

DB2 Performance Tuning -

Taking Snapshots -

Determining if a database parameter needs to be tuned will usually require examining statistics from a database snapshot. In order to take a snap shot you will need to make sure the different monitors are on before hand. The general procedure to get a snapshot is to do the following -

Verify what monitors are on -

```
db2 -v get monitor switch
```

Turn on monitors

```
db2 -v update monitor switches using bufferpool on
```

```
db2 -v update monitor switches using lock on
```

```
db2 -v update monitor switches using sort on
```

```
db2 -v update monitor switches using statement on
```

```
db2 -v update monitor switches using table on
```

```
db2 -v update monitor switches using timestamp on
```

```
db2 -v update monitor switches using UOW on
```

```
or - update dbm cfg using dft_mon_bufpool on dft_mon_lock on dft_mon_sort on  
dft_mon_stmt on dft_mon_table on dft_mon_uow on health_mon on;
```

```
db2 -v get monitor switches
```

Clear the monitors -

```
db2 -v reset monitor all
```

-- run the application in question --

Collect the Snapshot -

There are a number of snapshots you can take -

Snapshot Type	Command
Locks Snapshot	db2 get snapshot for locks on bgdb0
Database Manager Snapshot	db2 get snapshot for dbm
Database Snapshot	db2 get snapshot for database on bgdb0
Tablespace Snapshot	db2 get snapshot for tablespaces on bgdb0
Bufferpool Snapshot	db2 get snapshot for bufferpools on bgdb0
Applications	db2 get snapshot for applications on bgdb0
Dynamic SQL	db2 get snapshot for dynamic sql on bgdb0

For performance Issues we usually only care about the dbm, db, and bufferpool snapshots -

```
db2 -v get snapshot for all databases > /tmp/snap.out
db2 -v get snapshot for dbm >> /tmp/snap.out
db2 -v get snapshot for bufferpools >> /tmp/snap.out
db2 -v reset monitor all
db2 -v terminate
```

DB2 Parameters

Buffer Pools -

Recommendations -

The bufferpools main function is to improve database performance since it is the area of memory used by the database to read in and modify pages. So it is usually beneficial to make the bufferpools as large as possible without causing system stability issues. By default we set BGP systems to have a bufferpool size of 1.6GB which should be sufficient for most systems, however if your system has the memory you can experiment with increasing this.

Changing the Parameter -

You can either update the bufferpool size directly in syscat.bufferpools, or by setting the value of NPAGES to -1 and then by modifying the BUFFPAGE value

```
db2 connect to DB_NAME
db2 select * from syscat.bufferpools
db2 alter bufferpool IBMDEFAULTBP size -1
db2 connect reset
db2 update db cfg for dbname using BUFFPAGE bigger_value
db2 terminate
```

Investigative Steps -

To determine if the buffer pool size is big enough, you will need to examine the bufferpool hit ratio to see if the database is pulling too many pages from disk rather than memory. It indicates the percentage of time that the database manager did not need to load a page from disk in order to service a page request because the page was already in the buffer pool. The greater the buffer pool hit ratio, the lower the volume of disk I/O. To calculate this you will need to look at a few values from the bufferpool snapshot. Collect that snapshot while taxing the system by running -

```

db2 -v update monitor switches using bufferpool on
db2 -v get monitor switches
db2 -v reset monitor all
-- run BGP jobs --
db2 -v get snapshot for bufferpools >> /tmp/snap.out
db2 -v reset monitor all
db2 -v terminate

```

Calculate the bufferpool hit ratio by examining “logical reads” and “physical reads” as follows -

$$\left(1 - \left(\frac{(\text{buffer pool data physical reads} + \text{buffer pool index physical reads})}{(\text{buffer pool data logical reads} + \text{buffer pool index logical reads})} \right) \right) * 100\%$$

The hit ratio should be 95% or more. The higher the better.

Apart from simply increase the buff pool size, another way to improve performance is to create multiple bufferpools for frequently accessed large tables.

Number of Agents (MAXAGENTS, MAX_COORDAGENTS, NUM_INITAGENTS) -

Recommendations -

By default we like to enable the connection concentrator and connection pooling by setting the value of *max_connections* greater than the value of *max_coordagents*. This helps to reduce the load on overall system resources. Typically you will need to increase the *maxagents* based on how many concurrent blocks you plan to boot. For reference a system booting 400 blocks at a time typically will need a maxagents setting of 1500.

Changing the Parameter -

```

db2 -v update dbm cfg using MAXAGENTS <value>
db2 -v update dbm cfg using NUM_POOLAGENTS <value>
db2 -v update dbm cfg using NUM_INITAGENTS <value>
db2 -v terminate

```

max_coordagents is usually set to *maxagents - num_initagents* . You can set it to a real value like above, or set it to the default of *maxagents - num_initagents* via -

db2 -v update dbm cfg using MAX_COORDAGENTS -1

Investigative Steps -

Look at the snapshot for the database manager (db2 -v get snapshot for database manager). If you see that “Agents waiting for a token” or “Agents stolen from another application” is not equal to 0 then you may need to increase *maxagents* to allow more agents to be available to the db manager.

You can also look at the “High water mark” values for connections, agents, coordinating agents, etc. The output will look similar to this -

```
High water mark for connections           = 1205
Application connects                     = 5409
Secondary connects total                 = 3
Applications connected currently         = 31
Appls. executing in db manager currently = 0
Agents associated with applications       = 3
Maximum agents associated with applications= 1194
Maximum coordinating agents              = 1194
```

Using the connection concentrator you can get a situation where the database appears to hang. If there's a new incoming request, but all agents are occupied serving a transaction, the incoming transaction will appear to hang. This is because no agent is available to service your request. When you see situations, such as user requests that appear to hang your *max_coordagents* setting might not be high enough. This is typically controlled by *max_agents*, so increasing *maxagents* will increase *max_coordagents*.

Log Buffer Size – LOGBUFSZ

Recommendations -

This is the amount of db shared memory to use as a buffer for log records before writing these records to disk. Log records are written to disk when a transaction commits, the log buffer is full, or due to a db manager event

Increase if seeing high read activity on the disks that house the logs, or high disk utilization in general (iostat will show this). Increase *dbheap* too since the log buffer area uses space controlled by the *dbheap* parameter

By default we generally set LOGBUFSZ to 128.

Changing the Parameter -

The Log buffer is in 4 kb pages

```
db2 -v update database cfg for DB_NAME using LOGBUFSZ 256
db2 -v terminate
```

Investigative Steps -

Look at the db snapshot, and examine the Log pages read versus the Log pages written. Ideally the read pages should be 0 with numerous writes. If there are a bunch of reads you will need to increase the Buffer.

Application Heap Size – APPHEAPSZ

Recommendations -

This is the amount of memory needed to process a request given to a agent from an application. By default we set this to 2500.

Changing the Parameter -

```
db2 -v update db cfg for DB_NAME using applheapsz 256
```

Investigative Steps -

If the db2diag.log shows an error from an app complaining of not having enough heap storage then increase this parameter.

Sort Heap Size and Sort Heap Threshold (SORTHEAP / SHEAPTHRES) -

Recommendations -

Certain memory settings of db2 can now be automatically maintained by the database manager. We recommend setting both *sortheap* and *sheapthres* to AUTOMATIC. This allows db2 to modify the values as needed.

Changing the Parameter -

```
db2 -v update db cfg for DB_NAME using SORTHEAP <value>
db2 -v update dbm cfg using SHEAPTHRES <value>
db2 -v terminate
```

a value of AUTOMATIC above will make db2 maintain both heaps.

Investigative Steps -

db2 -v monitor switches using sort on
db2 -v get snapshot for database on DBNAME

Calculate the number of sorts per transaction and the % of sorts that overflow the memory that was available to them -

```
SortsPerTransaction =  
    (Total Sorts)  
    /  
    (Commit statements attempted + Rollback statements attempted)  
  
PercentSortOverflow = (Sort overflows * 100 ) / (Total sorts)
```

If SortsPerTransaction > 5 there might be too many sorts per transaction. If PercentSortOverflow is > 3% there may be serious large sorts occurring. Increasing SORTHEAP might help the symptom, but correct indexes will solve the problem.

Locks (LOCKLIST,MAXLOCKS, and LOCKTIMEOUT)

Recommendations -

The *locklist* is the amount of memory allocated to the list of locks and *maxlocks* is the percentage of that list that an application must hold before the database throws up a lock escalation. It also escalates locks when the locklist becomes full. Again, in DB2 9 these values can be automatically controlled by the database manager which we suggest. We generally leave *locktimeout* at -1 leaving lock timeout detection turned off. That means applications will wait for a lock until they have it granted or until a deadlock occurs

Changing the Parameter -

```
db2 -v update db cfg for db_name using LOCKLIST <value>  
db2 -v update db cfg for db_name using MAXLOCKS <value>  
db2 -v update db cfg for db_name using LOCKTIMEOUT <value>  
db2 -v terminate
```

Investigative Steps -

Look at the lock entries from the db snapshot. If the “Lock List memory in use (Bytes)” exceeds 50% of the defined *locklist* size, then increase the number of 4KB pages in the *locklist*. Lock escalations, timeouts and deadlocks will indicate potential system or application problems. The locking problems normally indicate concurrency problems in a application running on the server.

MAXFILOP -

If this value is small, db2 spends extra cpu time closing down files, being a good citizen and giving over resources to other operating system processes. If the number of files closed is greater than zero in a db snapshot, then you can incrementally increase this value to stop db2 from closing files.

CATALOGCACHE_SZ

This parameter helps db2 to shorten SQL statement plan preparation times. If the db, tablespace, tables, indexes, and views, are all in the cache db2 can learn about the plan faster. You should strive for a high package hit ratio of 95% or better. A db snapshot will show the ratio via - $100 - ((\text{Catalog cache inserts} \times 100) / \text{Catalog cache lookups})$. Also increase this if the Catalog cache overflow, and cache heap full is greater than zero. DBHEAP and CATALOGCACHE_SZ should be increased in tandem.

MINCOMMIT

Helps to group i/o to db2 logs together. First determine how many transactions per second the db is performing, and divide this by 10 - $\text{Commits} + \text{Rollbacks} (\text{Transactions}) / 10 = \text{MINCOMMIT}$.

In a DW db MINCOMMIT should be set to 1.

INTRA_PARALLEL

OLTP databases should have INTRA_PARALLEL set to NO, but DW databases should set it to YES to enable CPU parallelism, and have DFT_DEGREE set to ANY or -1.

MAX_QUERYDEGREE

OLTP databases should set this to 1, and DW databases should set this equal to the number of CPUs on the system, to prevent a user from setting their current degree to a value too high.

FCM_NUM_BUFFERS

FCM Manages communications between parallel agents. A shortage of buffers will cause FCM resource shortage messages in the db2diag.log file. A db snapshot will show existing usage

$\text{Free FCM buffers low water mark} / \text{Free FCM buffers} * 100 = \text{Percentage of FCM Buffers Used}$

$\text{Free FCM message anchors low water mark} / \text{Free FCM message anchors} * 100 = \text{Percentage of FCM Message Anchors Used}$
 $\text{Free FCM connection entries low water mark} / \text{Free FCM connection entries} * 100$

$= \text{Percentage of FCM Connection Entries Used}$

$\text{Free FCM request blocks low water mark} / \text{Free FCM request blocks} * 100 =$

Percentage of FCM Request Blocks Used

If any values are less than 15% you'll need to increase the initial allocation until all low water marks are greater than 15%.

DFT_QUERYOPT -

Defines how long db2 will take analyzing its sql execution plans. The smaller the value the less time db2 spends. Setting this to 1 is ok in OLTP dbs but in DW dbs because of the complex sql, setting this to 7 or 9 to spend a little more time on the access plans can be desired.

CHNGPGS_THERSH -

Defines the threshold / percent of dirty pages needed in the buffer pools before the NUM_IO_CLEANERS begin to write changed pages out to disk. The default of 60 is good for DW dbs, but OLTP db's might want to lower this to 50 or 40. If the value is too high, when the db finally does write out pages, it can be overwhelming to some apps, so doing it fewer times at smaller volumes can help reduce this. No one sets it below 30 though.

NUM_IO_CLEANERS -

This value deals with asynchronous writes, and effects the asynchronous write percentage, which should be 90% or higher. Formula = $(\text{Asynchronous pool data page writes} + \text{Asynchronous pool index page writes}) \times 100 / (\text{Buffer pool data writes} + \text{Buffer pool index writes})$

Generally setting this to the # of CPUs should be sufficient

NUM_IO_SERVERS -

Used to prefetch data into db2's buffer pools. To set this value, add up the number of physical disks on the db2 server, and use that number.

RAID Disk and Parallelism -

In a DW environment using regular SCSI or IDE disk drives, tablespaces should have multiple containers on different disks. For RAID devices where several disks appear as one to the operating system, be sure to do the following:

1. db2set DB2_STRIPED_CONTAINERS=YES (do this before creating tablespaces or before a redirected restore)
2. db2set DB2_PARALLEL_IO=* (or use TablespaceID numbers for tablespaces residing on the RAID devices — for example DB2_PARALLEL_IO=4,5,6,7,8,10,12,13)

3. Alter the tablespace PREFETCHSIZE for each tablespace residing on RAID devices such that the PREFETCHSIZE is a multiple of the EXTENTSIZE. Most companies use a multiple of three to five. Four is my favorite.

Proper container placement and alignment of the extent with the raid stripe size can reduce I/O times by 50 percent. Remember, I/O is still the slowest component in the transaction mix.

Other DB2 Performance Tools -

DB2PD -

db2pd -osinfo - will display information about the system being run on
db2pd -db dbname -pages - This will show what is using the bufferpools, you can pass the bufferpool id to just look at that object. Next you can run db2pd with -tcbstats to correlate ObjID's to the table names using the bufferpools
db2pd -db dbname -tcbstats - will show tablename and their ObjID's

db2pd -db dbname -logs - shows the status of the log archiver and how fast logs are filling up
db2pd -db dbname -logs -repeat 60 10 - runs db2pd every 60 seconds a total of 10 times

db2pd -db dbname -applications - This will list out all apps currently running in the db. Part of the info might show apps with a status of Lock-wait. To view the lock the app is waiting on run -

db2pd -db dbname -locks wait - this might show output similar to -

Locks:

TranHdl	Lockname	Type	Mode	Sts
6	030011000600A00A00000000052	Row	..X	W
2	030011000600A00A00000000052	Row	..X	G

which shows a row lock .

db2pd -db dbname -locks showlocks wait - dumps out the object information like -

Locks:

TranHdl	Type	Mode	Sts	TbpaceID	TableID	Page	Slot
6	Row	..X	W	3	17	2720	6
2	Row	..X	G	3	17	2720	6

db2pd -db dbname -tcbstats - will show the table name from the tableid

You can see what is under contention with db2dart – db2dart dbname /dd. It will ask for the table_id, tablespace_id, and the page to dump

Bufferpools -

db2pd -db dbname -bufferpools

Here are the columns it will produce -

- * Id - Bufferpool id
- * Name - Bufferpool name
- * PageSz - Page size for this bufferpool
- * PA-NumPgs - Number of pages in this bufferpool
- * BA-NumPgs - Number of pages in the block based portion of this bufferpool
- * NumTbsp - Number of tablespaces that are using this bufferpool
- * CurrentSz - Current size (in pages) of this bufferpool
- * PostAlter - If you have altered the size of the bufferpool and DB2 is currently shrinking or growing in size, this is the post alter size.
- * DatLRds - Number of logical data page reads for this bufferpool
- * DatPRds - Number of physical data page reads for this bufferpool
- * HitRatio - Hit ratio for data pages given the above logical and physical reads
- * IdxLRds - Number of logical index page reads for this bufferpool
- * IdxPRds - Number of physical index page reads for this bufferpool
- * HitRatio - Hit ratio for index pages given the above logical and physical reads
- * DataWrts - Number of data pages written out from this bufferpool
- * IdxWrts - Number of index pages written out for this bufferpool
- * DirRds - Number of direct reads
- * AsDatRds - Number of asynchronous data page reads
- * UnRdPFetch - Number of pages prefetched into the bufferpool but never read by an agent.

db2pd -db dbname -tcbstat <tablespaceid tabid> i.e. Db2pd -db bgdb0 -tcbstat 2 68 will show stats for a specific table. You can query just by tablespace id if desired.

Db2pd -everything - will dump all options the db2pd command has

DB2 Callout Script (db2cos) -

db2 9 has a new feature, db2cos (db2 call out script), which is executed any time the instance traps, panics, gets a seg fault or an exception causing the dbm to stop executing. You can put any script in the db2cos script, to dump info. By default it runs db2pd. So you can have it run db2pd -everything to dump a wealth of information.

There is a default db2cos script in /opt/ibm/db2/V9.1/bin that you can override with one in the instances sqllib directory

db2pdcfg can help you determine when to run db2cos -

db2pdcfg -catch sqlcode,reason_code action - sqlcode is the code you want to catch and reason_code is optional. By default action is db2cos, but you can set it to something else.

db2pdcfg -catch -911,2 - catches deadlocks

db2pdcfg -catch -911,68 - catches lock timeouts

db2pdcfg -catch -289 - catches everything when a tablespace full condition hit

DB2 Admin Views -

Show messages in the notify log in the last 24 hours of a certain severity -

```
SELECT TIMESTAMP, SUBSTR(MSG,1,400) AS MSG
FROM SYSIBMADM.PDLOGMSGS_LAST24HOURS
WHERE MSGSEVERITY IN ('C','E')
ORDER BY TIMESTAMP DESC
```

Select messages beyond 24 hours -

```
SELECT TIMESTAMP, SUBSTR(MSG,1,400) AS MSG
FROM TABLE (PD_GET_LOG_MSGS ( CURRENT TIMESTAMP - 3 DAYS)) AS PD
ORDER BY TIMESTAMP DESC
```

the following query shows the average time taken (in minutes) and the maximum time taken for full backups

```
SELECT AVG(TIMESTAMPDIFF(4,CHAR(TIMESTAMP(END_TIME)
- TIMESTAMP(START_TIME)))) AS AVG_BTIME,
MAX(TIMESTAMPDIFF(4,CHAR(TIMESTAMP(END_TIME)
- TIMESTAMP(START_TIME)))) AS MAX_BTIME
FROM SYSIBMADM.DB_HISTORY
WHERE OPERATION = 'B'
AND OPERATIONTYPE = 'F'
```

Or when was that last time the TBGPEVENTLOG table was reorganized

```
SELECT START_TIME
FROM SYSIBMADM.DB_HISTORY
WHERE TABNAME = 'TBGPEVENTLOG'
AND OPERATION = 'G'
```

Or any operation in the history file that failed

```
SELECT START_TIME, SQLCODE, SUBSTR(CMD_TEXT,1,50)
FROM SYSIBMADM.DB_HISTORY
WHERE SQLCODE < 0
```

Here is an example query that shows you any statements that have been executing for more than one minute along with the authid, the status of the application and the first few characters of the statement text.

```
SELECT ELAPSED_TIME_MIN, SUBSTR(AUTHID,1,10) AS AUTH_ID, AGENT_ID,
       APPL_STATUS, SUBSTR(STMT_TEXT,1,20) AS SQL_TEXT
FROM SYSIBMADM.LONG_RUNNING_SQL
WHERE ELAPSED_TIME_MIN > 0
ORDER BY ELAPSED_TIME_MIN DESC
```

To find who is waiting on locks and who is holding those locks being waited on -

```
SELECT SMALLINT(AGENT_ID) AS WAITING_ID,
       SUBSTR(APPL_NAME, 1,10) AS WAITING_APP,
       SUBSTR(AUTHID,1,10) AS WAITING_USER,
       SMALLINT(AGENT_ID_HOLDING_LK) AS HOLDER_ID,
       LOCK_MODE AS HELD,
       LOCK_OBJECT_TYPE AS TYPE,
       LOCK_MODE_REQUESTED AS REQUEST
FROM SYSIBMADM.LOCKWAITS
```

You can drill further using other snapshot views to see who is the holder:

```
SELECT SUBSTR(APPL_NAME, 1,10) AS HOLDING_APP,
       SUBSTR(PRIMARY_AUTH_ID,1,10) AS HOLDING_USER,
       SUBSTR(CLIENT_NNAME,1,20) AS HOLDING_CLIENT,
       APPL_STATUS
FROM SYSIBMADM.SNAPAPPL_INFO
WHERE AGENT_ID = 831
```

Show the size of all tables in the BGPSYSDB schema, displayed in KB.

```
SELECT SUBSTR(TABSCHEMA,1,10) AS SCHEMA,
       SUBSTR(TABNAME,1,15) AS TABNAME,
       INT(DATA_OBJECT_P_SIZE) AS OBJ_SZ_KB,
       INT(INDEX_OBJECT_P_SIZE) AS INX_SZ_KB,
       INT(XML_OBJECT_P_SIZE) AS XML_SZ_KB
FROM SYSIBMADM.ADMINTABINFO
WHERE TABSCHEMA='BGPSYSDB'
ORDER BY 3 DESC
```

Display the sum of all objects by schema

```
SELECT SUBSTR(TABSCHEMA,1,10) AS SCHEMA,
       SUM(DATA_OBJECT_P_SIZE) AS OBJ_SZ_KB,
       SUM(INDEX_OBJECT_P_SIZE) AS INX_SZ_KB,
       SUM(XML_OBJECT_P_SIZE) AS XML_SZ_KB
FROM SYSIBMADM.ADMINTABINFO
GROUP BY TABSCHEMA
ORDER BY 2 DESC
```

To view the db2 registry variables that are set -

```
SELECT SUBSTR(REG_VAR_NAME,1,20) AS NAME,  
        DBPARTITIONNUM,  
        SUBSTR(REG_VAR_VALUE,1,20) AS VALUE  
FROM SYSIBMADM.REG_VARIABLES  
WHERE REG_VAR_NAME = 'DB2_WORKLOAD'
```

Display the most frequently executed statement

```
SELECT SUBSTR(STMT_TEXT,1,50), NUM_EXECUTIONS  
FROM SYSIBMADM.TOP_DYNAMIC_SQL  
ORDER BY NUM_EXECUTIONS DESC  
FETCH FIRST ROW ONLY
```

Or the statement with the highest average execution time:

```
SELECT SUBSTR(STMT_TEXT,1,50), AVERAGE_EXECUTION_TIME_S  
FROM SYSIBMADM.TOP_DYNAMIC_SQL  
ORDER BY AVERAGE_EXECUTION_TIME_S DESC  
FETCH FIRST ROW ONLY
```