL> CREATE TABLE mytable
2 ( name varchar(40),
3 title varchar(20),
4 office_number number(4) )
5 TABLESPACE mytablespace
6 STORAGE ( INITIAL 495K NEXT
495K
7 MINEXTENTS 4 PCTINCREASE 0 );
Table created.
```

In this example, each data file has a size of 500K. This is called the *stripe size*. If the table is large, the stripes are also large (unless you add many stripes). Large stripes can be an advantage when you have large pieces of data within the table, such as BLOBs. In most OLTP applications, it is more advantageous to have a smaller striping factor to distribute the I/Os more evenly. The size of the data files depends on the size of your tables. Because it is difficult to manage hundreds of data files, it is not uncommon to have one data file per disk volume per tablespace. If your database is 10 gigabytes in size and you have 10 disk volumes, your data file size will be 1

gigabyte. When you add more data files of a smaller size, your I/Os are distributed more evenly, but the system is harder to manage because there are more files. you can achieve both manageability and ease of use by using a hardware or software disk array. Oracle striping can be used in conjunction with OS or hardware striping.

4.4 OS Striping

Depending on the operating system, striping can be done at the OS level either through an operating system facility or through a third-party application. OS striping is done at OS installation time. OS disk striping is done by taking two or more disks and creating one large *logical disk*. In sequence, the stripes appear on the first disk, then the second disk, and so on (see Figure 1).

The size of each stripe depends on the OS and the striping software you are running. To figure out which disk has the desired piece of data, the OS must keep track of where the data is. To do this, a certain amount of CPU time must be spent maintaining this information. If fault tolerance is used, even more CPU resources are required. Depending on the software you are using to stripe

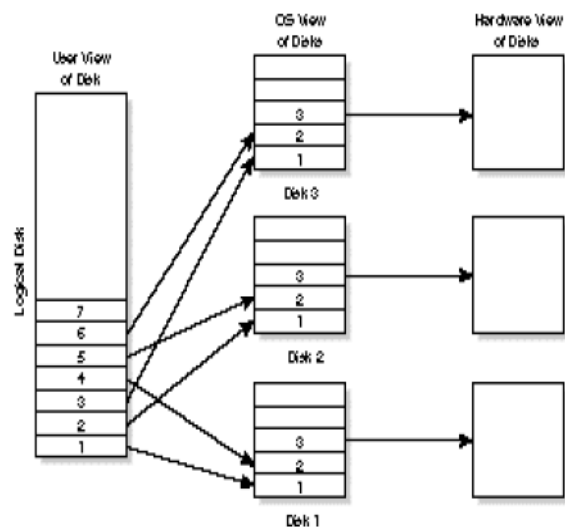


Fig 1: OS Striping

the disks, the OS monitoring facilities may display disk I/O rates on a per-disk basis or

on a per-logical-disk basis. Regardless of how the information is shown, you can easily determine the I/O rate per disk. Many of the OS-striping software packages on the market today can also take advantage of RAID technology to provide a measure of fault tolerance. OS striping is very good; however, It does consume system resources that hardware striping does not.

4.5 Hardware Striping

Hardware striping has a similar effect to OS striping. Hardware fault tolerance is obtained by replacing your disk controller with a disk array. A *disk array* is a controller that uses many disks to make one *logical* disk. The system takes a small slice of data from each of the disks in sequence to make up the larger logical disk (see Figure 2).

Hardware fault tolerance has the advantage of not taking any additional CPU or memory resources on the server. All the logic to do the striping is done at the controller level. As with OS striping, hardware striping can also take advantage of RAID technology for fault tolerance. As you can see in Figures 1 and 2, to the user and the RDBMS software, the effect is the same whether you use OS or hardware disk striping.

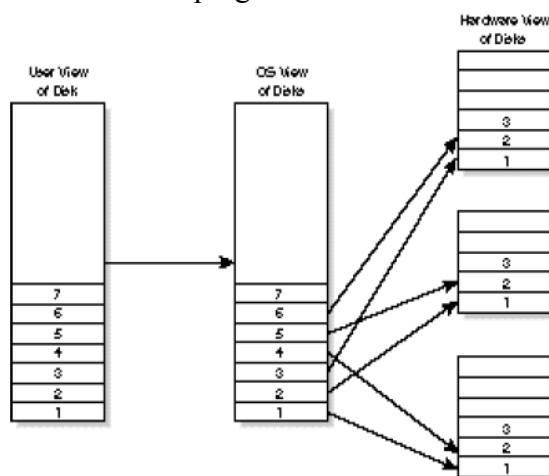


Fig 2: Hardware Striping

The main difference between the two is where the actual overhead of maintaining the disk array is maintained.

4.6 Review of Striping Options

Whether you use Oracle striping, OS striping, or hardware striping, the goal is the same: distribute the random I/Os across as many disks as possible. In this way, you can keep the number of I/Os per second requested within the bounds of the physical disks. If you use Oracle striping or OS striping, you can usually monitor the performance of each disk individually to see how hard they are being driven. If you use hardware striping, remember that the OS monitoring facilities typically see the disk volume as one logical disk. You can easily determine how hard the disks are being driven by dividing the I/O rate by the number of drives. With hardware and OS striping, the stripes are small enough that the I/Os are usually divided among the drives fairly evenly. Be sure to monitor the drives periodically to verify that you are not up against I/O limits.

Use this formula to calculate the I/O rate per drive:

$$\text{I/Os per disk} = (\text{Number of I/Os per second per volume}) / (\text{Number of drives in the volume})$$

Suppose that you have a disk array with four drives generating 120 I/Os per second. The number of I/Os per second per disk is calculated as follows:

$$\text{I/Os per disk} = 120 / 4 = 30 \text{ I/Os per second per disk}$$

For data volumes that are accessed randomly, you don't want to push the disks past 50 to 60 I/Os per second per disk.

To estimate how many disks you need for data volumes, use this formula:

$$\text{Number of disks} = \text{I/Os per second needed} / 60 \text{ I/Os per second per disk}$$

If your application requires a certain data file to supply 500 I/Os per second (based on analysis and calculations), you can estimate the number of disk drives needed as follows:

Number of disks = 500 I/Os per second /
60 I/Os per second per disk = 16 2/3
disks or 17 disks

This calculation gives you a good approximation for how large to build the data volumes with no fault tolerance.

4.7 Separate Data and Indexes

Another way to reduce disk contention is to separate the data files from their associated indexes. Remember that disk contention is caused by multiple processes trying to obtain the same resources. For a particularly “hot” table with data that many processes try to access, the indexes associated with that data will be “hot” also.

Placing the data files and index files on different disks reduces the contention on particularly hot tables. Distributing the files also allows more concurrency by allowing simultaneous accesses to the data files and the indexes. Look at the Oracle dynamic performance tables to determine which tables and indexes are the most active.

4.8 Eliminate Non-Oracle Disk I/Os

Although it is not necessary to eliminate all non-Oracle I/Os, reducing significant I/Os will help performance. Most systems are tuned to handle a specific throughput requirement or response time requirement. Any additional I/Os that slow down Oracle can affect both these requirements. Another reason to reduce non-Oracle I/Os is to increase the accuracy of the Oracle dynamic performance table, V\$FILESTAT. If only Oracle files are on the disks you are monitoring, the statistics in this table should be very accurate.

5. Reducing Unnecessary I/O Overhead

Reducing unnecessary I/O overhead can increase the throughput available for user tasks. Unnecessary overhead such as *chaining* and *migrating* of rows hurts performance. Migrating and chaining occur when an UPDATE statement increases the size of a row so that it no longer fits in the data block. When this happens, Oracle tries to find space for this new row. If a block is available with enough room, Oracle moves the entire row to that new block. This is called *migrating*. If no data block is available with enough space, Oracle splits the row into multiple pieces and stores them in several data blocks. This is called *chaining*.

6. Migrated and Chained Rows

Migrated rows cause overhead in the system because Oracle must spend the CPU time to find space for the row and then copy the row to the new data block. This takes both CPU time and I/Os. Therefore, any UPDATE statement that causes a migration incurs a performance penalty. Chained rows cause overhead in the system not only when they are created but each time they are accessed. A chained row requires more than one I/O to read the row. Remember that Oracle reads from the disk data blocks; each time the row is accessed, multiple blocks must be read into the SGA.

You can check for chained rows with the LIST CHAINED ROWS option of the ANALYZE command.

You can use these SQL statements to check for chained or migrated rows:

```
SQL> Rem
SQL> CREATE TABLE chained_rows (
2 owner_name varchar2(30),
3 table_name varchar2(30),
4 cluster_name varchar2(30),
5 head_rowid rowid,
6 timestamp date);
Table created.
SQL> Rem
SQL> Rem Analyze the Table in Question
```

```

SQL> Rem
SQL> ANALYZE
2 TABLE scott.emp LIST CHAINED
ROWS;
Table analyzed.
SQL> Rem
SQL> Rem Check the Results
SQL> Rem
SQL> SELECT * from chained_rows;
no rows selected

```

If any rows are selected, you have either chained or migrated rows. To solve the problem of migrated rows, copy the rows in question to a temporary table, delete the rows from the initial table, and reinsert the rows into the original table from the temporary table. Run the chained-row command again to show only chained rows. If you see an abundance of chained rows, this is an indication that the Oracle database block size is too small. You may want to export the data and rebuild the database with a larger block size. You may not be able to avoid having chained rows, especially if your table has a LONG column or long CHAR or VARCHAR2 columns. If you are aware of very large columns, it can be advantageous to adjust the database block size before implementing the database. A properly sized block ensures that the blocks are used efficiently and I/Os are kept to a minimum. Don't over-build the blocks or you may end up wasting space. The block size is determined by the Oracle parameter DB_BLOCK_SIZE. Remember that the amount of memory used for database block buffers is calculated as follows:

```

Memory used = DB_BLOCK_BUFFERS
(number) * DB_BLOCK_SIZE (bytes)

```

Be careful to avoid paging or swapping caused by an SGA that doesn't fit into RAM.

7. Dynamic Extensions

Additional I/O is generated by the extension of segments. Remember that segments are allocated for data in the database at creation time. As the table grows, extents are added to accommodate this growth. Dynamic extension not only causes additional I/Os, it also causes additional SQL statements to be executed. These additional calls, known as *recursive calls*, as well as the additional I/Os can impact performance.

You can check the number of recursive calls through the dynamic performance table, V\$SYSSTAT.

Use the following command:

```

SQL> SELECT name, value
2 FROM v$SYSSTAT
3 WHERE name = 'recursive calls';
NAME VALUE
-----
recursive calls 5440

```

Check for recursive calls after your application has started running and then 15 to 20 minutes later. This information will tell you approximately how many recursive calls the application is causing. Recursive calls are also caused by the following:

- Execution of Data Definition Language statements.
- Execution of SQL statements within stored procedures, functions, packages, and anonymous PL/SQL blocks.
- Enforcement of referential integrity constraints.
- The firing of database triggers.
- Misses on the data dictionary cache.

As you can see, many other conditions can also cause recursive calls. One way to check whether you are creating extents dynamically is to check the table DBA_EXTENTS. If you see that many extents have been created, it may be time to export your data, rebuild the tablespace, and reload the data. Sizing a segment large enough to fit your data properly benefits you in two ways:

- Blocks in a single extent are contiguous and allow multiblock reads to be more effective, thus reducing I/O.
- Large extents are less likely to be dynamically extended.

Try to size your segments so that dynamic extension is generally avoided and there is adequate space for growth.

8. Conclusion

In this paper we have explained the impact of efficient configuration of I/O for enhancing the performance of RDBMS. For practical explanation we have used one of the popular RDBMS i.e. oracle 10g. We have suggested many parts of I/O subsystem those impact the performance of RDBMS.

References

- [1]. Lightstone, S. *et al.*, "Toward Autonomic Computing with DB2 Universal Database", *SIGMOD Record*, Vol. 31, No.3, September 2002.
- [2]. Xu, X., Martin, P. and Powley, W., "Configuring Buffer Pools in DB2 UDB", IBM Canada Ltd., the National Science and Engineering Research Council (NSERC) and Communication and Information Technology Ontario (CITO), 2002.
- [3]. Chaudhuri, S. (ed). Special Issue on, "Self-tuning Databases and Application Tuning", *IEEE Data Engineering Bulletin* 22(2), June 1999.
- [4]. Bernstein, P. *et al.*, "The Asilomar Report on Database Research", *ACM SIGMOD Record* 27(4), December 1998, pp. 74 - 80.
- [5]. Nguyen, H. C., Ockene, A., Revell, R., and Skwish, W. J., "The role of detailed simulation in capacity planning". *IBM Syst. J.* 19, 1 (1980), 81-101.
- [6]. Seaman, P. H., "Modeling considerations for predicting performance of CICS/VS systems", *IBM Syst. J.* 19, 1 (1980), 68-80.
- [7]. Foster, D. V., McGehearty, P. F., Sauer, C. H., and Waggoner, C. N., "A language for analysis of queuing models", *Proceedings of the 5th Annual Pittsburgh Modeling and Simulation Conference* (Univ. of Pittsburgh, Pittsburgh, Pa., Apr. 24-26). 1974, pp. 381-386.
- [8]. Reiser, M., and Sauer, C. H., "Queuing network models: Methods of solution and their program implementation", *Current Trends in Programming Methodology*. Vol. 3, Software Modeling and Its Impact on Performance, K. M. Chandy and R. T. Yeb, Eds. Prentice-Hall, Englewood Cliffs, N. J., 1978, pp. 115-167.
- [9]. Borovits, I., and Neumann, S., "Computer Systems Performance Evaluation", *D.C. Heath and Co., Lexington, Mass.*, 1979.
- [10]. Enrique Vargas, "High Availability Fundamentals", *Sun BluePrints™ OnLine*, November 2000, <http://www.sun.com/blueprints>
- [11]. Harry Singh, "Distributed Fault-Tolerant/High-Availability Systems", *Trillium Digital Systems, a division of Intel Corporation, 12100 Wilshire Boulevard, Suite 1800 Los Angeles, CA 90025-7118 U.S.A. Document Number 8761019.12.*
- [12]. David McKinley, "High availability system. High availability system platforms", *Dedicated Systems Magazine* - 2000 Q4 (<http://www.dedicated-systems.com>)
- [13]. Sasidhar Pendyala, "Oracle's Technologies for High Availability", Oracle Software India Ltd., India Development Centre.
- [14]. James Koopmann, "Database Performance and some Christmas Cheer", an article in the *Database Journal*, January 2, 2003.
- [15]. Frank Naudé, "Oracle Monitoring and Performance Tuning", <http://www.orafaq.com/faqdbapf.htm>

- [16]. Michael Marxmeier, "Database Performance Tuning",
- [17]. <http://www.hpeloquence.com/support/misc/dbtuning.html>
- [18]. Sharma H., Shastri A., Biswas R. "Architecture of Automated Database Tuning Using SGA Parameters", *Database System Journal*, Romania, 2012
- [19]. Sharma H., Shastri A., Biswas R. "A Framework for Automated Database Tuning Using Dynamic SGA Parameters and Basic Operating System Utilities", *Database System Journal*, Romania, 2013
- [20]. Mihyar Hesson, "Database performance Issues"
- [21]. PROGRESS SOFTWARE, Progress Software Professional Services,

Hitesh Kumar Sharma: Author is an Assistant Professor (Senior Scale) in University of Petroleum & Energy Studies, Dehradun. He has published 30+ research papers in International Journal and 10+ research papers in National Journals.

Christalin Nelson. S: Author is an Assistant Professor (Selection Grade) in University of Petroleum & Energy Studies, Dehradun. He has published 40+ research papers in International Journal and 12 research papers in National Journals. He is Programme Head of the computer Science Department.

Sanjeev Kumar Singh: Author is an Associate Professor in Galgotias University, Noida. He has published 35+ research paper in International Journal and 15+ research papers in National Journals. He is Ph.D. in Mathematics.