IBM® Smart Analytics System

Best practices

Performance monitoring in a data warehouse

Toni Bollinger
IBM Data Warehousing and Advanced Analytics Specialist

Detlev Kuntze
IBM Data Warehousing Center of Excellence

Gregor Meyer
IBM Data Warehousing Center of Excellence

Sascha Laudien
IBM Data Warehousing Center of Excellence

Farzana Anwar
IBM Information Development

Issued: September 2013

Executive summary

Monitoring a data warehouse system is important to help ensure that it performs optimally. This paper describes the most important DB2 software and operating system metrics for monitoring the performance of the IBM Smart Analytics System or IBM PureData™ System for Operational Analytics or a system having a similar architecture. This paper also presents a general methodology to help find reasons for performance problems. This approach starts with the operating system metrics and drills down to the DB2 metrics that explain the behaviour that is seen at the operating system level and help to identify the causes of performance problems. This approach is illustrated by information about typical performance problems.

Introduction

This best practices paper covers real-time monitoring of the IBM Smart Analytics System and IBM PureData System for Operational Analytics. You can apply most of the content to other types of clusters of servers running a data warehouse system with DB2 software and database partitioning under AIX and Linux operating systems. The focus of this paper is finding the reasons for performance problems. These can be bottlenecks that are in the operating system, are in the DB2 database software, or are related to a single query. The focus is on data warehouse systems with long-running queries rather than transactional systems with mostly short queries.

A main goal of this paper is to provide a set of key performance indicators (KPIs) or metrics for the operating system and DB2 software, along with a methodology for analyzing performance problems in a distributed DB2 environment. This paper describes scenarios to help you gather the right information depending on the symptoms of the performance problem.

This paper first provides an overview of the approach and what to consider in general when monitoring the performance of a data warehouse. It then describes the most important operating system and DB2 metrics for multiserver data warehouse systems. The last section describes in detail several performance problem scenarios that are related to data warehouse or BI workloads and explains how to use the metrics for analyzing the problems.

Most of the information about KPIs that are described in the paper has sample commands that extract actual values. However, these examples are not intended to provide comprehensive tooling. You can use this best practices paper as a guideline for selecting the metrics to focus on when using monitoring tools such as IBM InfoSphere® Optim™ Performance Manager. You can use this paper to complement the best practices paper *Managing data warehouse performance with IBM InfoSphere Optim Performance Manager*, published in September 2012, which covers historical and end-to-end monitoring.

Performance monitoring methodology

Monitoring a database system requires an understanding of the various performance measures and how to interpret them relative to overall system usage. The following measures are explained in the next sections:

- Operating system measures:
  - CPU or thread utilization
  - Main memory usage and swapping
  - Network utilization
  - Disk usage
- DB2 measures:
  - DB2 memory usage
  - Locking characteristics
  - Number of sort and hash join overflows
  - Buffer pool performance
  - CPU or data skew

To determine whether a situation is just a normal peak or something more critical, try to get a good overall picture of the system when it is idle and when the load is average, high, or maximum. When you encounter a performance problem first determine what changed since the last time that the system performed satisfactorily. In particular, consider whether the following factors apply:

- Workload or queries are different.
- More data is being processed.
- More connections exist.
- The software version was changed.
- The DB2 system was upgraded, which might have included the installation of fix packs.
- Data nodes were added.
- Data was redistributed.
- The database design was changed, for example, an MQT or index was added.
- The `RUNSTATS` command was issued with different parameters, for example, executed with sampling.

If you have no initial hypothesis for a performance problem, first determine whether there are any bottlenecks on the operating system side. Areas of bottlenecks include the CPU, main memory usage and swapping, the network, and I/O (disk usage). Based on the information that you gather, you can then drill down to the corresponding DB2 measures. This approach is described in the section “Some typical performance scenarios.”

Main performance measures and how to monitor them

To monitor data warehouse performance, you must look at both operating system and database performance metrics. Another good source for learning about monitoring is the best practices paper *Tuning and Monitoring Database System Performance*. It explains DB2 monitoring and tuning in OLTP environments.

Operating system performance measures

The interactive tool that is shown in this paper is the `nmon` utility. If you want to monitor a cluster, you must open a window (for example, with PuTTY software) on each server and start the `nmon` utility there.

The command-line tools that are shown are tailored for specific measurements. Some tools behave slightly differently on Linux operating systems than they do on AIX operating systems.

Monitor all servers of a cluster to detect anomalies or deviations that might impact the overall performance. To run a command-line tool through a single invocation on all servers of a cluster, use the `rah` command.

CPU usage

When considering CPU load, differentiate between these types of CPU measurements:

- User CPU: The number of CPU cycles that are required for user processes
- System CPU: The number of CPU cycles that are used by the operating system kernel, for example, for paging or process switching
- I/O wait: The percentage of CPU cycles that is spent waiting for data
- Idle time: The number of cycles where there is nothing for the CPU to do

The sum of user and system CPU usage can be close to 100% during peak times. A 4:1 ratio of user CPU usage to system CPU usage is considered normal. If the system CPU usage is increasing for time frames that are longer than the average execution time of your SQL, query check the system and identify the cause of this imbalance.

For the I/O wait measurement, values of up to 25% are normal. Peak values of greater than 25% for longer time frames suggest that disk activity is too high and need investigation.

For multi-core and multithreaded CPUs, in addition to checking the overall CPU usage, check the usage at the core or thread level to determine core-related or thread-related bottlenecks. For example, an overall CPU usage of 8% looks low at first glance. However, there might be a CPU-related bottleneck if one thread is running at almost 100% CPU usage and the other 15 threads are idle.

To determine the load on a cluster, look at the length of the run queue and the number of process switches. These provide a good hint of how busy, in terms of parallel running jobs, the system is and can be.

Examine the length of the run queue and the number of process switches to determine whether they are the reason for high system CPU usage.

Monitoring CPU usage

You can use the `nmon` utility to monitor CPU usage. You must run it on each server in a cluster. If you run it in the c-interactive mode, it graphically shows the CPU usage.

For a system with a large number of CPUs, you can also use the `nmon` utility in l-interactive mode, which displays the average overall CPU usage over time. It is also useful to view the usage for the most active processes by using the t-option. These processes should be for an active `db2sysc` command on a DB2 system.

Figure 1 shows the overall CPU usage and the usage for the important processes for an IBM Smart Analytics System 7700 data module with 16 cores. The CPU usage of the processes is indicated per core. The value of 160% for the most active process means that this process can use 80% of two CPUs.

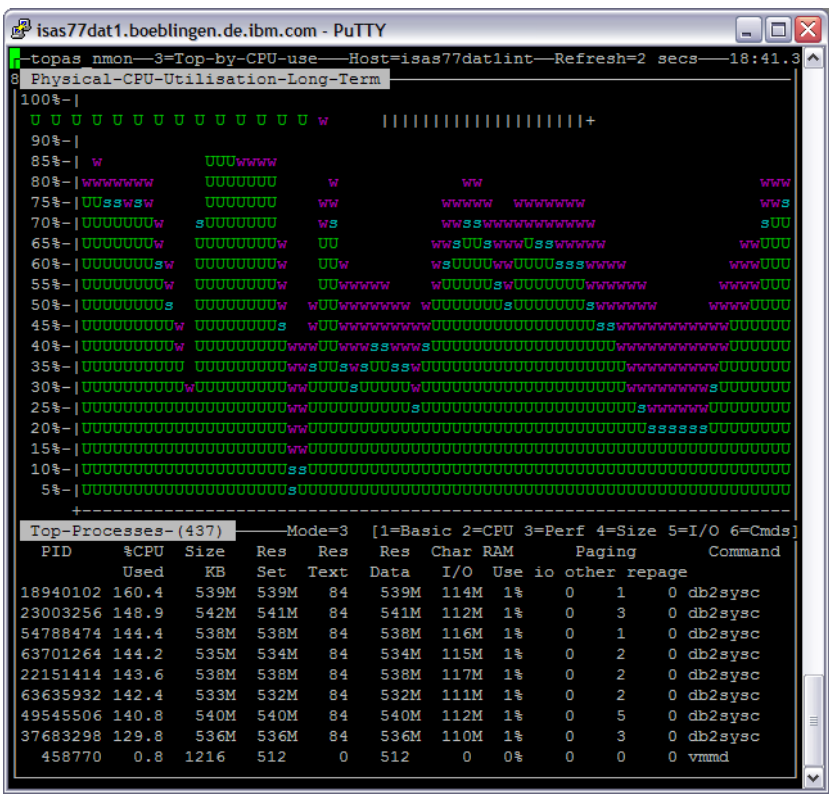


Figure 1. nmon utility running in l and t interactive modes

An alternative to the `nmon` utility is the `vmstat` command:

```
vmstat interval count
```

This `vmstat` command executes `vmstat <count>` times every `<interval>` seconds. ***Figure 2*** shows sample output from running "`vmstat 2 2`" on an AIX operating system where the user CPU load is approximately 90% - 89%:

```
System configuration: lcpu=64 mem=127744MB

kthr    memory              page              faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
173  0 25185655 7648547   0   0   0   0    0   0 19598 546686 229916 90  7  3  0
187  0 25187271 7646929   0   0   0   0    0   0 26439 660818 301124 89  7  4  0
```

Figure 2. Output of vmstat command on an AIX operating system

The columns under the cpu heading represent percentages of different types of CPU time:

- us: Time spent on user activities
- sy: Time spent on system activities
- id: Idle time
- wa: Wait time

For Linux operating systems, disregard the first output line because it contains these values for the time since system startup. The subsequent lines are for any corresponding intervals.

To get a brief impression of the CPU load of the past 15 minutes, you can use the `uptime` command:

```
rah uptime
```

Issuing the `rah` command from the administration node restricts the check to the DB2 related servers. The screen capture in ***Figure 3*** shows sample `uptime` command output, including the number of users and the average CPU usage for 1, 5, and 15 minutes. Using this information, you can decide whether a high load is only a short peak event or whether it lasts for a longer time.

```
  10:28AM   up 84 days,  22:55,   7 users,  load average: 41.74, 22.38, 13.87
isas77adm1int: uptime completed ok

  10:28AM   up 84 days,  10:53,   0 users,  load average: 0.46, 0.24, 0.15
isas77dat1int: uptime completed ok

  10:28AM   up 84 days,  22:55,   0 users,  load average: 0.02, 0.09, 0.13
isas77dat2int: uptime completed ok
```

Figure 3. rah uptime command output for a cluster with CPU load on the administration node

You can show the size of the run queue and the number of process switches by using the `nmon` utility in the k-interactive mode. ***Figure 4*** shows an example of an `nmon` kernel window with size of the run queue and the number of process switches in the left upper corner:

```
 Kernel
RunQueue=    351.9 | swapIn =       0.0 | Directory Search | Kernel Processes
pswitch =  95059.8 | syscall= 321417.8 | iget   =       0.0 | ksched=      0.0
fork    =    184.4 | read   =   1699.0 | dirblk=       0.0 | koverf=      0.0
exec    =      0.0 | write  =    923.2 | namei =  30742.0 | kexit =      0.0
msg     =   5040.6 | readch =     85020722.9           | Load Averages
sem     =     77.0 | writech=          635.8           | 1 min =    173.87
HW Intrp=  34339.0 | R+W(MB/s)=         10.1           | 5 min =     78.56
SW Intrp=     50.0 | Up Time=211.1 days (max=497)      | 15 min=     32.04
```

Figure 4. nmon kernel window

The size of the run queue also appears in the first column of the `vmstat` command output, labeled with r. The number of process switches is shown in the `vmstat` command output in the cs (context switches) column. For instance, in ***Figure 2***, the values for the run queue are 173 and 187, and for those processes, where the numbers of context switches are 229916 and 301124.

I/O

Disk I/O is the slowest way to deal with data. OLAP systems always show a high amount of I/O. For these reasons, it is important to optimize data transfers and to identify and minimize I/O waits. Key measures to monitor for I/O are the amount of data read/written per second, the number of I/O operations per second, and the degree of utilization.

You should know the performance characteristics of your storage system. To determine the maximum possible I/O performance, use the `dd` command on AIX operating systems and the `sg_dd` command on Linux operating systems to generate a sample I/O work load containing queries that reads from and writes to raw physical devices.

For example, for an IBM Smart Analytics System 7700 cluster, the following command reads from the hdisk device 19 blocks of size 512 KB, a total of 32,768 times:

```
dd if=/dev/rhdisk19 of=/dev/null bs=512k count=32768
```

If you run this command a number of times in parallel (up to 12 times should be sufficient), you can determine the maximum sequential read performance of a physical device. For a 7700 device, the performance is approximately 750 MB per second and 1500 transactions per second on average. For a 5600 R1 cluster, you can use the similar `sg_dd` command to check for the maximum sequential read capacity:

```
sg_dd odir=1 dio=1 of=/dev/null bs=512k count=100000000 if=/dev/sdb
```

Because the 5600 R1 system uses less than half the number of disks per RAID6 device that the 7700 does, expected throughput is approximately 300 MB per second and approximately 600 transactions per second.

Monitoring I/O

You can use the `iostat` command to measure each I/O device, as shown:

```
iostat interval count
```

This call initiates `iostat <count>` times every `<interval>` seconds. If you specify the -k option on a Linux operating system, the output includes the translations per second and KB that are read per second for each disk or device for the specified interval. For Linux operating systems, the command behaves similarly to the `vmstat` command because the first execution returns these values for the period since the last system reboot. For AIX operating systems, issuing the `iostat` command with the `-f` or `-F` option displays the I/O activity per file system. With the `nmon` utility, you can monitor the I/O activity by using the d-interactive mode.

Network traffic

The most important network in an IBM Smart Analytics System is the internal application network – also know as the fast communication manager (FCM) – that is used by DB2 to exchange results between the administration and data nodes or to exchange data between data nodes.

Bottlenecks can appear in the communication between the administration node and the data nodes or when the volume of messages overall is driving the network to its capacity limit. This capacity limit is determined by the bandwidth of the network (usually 1 Gb/second or 10 Gb/second) and the number of switches. For a 10 Gb network with two switches, the capacity is 2 x 10 Gbit/second, which is 2.5 GB/second. Although this is the theoretical limit, the realistically achievable maximum throughput for the IBM Smart Analytics System 7700 product is 80% - 90% of this value.

The most important measure is the amount of data that is sent and received by the server, which is usually measured in KB/second. Because all servers in a cluster share the internal application network, you must sum the measures for the servers to determine whether the capacity limit has been reached.

If the network performance is lower than you expect, look at the number of sent and received packages and the number of transmission errors. These errors should be less than 1% of the number of received and sent packages.

Monitoring network traffic

To monitor the network traffic for one server in a cluster, use the `nmon` utility. The following sample output was generated by specifying the interactive `n` option:

```
-topas_nmon--b=Black&White--------Host=isas77dat1int--Refresh=2 secs----09:12.12-
 Network ---------------------------------------------------------------------------
I/F Name Recv=KB/s Trans=KB/s packin packout insize outsize Peak->Recv TransKB
   en12        0.1        0.1      1.5      0.5   79.7  266.0        0.3      0.4
   en11    72657.9    76765.4  48724.0  48711.5 1527.0 1613.7    72657.9  76765.4
    lo0        0.0        0.0      0.0      0.0    0.0    0.0        0.2      0.2
  Total       71.0       75.0 in Mbytes/second    Overflow=0
```
Figure 5. Part of the nmon command output, showing network traffic statistics

In Figure 5, `en11` is the network interface of the internal application network of the IBM Smart Analytics System 7700 product. The output shows that 72 KB received and 76 MB sent represent the highest activity level.

To check whether any transmission errors occurred, use the `netstat` command:

`netstat –I` *`interface name time interval`*	(for AIX operating systems)
`netstat –I` *`interface name`* `–c`	(for Linux operating systems)

This command displays the number of packages that were sent and received and the associated number of errors for *interface name* every *time interval* seconds. The first set of lines in the output shows the summary of the measures since the network was last started.

Main memory

On both Linux and AIX systems, the main memory is used for program execution and for file caching. If the system runs out of main memory space, it must transfer some of the data out of memory and onto disk. This disk space is called *paging space*. From a monitoring perspective, the measure of paging space usage is the most important. Whereas the usage of real memory can grow close to 100% without any negative impact, increased paging means increased system CPU and I/O activity, resulting in an overall decrease in system performance.

As much as possible, avoid paging. Short periods of small paging space usage (where the size of the paging space is less than 5% - 10 % of physical main memory size) can be tolerated. However, larger paging space usage can lead to a severe system slowdown, so that simple DB2 software or operating system commands seem to hang.

Monitoring main memory

To monitor the amount of memory swapping, use the `vmstat` command. On AIX operating systems, where a unit of memory is a 4 kB page, see the page section of the output, columns pi and po. On Linux operating systems, where the unit is 1 kB, see the swap section of the output, columns si and so. ***Figure*** 26 shows sample output of the `nmon` command on an AIX operating system:

```
-topas nmon—C=many-CPUs————————Host=isas77datlint—Refresh=2 secs———16:51.23———
 Memory
               Physical  PageSpace |        pages/sec  In      Out  | FileSystemCache
% Used          63.6%       1.9%   | to Paging Space   0.0     0.0  | (numperm)   0.9%
% Free          36.4%      98.1%   | to File System   16.0    20.0  | Process    43.6%
GB Used         79.4GB      2.4GB  | Page Scans        0.0          | System     19.1%
GB Free         45.4GB    124.6GB  | Page Cycles       0.0          | Free       36.4%
Total(GB)      124.8GB    127.0GB  | Page Steals       0.0          |            ------
                                   | Page Faults   55657.6          | Total     100.0%
------------------------------------------------------------        | numclient   0.9%
Min/Maxperm      3718MB(  3%)  111544MB( 87%) <--% of RAM           | maxclient  87.3%
Min/Maxfree      960    1088       Total Virtual   251.8GB          | User       41.6%
Min/Maxpgahead     2       8    Accessed Virtual    78.9GB 31.4%|    | Pinned     20.5%
```

Figure 6. nmon command output showing memory utilization

You can run the `nmon` command in m-interactive mode to monitor the memory utilization, including swapping memory of a single server in a cluster. In the output, check the numbers in the PageSpace and pages per second columns. ***Figure*** 6 shows that a bit more than a third of main memory is still free and that no paging is taking place.

Database-related performance measures

Starting with DB2 Version 9.7, monitoring table functions are provided to retrieve monitoring information from in-memory locations (in-memory metrics). Use these monitoring functions if possible, because they have lower memory usage than the older snapshot-based functions.

For monitoring DB2 performance measures, use the table functions that are based on the in-memory metrics. To monitor running queries, use the `MON_GET_UNIT_OF_WORK` and `MON_GET_ACTIVITY_DETAILS` functions. If a query terminated and the connection is still open, the `MON_GET_CONNECTION` function returns the aggregated performance measurements for all activities that ran for that connection.

To determine the aggregated measures of activities by service subclass or workload over a longer time frame, you can use the `MON_GET_SERVICE_SUBCLASS` and `MON_GET_WORKLOAD` functions. These functions and workload management in a data warehouse environment in general are discussed in detail in the best practices paper *Implementing DB2 workload management in a data warehouse*.

All of these monitoring functions have essentially the same monitor elements and determine essentially the same measures. The only difference is their scope: activity, unit of work, connection, service subclass, or workload.

Determine the measures separately for each DB2 partition (referred to as *member* by these functions) by specifying `-2` for the second parameter of these functions so that all database members are considered. Then, aggregate the measures for the whole system. In this way, you can collect all the information that you need to get a system-wide overview of DB2 performance, but you can also locate performance problems that are restricted to certain partitions.

You can find detailed descriptions of these monitoring functions in the following DB2 Information Center topic:
http://publib.boulder.ibm.com/infocenter/db2luw/v10r5/topic/com.ibm.db2.luw.sql.rtn.doc/doc/c0053963.html.

Two additional functions were introduced in DB2 Version 10: `MON_SAMPLE_WORKLOAD_METRICS` and `MON_SAMPLE_SERVICE_CLASS_METRICS`. You can use these functions to retrieve selected measurements for a particular time period after the invocation. In this way, you can retrieve the current performance measurements at the workload and service class levels.

If you prefer an interactive tool, you can use the `db2top` command for retrieving measurements. Because it is based on snapshots, its use has a higher impact on performance. Link:
http://publib.boulder.ibm.com/infocenter/db2luw/v10r5/topic/com.ibm.db2.luw.admin.mon.doc/doc/t0025223.html.

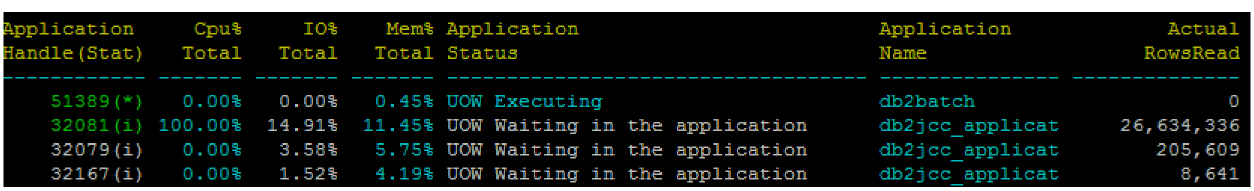
CPU time and usage

First, determine the overall CPU time that has been needed for the execution of a statement so far. You can contrast this time with the elapsed time to determine the CPU utilization. Then, you can drill down to determine how much time is spent on the various activities, such as sorting, I/O, and FCM traffic flow.

For SQL queries, review wait times, such as lock wait time, I/O wait time, and FCM wait time. These wait times should be as low as possible, with one exception: for the coordinator partition, which wait on the results of the data partitions, because they handle almost all of the query execution in a well-partitioned database setup.

Monitoring CPU time and usage

To determine the relative share that each DB2 application uses of the total CPU usage of all active DB2 applications, use the `db2top` command. This share is displayed in the session view under the column CPU% Total, as shown in ***Figure***. You can obtain this information by specifying the l interactive command:



Application Handle(Stat)	Cpu% Total	IO% Total	Mem% Total	Application Status	Application Name	Actual RowsRead
51389(*)	0.00%	0.00%	0.45%	UOW Executing	db2batch	0
32081(i)	100.00%	14.91%	11.45%	UOW Waiting in the application	db2jcc_applicat	26,634,336
32079(i)	0.00%	3.58%	5.75%	UOW Waiting in the application	db2jcc_applicat	205,609
32167(i)	0.00%	1.52%	4.19%	UOW Waiting in the application	db2jcc_applicat	8,641

Figure 7. db2top command view showing relative CPU share

The monitoring table functions provide a comprehensive set of monitor elements for determining the execution time characteristics. The most important monitor elements are as follows:

- `total_cpu_time`: The total user and system CPU time that the DB2 product uses, in microseconds.
- `total_rqst_time`: The total time that is spent working on requests, in milliseconds. The requests are for all types of DB2 activities, such as the execution of SQL statements, loads, execution of stored procedures, RUNSTATS, commits, and statement compilation.
- `total_wait_time`: Total time that the DB2 product spends waiting, in milliseconds.
- `total_section_time`: The sum of the time that is spent executing sections of compiled SQL statements, with wait times, in milliseconds.
- `total_section_proc_time`: The total time that is spent executing sections of compiled SQL statements, without wait times, in milliseconds.

The values of these measures are computed by summing the corresponding values of the DB2 threads. If the threads running on a partition overlap, these values can be greater than the overall elapsed execution time. For example, this can happen for the `TOTAL_SECTION_TIME` function if sections of an SQL statement are at least partially executed in parallel.

To determine the overall elapsed execution time of DB2 requests, use the `total_app_rqst_time` monitor element. It returns, for the coordinator partition, the total elapsed time that is spent on application requests, in milliseconds. For other partitions, the monitor element returns 0.

The following query calculates, for each active application, the average values of measurements for the data partitions. It uses the `total_app_rqst_time` element for determining both the user and system CPU utilization. The value of the `total_app_rqst_time` element is updated less frequently than the value of the other measures, the value for CPU utilization can be temporarily too high. It is accurate, however, at the end of the execution of an activity.

```
with connect_agg as (
SELECT t.APPLICATION_HANDLE,
       count(t.member) as NUM_MEMBER,
       decimal(avg(t.TOTAL_CPU_TIME*1.00)/1000000, 12,2) as AVG_TOTAL_CPU_TIME_SEC,
       decimal(avg(t.TOTAL_WAIT_TIME*1.00)/1000, 12,2) as AVG_TOTAL_WAIT_TIME_SEC,
       decimal(avg(t.TOTAL_RQST_TIME*1.00)/1000, 12,2) as AVG_TOTAL_RQST_TIME_SEC,
       decimal(avg(t.TOTAL_SECTION_TIME*1.00)/1000,12,2) as AVG_TOTAL_SECTION_TIME_SEC,
       decimal(avg(t.TOTAL_SECTION_PROC_TIME*1.00)/1000, 12,2) as AVG_TOTAL_SECTION_PROC_TIME_SEC
FROM TABLE(MON_GET_CONNECTION(cast(NULL as bigint), -2)) AS t
WHERE t.MEMBER > 0
GROUP BY t.APPLICATION_HANDLE
)
select c.APPLICATION_HANDLE, NUM_MEMBER,
       AVG_TOTAL_CPU_TIME_SEC, AVG_TOTAL_WAIT_TIME_SEC, AVG_TOTAL_RQST_TIME_SEC,
       AVG_TOTAL_SECTION_TIME_SEC, AVG_TOTAL_SECTION_PROC_TIME_SEC,
       decimal(t0.TOTAL_APP_RQST_TIME/1000.0, 12,2) as TOTAL_APP_RQST_TIME_SEC,
       case when t0.TOTAL_APP_RQST_TIME > 0  then
decimal(AVG_TOTAL_CPU_TIME_SEC*100000/(t0.TOTAL_APP_RQST_TIME*<nbCoresPerPartition>), 12, 2) end
as CPU_USAGE_PCT
FROM connect_agg c
inner join TABLE(MON_GET_CONNECTION(cast(NULL as bigint), -2)) AS t0 on (c.APPLICATION_HANDLE =
t0.APPLICATION_HANDLE)
where t0.MEMBER = = t0.COORD_MEMBER;
```

For the computation of the CPU usage, consider the number of cores per partition. For an IBM Smart Analytics System 7700, which has two cores per data partition, replace *nbCoresperPartition* in the previous SQL query with 2.

There are other time-related measures, for example, those for the different wait times (such as lock wait time), I/O times, and the time that is required for FCM traffic and sorting. The DB2 Information Centre topic http://publib.boulder.ibm.com/infocenter/db2luw/v10r5/index.jsp?topic=/com.ibm.db2.luw.admin.mon.doc/doc/c0055434.html gives a good overview of the relationships between all these time metrics.

I/O-related performance measures

Buffer pools

The most important measure for buffer pools is the hit ratio. The hit ratio is the percentage of logical reads (and not physical reads), that is, they are read directly from the buffer pool.

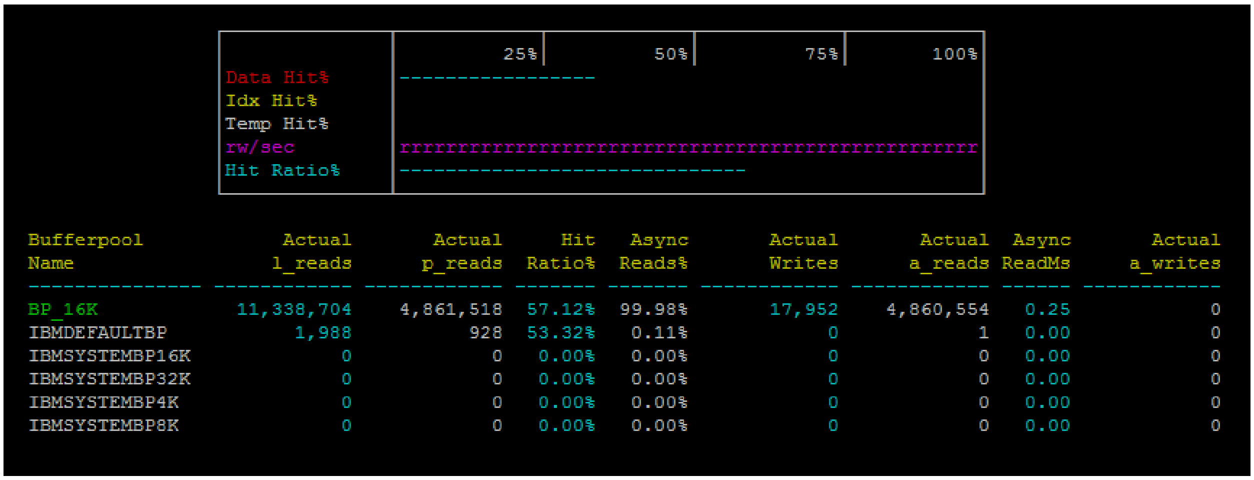
In OLTP systems, the buffer pool hit ratio should be close to 100%. However, it is not possible to define a concrete target for data warehouse systems, because the amount of data that is processed by a query can vary considerably. For example, the scan of a huge table might result in a very low hit ratio. In general, the higher the hit ratio for a query, the greater the amount of data that is processed.

Other measurements to look at are the number of logical and physical reads and the number of writes, which represent the amount of activity a buffer pool.

On an IBM Smart Analytics System, if the number of reads from the IBMDEFAULTBP buffer pool is not small, determine whether a table space was created accidentally for user data within that buffer pool.

Monitoring buffer pools

In the buffer pool view of the `db2top` command (interactive command option `b`), you can check these measures for each buffer pool. ***Figure 8***8 shows an example of the output:



Bufferpool Name	Actual l_reads	Actual p_reads	Hit Ratio%	Async Reads%	Actual Writes	Actual a_reads	Async ReadMs	Actual a_writes
BP_16K	11,338,704	4,861,518	57.12%	99.98%	17,952	4,860,554	0.25	0
IBMDEFAULTBP	1,988	928	53.32%	0.11%	0	1	0.00	0
IBMSYSTEMBP16K	0	0	0.00%	0.00%	0	0	0.00	0
IBMSYSTEMBP32K	0	0	0.00%	0.00%	0	0	0.00	0
IBMSYSTEMBP4K	0	0	0.00%	0.00%	0	0	0.00	0
IBMSYSTEMBP8K	0	0	0.00%	0.00%	0	0	0.00	0

Figure 8. Buffer pool view of the db2top command

To determine these values for each buffer pool, you can use the `MON_GET_BUFFERPOOL` monitoring table function. The following query was adapted from the query in the DB2 Information Centre topic at http://publib.boulder.ibm.com/infocenter/db2luw/v10r5/topic/com.ibm.db2.luw.sql.rtn.doc/doc/r0053942.html. This query calculates the hit ratio from the number of logical and physical reads for each buffer pool.

```
WITH bp_metrics AS (
  SELECT bp_name,
       sum( pool_data_l_reads + pool_temp_data_l_reads +
          pool_index_l_reads + pool_temp_index_l_reads +
          pool_xda_l_reads + pool_temp_xda_l_reads) as logical_reads,
       sum( pool_data_p_reads + pool_temp_data_p_reads +
          pool_index_p_reads + pool_temp_index_p_reads +
          pool_xda_p_reads + pool_temp_xda_p_reads) as physical_reads
  FROM TABLE(MON_GET_BUFFERPOOL('',-2)) AS METRICS
  GROUP BY bp_name
```

```
)
SELECT
 VARCHAR(bp_name,20) AS bp_name,
 logical_reads,
 physical_reads,
 CASE WHEN logical_reads > 0
  THEN DEC((1 - (FLOAT(physical_reads) / FLOAT(logical_reads))) * 100,5,2)
  ELSE NULL
 END AS HIT_RATIO
FROM bp_metrics;
```

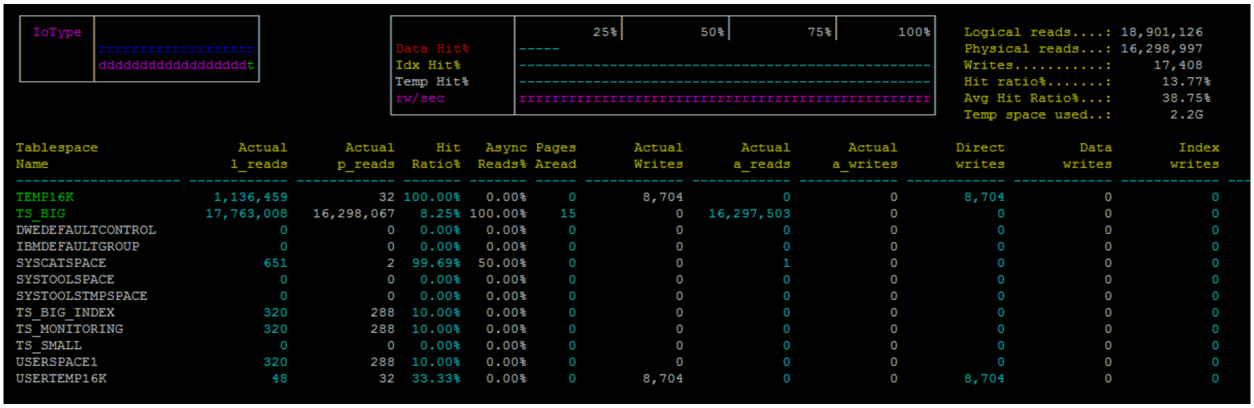
Table spaces

Table space monitoring drills one level deeper than buffer pool monitoring. For an Smart Analytics System, the table spaces of interest are TS_SMALL, TS_BIG, TS_BIG_INDEX, and the table spaces for temporary data (TEMP16K and USERTEMP16K). All of these table spaces are associated with the buffer pool BP_16K. As with buffer pools, an important measurement for each table space is the hit ratio.

Compare the number of asynchronous read and writes with the total number of physical reads and writes to see the effectiveness of the prefetchers. If the percentage of asynchronous reads and writes is considerably below 80%, increase the value of the `num_ioservers` database configuration parameter.

Monitoring table spaces

Using the `db2top` command, you can get to the table space information with the `t` interactive option. ***Figure 9*** shows sample table space measurements. For the TS_BIG table space, the buffer pool hit ratio is quite low, but the asynchronous read ratio is very good.



Tablespace Name	Actual l_reads	Actual p_reads	Hit Ratio%	Async Reads%	Pages Aread	Actual Writes	Actual a_reads	Actual a_writes	Direct writes	Data writes	Index writes
TEMP16K	1,136,459	32	100.00%	0.00%	0	8,704	0	0	8,704	0	0
TS_BIG	17,763,008	16,298,067	8.25%	100.00%	15	0	16,297,503	0	0	0	0
DWEDEFAULTCONTROL	0	0	0.00%	0.00%	0	0	0	0	0	0	0
IBMDEFAULTGROUP	0	0	0.00%	0.00%	0	0	0	0	0	0	0
SYSCATSPACE	651	2	99.69%	50.00%	0	0	1	0	0	0	0
SYSTOOLSPACE	0	0	0.00%	0.00%	0	0	0	0	0	0	0
SYSTOOLSTMPSPACE	0	0	0.00%	0.00%	0	0	0	0	0	0	0
TS_BIG_INDEX	320	288	10.00%	0.00%	0	0	0	0	0	0	0
TS_MONITORING	320	288	10.00%	0.00%	0	0	0	0	0	0	0
TS_SMALL	0	0	0.00%	0.00%	0	0	0	0	0	0	0
USERSPACE1	320	288	10.00%	0.00%	0	0	0	0	0	0	0
USERTEMP16K	48	32	33.33%	0.00%	0	8,704	0	0	8,704	0	0

Figure 9. Table space monitoring with the db2top command

You can retrieve these and more measurements with the `MON_GET_TABLESPACE` table function. The following query retrieves the buffer pool hit ratio and the ratios for asynchronous buffer pool read and write operations:

```
with tbsp_metrics as (
SELECT  varchar(tbsp_name, 20) as tbsp_name,
        tbsp_type,
        max (tbsp_page_size) as tbsp_page_size,
        count(member) as num_member,
        sum( pool_data_l_reads + pool_temp_data_l_reads + pool_index_l_reads + pool_temp_index_l_reads +
pool_xda_l_reads + pool_temp_xda_l_reads) as sum_logical_reads,
        sum( pool_data_p_reads + pool_temp_data_p_reads + pool_index_p_reads + pool_temp_index_p_reads +
pool_xda_p_reads + pool_temp_xda_p_reads) as sum_physical_reads,
        sum( pool_data_p_reads + pool_index_p_reads) as sum_data_index_p_reads,
        sum( pool_async_data_reads + pool_async_index_reads) as sum_async_data_index_reads,
        sum(pool_data_writes) as sum_pool_data_writes,
        sum(pool_async_data_writes) as sum_pool_async_data_writes
FROM TABLE(MON_GET_TABLESPACE('',-2)) AS t
group by tbsp_name, tbsp_type
)
select  tbsp_name, tbsp_type, tbsp_page_size, num_member,
        sum_logical_reads, sum_physical_reads,
        case when sum_logical_reads > 0 then decimal((1 - float(sum_physical_reads)/float(sum_logical_reads)) * 100.0,
5,2) else null end as bp_hit_ratio,
        sum_data_index_p_reads, sum_async_data_index_reads,
        case when sum_data_index_p_reads > 0 then
decimal(float(sum_async_data_index_reads)*100/float(sum_data_index_p_reads),5,2) else null end as
async_data_index_read_ratio,
        sum_pool_data_writes, sum_pool_async_data_writes,
        case when sum_pool_data_writes > 0 then
decimal(float(sum_pool_async_data_writes)*100/float(sum_pool_data_writes),5,2) else null end as
async_data_write_ratio
from tbsp_metrics; _
```

Memory usage

The DB2 product should use only as much memory as fits into main memory.

Always monitor DB2 memory usage with operating system memory usage to ensure that the DB2 product does not exhaust the main memory resources.

The most important DB2 memory consumers are as follows:

- FCMBP: The shared memory for the FCM, which is controlled by the `fcm_num_buffers` database manager configuration parameter. The DB2 product attempts to determine an optimal value, so you do not have to change this setting.

- Utility: Memory that is used by utilities such as RUNSTATS, LOAD, and BACKUP, whose upper limit per database is specified by the value of the util_heap_sz database configuration parameter.

- BufferPool: The size of the buffer pools.

- SortHeap: The current total size of the sort heap that are used for sorts and hash joins. Because each application allocates its own sort heap, the total size can vary considerably over time.

Monitoring memory usage

You can monitor DB2 memory usage by using the DB2 memory information of the db2top command, which you can get with the “m” interactive command, as shown in the following example:

Memory Type	Level	Memory Pool	Percent Total	Current Size	High WaterMark	Percent Max	Maximum Size	# of Pool(s)
Instance	BCUAIX	Monitor	0.00%	5.3M	5.3M	83.33%	6.3M	17
Instance	BCUAIX	FCMBP	4.72%	5.2G	8.8G	100.00%	5.2G	17
Instance	BCUAIX	Other	2.53%	2.8G	2.9G	87.96%	3.2G	17
Database	BCUDB	Applications	0.02%	21.7M	22.0M	14.60%	149.0M	298
Database	BCUDB	Database	0.64%	726.6M	726.6M	48.67%	1.4G	17
Database	BCUDB	Lock Mgr	1.05%	1.1G	1.1G	99.91%	1.1G	17
Database	BCUDB	Utility	0.00%	1.0M	1.0M	0.02%	4.2G	17
Database	BCUDB	Package Cache	0.00%	5.6M	5.6M	26.47%	21.2M	17
Database	BCUDB	Catalog Cache	0.00%	3.1M	3.1M	23.08%	13.8M	17
Database	BCUDB	Other	0.00%	3.1M	3.1M	0.94%	340.0M	17
Database	BCUDB	BufferPool	70.86%	78.9G	78.9G	100.00%	78.9G	102
Database	BCUDB	ApplShrHeap	0.01%	8.8M	8.8M	0.67%	1.2G	17
Application	BCUDB	Applications	0.02%	21.7M	22.0M	14.60%	149.0M	298
Application	BCUDB	Other	0.04%	44.3M	44.6M	0.00%	28.3T	233
Application	BCUDB	SortHeap	20.09%	22.3G	22.3G	0.56%	3.8T	32

Figure 10. DB2 memory view of the db2top command

To determine memory usage across data modules, sum the values for the biggest memory consumers, and compare the result with the available main memory. The db2top command default screen in Figure 10 shows the aggregated memory usage for all database partitions. To get an impression of the memory usage of the IBM Smart Analytics System data modules, you can sum the values for the biggest memory consumers and compare the result with the available main memory in all data modules. In the sample db2top command output in ***Figure*** the most important memory consumers, which are FCMBP, (Instance), Other, Bufferpool, and SortHeap, are using approximately 106.5 GB of main memory. The two data modules together are using 256 GB of RAM on the IBM Smart Analytics System 7700. This shows that main memory is not a critical resource.

To check whether main memory is sufficient even when all memory pools have reached the maximum specified size, you can sum the values for the high watermark.

The db2top command shows the database and application memory values for only one database. To get a complete picture of the memory consumption, you must run the db2top command for each active database.

With DB2 V9.7 Fix Pack 5 and later, you can also use the MON_GET_MEMORY_POOL monitoring table function to compute values for all databases together with the total memory usage of DB2 instance per host name:

```
with mon_mem as (
SELECT  varchar(host_name, 20) AS host_name,
        varchar(memory_set_type, 20) AS set_type,
        varchar(memory_pool_type,20) AS pool_type,
        varchar(db_name, 20) AS db_name,
        memory_pool_used,
        memory_pool_used_hwm
FROM TABLE(
       MON_GET_MEMORY_POOL(NULL, CURRENT_SERVER, -2))
)
select  host_name, set_type, pool_type, db_name,
        sum(memory_pool_used)/1000000 as memory_pool_used_MB,
        sum(memory_pool_used_hwm)/1000000 as memory_pool_used_hwm_MB
from mon_mem
-- group by set_type, pool_type, dbname
group by grouping sets ((set_type, pool_type, db_name), (host_name));
```

Locking

Locking monitoring is important for assuring optimal concurrency and for detecting and minimizing lock wait and deadlock situations. In particular, you should monitor the following items:

- The number of lock waits and the lock wait times
- The number of deadlocks
- The number of lock escalations.

In the case of locks, you should determine the blocking application and blocked application, together with the responsible table. Values for measurements should be as low as possible.

Using the `db2top` command, you can monitor these measures at the database level through the database view (the `d` interactive option). It gives you an overview of the whole database. For monitoring locking, check the values for lock waits, deadlocks, and lock escalation, as shown in the sample in Figure 11:

```
                              Database

          25%|        50%|        75%|      100%|
MaxActSess  -
AssAgents   ---------------------------------------------------
SortMemory
LogUsed     ----
FCM BufLow  ---

Start Date Start Time     Status    Shthres    Buffers     FCMBuf   OtherMem
2012/11/08   09:56:23     Active      90.7G      78.9G       5.2G       4.7G

  Sessions     ActSess   LockUsed LockEscals  Deadlocks   LogReads  LogWrites
       201           1         0%         64          0          0          0

   L_Reads     P_Reads   HitRatio    A_Reads     Writes   A_Writes  Lock Wait
         0           0      0.00%      0.00%          0          0         16

  Sortheap     SortOvf PctSortOvf AvgPRdTime AvgDRdTime AvgPWrTime AvgDWrTime
         0           0      0.00%       0.00       0.00       0.00       0.00
```

Figure 11. Database view of db2top command

You can see details about the lock status of each application in the Locks view (interactive option U) of the `db2top` command, as shown in the sample output in Figure 12. It shows, for each application, the locked object, lock mode, locks status, and lock count and shows whether the application is blocking another application.

```
      Locks held.......:         94 [0.00]
      Agents waiting...:         16
      Appls Connected..:         36

Agent        Application          Application                         Object                                   Lock     Object     Lock       Lock     Is
Id(State)    Name                 Status                              Name                                     Mode     Type       Status     Count    Blocker
------------ -------------------- ----------------------------------- ---------------------------------------- -------- ---------- ---------- -------- -------
    34288(*) db2bp                UOW Executing                       Internal Variation                       S        Variation  Granted          17     Yes
    34288(*) db2bp                UOW Executing                       Internal Plan                            S        Plan       Granted          17     Yes
    34288(*) db2bp                UOW Executing                       STAGE.PRCHS_PRFL_DETAIL                  IS       Table      Granted          10     Yes
    34288(*) db2bp                UOW Executing                       MART.PRCHS_PRFL_DETAIL                   X        Table      Granted          16     Yes
    34304(l) db2bp                Lock Waiting                        Internal Variation                       S        Variation  Granted          17      No
    34304(l) db2bp                Lock Waiting                        Internal Plan                            S        Plan       Granted          17      No
    34304(l) db2bp                Lock Waiting                        MART.PRCHS_PRFL_DETAIL[0]                IS [X]   Table      -                 -      No
```

Figure 12. Locks view of db2top command

In Figure 13, the Lock chain view (interactive option L) shows the blocked application with the blocking agents:

```
Blocked/Blocking Agent Chain
-------------------------------
34288->34304
```

Figure 13. Lock chain view of db2top command

The `MON_GET_UNIT_OF_WORK` and `MON_GET_CONNECTION` monitoring table functions provide an overview of the locking situation. The following monitor elements are of interest:

- `lock waits`: The total number of lock waits
- `lock_wait_time`: The total wait time due to locks, in milliseconds
- `lock_timeouts`: The number of lock timeouts

- `lock_escals`: The total number of lock escalations
- `deadlocks`: The total number of deadlocks

The following query retrieves these measures for each application:

```
select  APPLICATION_HANDLE,
        count(MEMBER) as NUM_MEMBER,
        max(connection_start_time) as connection_start_time,
        sum(LOCK_ESCALS) as LOCK_ESCALS,
        sum(LOCK_TIMEOUTS) as LOCK_TIMEOUTS,
        integer(sum(LOCK_WAIT_TIME)/1000) as LOCK_WAIT_TIME_SEC,
        sum(LOCK_WAITS) as LOCK_WAITS,
        sum(DEADLOCKS) as DEADLOCKS
from TABLE(MON_GET_CONNECTION(cast(NULL as bigint), -2)) AS t
group by APPLICATION_HANDLE;
```

You can use the MON_LOCKWAITS administrative view to show details for the blocked applications. For example, the following query retrieves the application handle of the requester and holder of the lock together with the lock object type, lock mode, and name of the locked table:

```
select  REQ_APPLICATION_HANDLE, REQ_MEMBER,
        HLD_APPLICATION_HANDLE, HLD_MEMBER,
        LOCK_OBJECT_TYPE, LOCK_MODE,
        varchar(TABSCHEMA, 20) as TABSCHEMA,
        varchar(TABNAME, 20) as TABNAME
from SYSIBMADM.MON_LOCKWAITS ;
```

Sorts and hash joins

The DB2 product allocates a separate sort heap for every sort and hash join operation in main memory whose maximum size is controlled by the `SortHeap` database configuration parameter. If this space is not sufficient, a sort heap overflow occurs, and the sort “spills” into temporary tables in the system temporary table space.

Avoid sort heap overflows, because of their negative impact on performance. This effect is less important if the spilled data remains in the buffer pool, such that no I/O operations are triggered. The same is true if you define a system temporary table space on solid state disks (SSDs).

As the total size of the sort heaps in main memory grows with the number of running sorts, it can exceed the threshold that you specify by using the `sheapthres` database manager configuration parameter. Sorts that are requested after this are called *post-threshold sorts*.

Avoid post-threshold sorts because the main memory might be exhausted, and paging might start, slowing down the system.

Furthermore, it is worthwhile having a closer look at a sort if the sort time is a significant percentage (such as 20% - 25%) of the total execution time. Consider query optimization techniques, such as defining an index on the sort column to avoid the sort operation.

For hash joins, you should avoid hash loops because they cause a considerable slowdown. Hash loops are less likely for the standard IBM Smart Analytics System or IBM PureData System for Operational Analytics settings of 35,000 4 KB blocks for the `sortheap` database configuration parameter. However, hash join overflows are quite common in a data warehouse environment when two big tables are joined.

Monitoring sorts and hash joins

For monitoring sort behavior, you can use the database view of the `db2top` command, which you also use for identifying a lock situation (see ***Figure***). Look at the values for the Sortheap, SortOvf, and PctSortOvf. The PctSortOvf shows the percentage of the total number of sort overflows.

As shown in the sample in Figure 14, the upper part of the application view of the `db2top` command shows the number of sorts, the number of sort overflows, and the sort time. For hash joins, it shows the number together with the number of hash loops and hash join overflows.

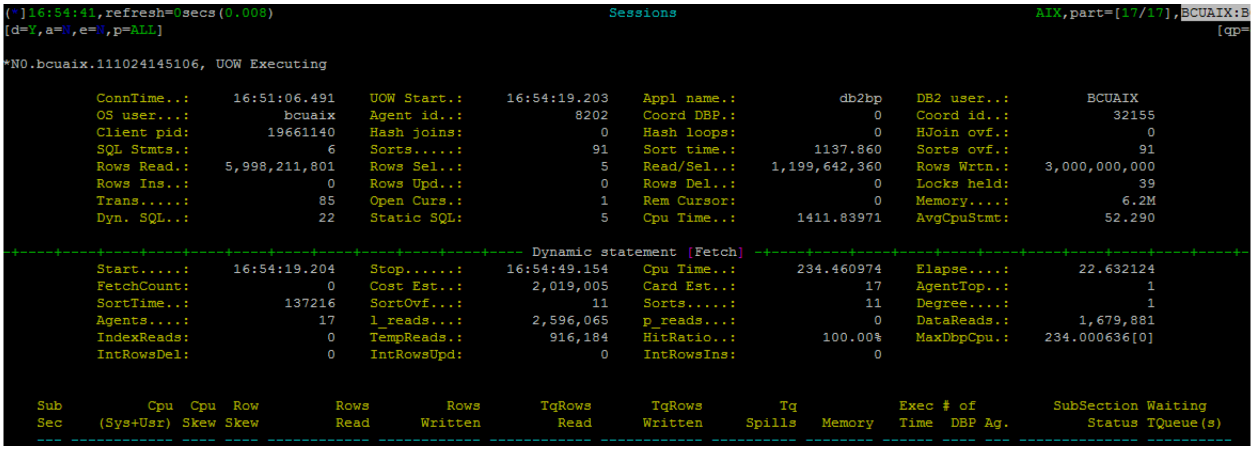


Figure 14. Upper part of application view of db2top command

You can also use the following monitor elements of the `MON_GET_CONNECTION` and `MON_GET_UNIT_OF_WORK` monitoring table functions:

- `total_sorts`: The total number of sorts.
- `total_section_sorts`: The total number of sorts that are executed by a compiled SQL statement.
- `sort_overflows`: The number of sort overflows.
- `post_threshold_sorts`: The number of post-threshold sorts, that is, those sorts that are requested after the total sort heap has exceeded the threshold that you specified by using the `sheapthres` database manager configuration parameter.
- `total_section_proc_time`: The total time, in milliseconds, that are spent processing a compiled SQL statement, without wait times. The value can exceed the total execution time for a member if intraparallelism is turned on.

- `total_section_sort_proc_time`: The total time, in milliseconds, for processing section sorts, without wait times.
- `rows modified`: The number of modified rows, which can be rows that are written to a temporary table from spilled sorts or hash join overflows.
- `pool_data_writes`: The number of pages that are written from the buffer pool to disk, which can be pages that are written to temporary table space containers. If this value is still 0 when sort or hash join overflows occur, the temporary tables for the spilled sorts can be kept in the buffer pool.
- `total_hash_joins`: The total number of hash joins
- `total_hash_loops`: The total number of hash loops
- `hash_join_overflows`: The total number of hash join overflows

The hash joins specific monitoring elements are included in MON_GET_CONNECTION and MON_GET_UNIT_OF_WORK starting with DB2 V10.5. For prior versions of DB2 you can use the snapshot-based monitoring table functions SNAP_GET_APPL or SNAP_GET_APPL_V95.

For each connected database application, the following query computes these values for sort and hash join overflows for the data partitions and aggregates them:

```
SELECT  APPLICATION_HANDLE,
        count(*) as NUM_MEMBER,
        sum(TOTAL_SORTS) as SUM_TOTAL_SORTS,
        sum(TOTAL_SECTION_SORTS) as SUM_TOTAL_SECTION_SORTS,
        sum(SORT_OVERFLOWS) as SUM_SORT_OVERFLOWS,
        sum(POST_THRESHOLD_SORTS) as SUM_POST_THRESHOLD_SORTS,
        decimal(avg(TOTAL_SECTION_PROC_TIME)/1000.0,8,2) as AVG_TOTAL_SECTION_PROC_TIME_SEC,
        decimal(avg(TOTAL_SECTION_SORT_PROC_TIME)/1000.0,8,2) as
AVG_TOTAL_SECTION_SORT_PROC_TIME_SEC,
        sum(TOTAL_HASH_JOINS) as SUM_TOTAL_HASH_JOINS,
        sum(TOTAL_HASH_LOOPS) as SUM_TOTAL_HASH_LOOPS,
        sum(HASH_JOIN_OVERFLOWS) as SUM_HASH_JOIN_OVERFLOWS,
        sum(ROWS_MODIFIED) as SUM_ROWS_MODIFIED,
        sum(POOL_DATA_WRITES) as SUM_POOL_DATA_WRITES
FROM TABLE(MON_GET_CONNECTION(cast(NULL as bigint), -2)) AS t
WHERE MEMBER > 0
GROUP BY APPLICATION_HANDLE;
```

Data skew

Data skew represents the uneven distribution of data across database partitions. Static data skew occurs when the data of a single table is not equally distributed. A dynamic data skew can occur during the execution of a query, for example, if two tables are joined or if a query predicate selects different percentages of data from the partitions.

Because the run time of a query is bounded by the "slowest" partition, the difference between the maximum and average execution times of the database partitions should be as small as possible. The following is the definition of skew, where *measure* can represent CPU time, the number of rows that are read, or the number of rows that are written:

1 - avg(*measure*) / max(*measure*)

This value should be as close to 0 as possible.

When monitoring CPU skew, calculate the acceleration of a query to estimate the performance improvement with an even distribution of the data. Given that the execution time of a query is bounded by the slowest performing partition (with maximum CPU time), it is defined as follows:

max(*CPU_time*) / avg(*CPU_time*)

Monitoring data skew

The skew view of the `db2top` command (by using the `J` option) shows the CPU time and number of rows that are read and written (per second and total) for each partition. As shown in Figure 15, you can get a similar view for a specific application by using the extended application view. To get this view, specify interactive command option `a` with an application ID, and then type option `X`.

Sub Sec	Node Nbr	Cpu (Sys+Usr)	Rows Read	Rows Written	TqRows Read
0	0	0.217	0	0	0
-1	1	2.569	8,236,703	10,669	0
1	2	2.617	8,538,502	11,358	0
1	3	2.599	8,297,018	10,821	0
1	4	2.133	6,996,010	7,675	0
1	5	2.677	8,576,765	11,441	0
1	6	2.125	7,166,533	8,082	0
1	7	2.655	8,483,934	11,292	0
1	8	2.473	8,211,514	10,603	0
1	9	2.404	7,816,758	9,878	0
1	10	2.101	7,189,326	8,150	0
1	11	2.825	9,059,051	12,296	0
1	12	2.701	8,739,799	11,734	0
1	13	2.782	8,756,954	11,783	0
1	14	2.662	8,396,594	11,114	0
1	15	2.278	7,488,199	8,922	0
1	16	2.727	8,480,005	11,238	0

Figure 5. Skew monitoring for one application

In this example, the numbers differ considerably for partitions 4, 6, and 10. To find the reason for these differences, you must first determine the application in which this skew occurs. Then, check the distribution of the tables that are involved in the query that is run by this application.

The following query retrieves the average, maximum, and corresponding skew values for the CPU time, section processing time, number of rows that are read, and number of rows that are written for the data partitions for each application. The query also retrieves the CPU slowdown value. This query uses the `MEM_GET_CONNECTION` table function and the following monitor elements:

- `total_cpu_time`: The total user and system CPU time, measured in milliseconds
- `total_rqst_time`: The time that the DB2 product spends executing requests
- `rows read`: The number of rows that are read
- `rows written`: The number of rows that are written
- `rows modified`: The number of rows that are inserted, updated, or deleted

```
SELECT  APPLICATION_HANDLE,
        count( MEMBER) as NUM_PARTITIONS,
        avg(TOTAL_CPU_TIME/1000) as AVG_TOTAL_CPU_TIME_MS,
        max(TOTAL_CPU_TIME/1000) as MAX_TOTAL_CPU_TIME_MS,
        decimal(case when max(TOTAL_CPU_TIME) > 0 then (1-
avg(TOTAL_CPU_TIME*1.0)*1.0/max(TOTAL_CPU_TIME)) else 0 end, 8,4) as SKEW_TOTAL_CPU_TIME,
        decimal(case when avg(TOTAL_CPU_TIME) > 0 then max(TOTAL_CPU_TIME)*1.0/avg(TOTAL_CPU_TIME)
else 1 end, 8, 4) as SLOWDOWN_CPU_TIME,
        avg(TOTAL_RQST_TIME) as AVG_TOTAL_RQST_TIME_MS,
        max(TOTAL_RQST_TIME) as MAX_TOTAL_RQST_TIME_MS,
        decimal(case when max(TOTAL_RQST_TIME) > 0 then (1-
avg(TOTAL_RQST_TIME)*1.0/max(TOTAL_RQST_TIME)) else 0 end, 8, 4) as SKEW_TOTAL_RQST_TIME,
```

```
        avg(ROWS_READ) as AVG_ROWS_READ,
        max(ROWS_READ) as MAX_ROWS_READ,
        decimal( case when max(ROWS_READ) > 0 then (1- avg(ROWS_READ)*1.0/max(ROWS_READ))  else 0 end, 8,
4) as SKEW_ROWS_READ,
        avg(ROWS_MODIFIED) as AVG_ROWS_WRITTEN,
        max(ROWS_MODIFIED) as MAX_ROWS_WRITTEN,
        decimal(case when max(ROWS_MODIFIED) > 0 then (1- avg(ROWS_MODIFIED)*1.0/max(ROWS_MODIFIED))
else 0 end, 8, 4) as SKEW_ROWS_WRITTEN
FROM TABLE(MON_GET_CONNECTION(cast(NULL as bigint), -2)) AS t
where MEMBER > 0
group by APPLICATION_HANDLE;
```

FCM

Data is exchanged between database partitions through the fast communication manager (FCM). It uses shared memory if the database partitions are on the same server/LPAR or, otherwise, the internal application network.

In an ideal partitioned database setup, the FCM traffic between data partitions is minimal, and the only major traffic is between the administrator/coordinator partition and the data partitions.

Check the reasons for FCM traffic if its volume represents more than 10% of the total volume of data that is read.

Monitoring FCM

You can use the partition view of the `db2top` command to monitor the FCM traffic for all partitions of a database (`p` option). The view compares the work load of the administration node and the other data nodes.

As shown in the sample in Figure 16, the values of the columns Actual BufSent and Actual BufRcvd represent the FCM traffic. The Actual BufSent column shows the total number of FCM buffers, in 4 KB blocks, which were sent per second. The Actual BufRcvd column shows the total number of FCM buffers that were received per second.

Partition Number	Partition Status	Buffer LWM	Actual BufSent	Actual BufRcvd
0*	Active	10548	3,803,536	6,163,698
1	Active	76205	8,538,437	4,000,595
2	Active	76205	6,383,575	6,616,672
3	Active	76205	5,548,368	4,049,543
4	Active	76205	5,522,774	4,025,459
5	Active	76205	5,505,914	4,009,620
6	Active	76205	5,474,237	3,979,896
7	Active	76205	5,490,022	3,994,549
8	Active	76205	5,471,102	3,977,012
9	Active	76201	5,545,129	4,046,368
10	Active	76201	5,485,099	3,990,110
11	Active	76201	5,557,694	4,058,051
12	Active	76201	5,628,927	4,422,319
13	Active	76201	5,521,189	4,024,041
14	Active	76201	5,512,433	4,015,840
15	Active	76201	5,589,183	4,087,798
16	Active	76201	5,534,568	4,036,596

Figure 6. Partition view of db2top command

The following monitor elements are useful for monitoring the FCM traffic:

- `fcm_recv_volume`: The total amount of data, in bytes, that was received through the FCM
- `fcm_recs_total`: The total number of buffers that were received through the FCM
- `fcm_send_volume`: The total amount of data, in bytes, that was sent through the FCM
- `fcm_sends_total`: The total number of buffers that were sent through the FCM

You can use the following query to retrieve these measures for each application and database partition. It computes the data volumes in megabytes. For comparison, it also returns the volumes of logical reads.

```
SELECT  APPLICATION_HANDLE,
        MEMBER,
        decimal(FCM_RECV_VOLUME*1.0 /(1024*1024), 12,3) as FCM_RECV_VOLUME_MB,
        FCM_RECVS_TOTAL,
        decimal(FCM_SEND_VOLUME*1.0 /(1024*1024), 12,3) as FCM_SEND_VOLUME_MB,
        FCM_SENDS_TOTAL,
        decimal(
(POOL_DATA_L_READS+POOL_INDEX_L_READS+POOL_TEMP_DATA_L_READS+POOL_TEMP_INDEX_L_RE
ADS+POOL_TEMP_XDA_L_READS+POOL_XDA_L_READS)*16.0/1024, 12,3) as L_READS_VOLUME_MB
FROM TABLE(MON_GET_CONNECTION(cast(NULL as bigint), -2)) AS t
ORDER BY APPLICATION_HANDLE, MEMBER;
```

Determining the SQL statements that are being executed

If you determine a significant performance problem at the connection, unit of work, or activity level, determine the SQL statement that is responsible for that behavior.

Using the `db2top` command with the *`application_id`* interactive command option, you get the application view. This view shows the SQL statement that is being executed for the selected application. You can use the e interactive option to create an explain report for this statement with the `db2expln` command.

You can also use the SYSIBMADM.MON_CURRENT_SQL administrative view to determine the statements that are being executed, together with their associated application ID or handle:

```
select  APPLICATION_HANDLE,
        varchar(substr(APPLICATION_NAME, 1, 20),20) as APPLICATION_NAME,
        varchar(substr(SESSION_AUTH_ID, 1, 20),20) as SESSION_AUTH_ID,
        varchar(substr(CLIENT_APPLNAME, 1, 20),20) as CLIENT_APPLNAME,
        varchar(substr(STMT_TEXT, 1, 100), 100) as STMT_TEXT
from SYSIBMADM.MON_CURRENT_SQL;
```

With this query, you can also retrieve the application ID for a specific query for which you might have only partial information, such as the names of the tables that are accessed by that query or the application name. For generating EXPLAIN, use the `db2expln` and `db2exfmt` utilities.

Some typical performance scenarios

This section describes some typical performance problems and how to determine the causes. Always first check the operating system performance measures and compare them to the standard behavior of the system to see whether there are any bottlenecks or other peculiarities. After that, check the DB2 KPIs that can help explain these peculiarities.

So, the recommended process for analyzing a DB2 performance problem is as follows:

1. Check the operating system parameters to find the bottlenecks or peculiarities.
2. Check the DB2 KPIs to better understand the problem.
3. Depending on the results, determine the reasons for the performance problem and a possible solution for it.

CPU performance

It is important to monitor CPU performance to ensure that the system is running at optimal level.

High user CPU

Ideally, the DB2 product uses almost all computing resources for executing a query, resulting in a high user CPU measurement. If the execution plan is efficient, expect an optimal execution time for the query.

However, sometimes the execution plan is not optimal. An example is when a nested loop join of two tables with a small number of rows is used but a hash join would be a better choice. Since the number of rows that are read for executing a nested loop join is the product of the cardinality of the two tables, joining two relatively small tables with 100,000 rows leads to reading 10 billion rows. Because both tables fit into the buffer pool, there is no I/O where the CPUs are busy reading rows and comparing values.

The best practices paper ***Query optimization in a data warehouse*** gives hints on how to optimize the execution of SQL queries in the warehousing context.

If a query is still running, proceed as follows to determine the queries that are responsible for the high CPU load:

You can use the session view of the `db2top` command to determine the applications with the highest CPU share, as shown in Figure. Use the application view for these applications to determine the SQL statement that is being executed, and create the execution plan by using the `e` interactive option.
In the extended application view, where you get to with the `X` interactive option can see what sections use the largest percentage of CPU time. To determine possible improvements, look at these parts of the execution plan first.

Alternatively, you can determine the most active connections and the SQL statements that are being executed by combining the queries in the sections "CPU time and usage"and "Determining the SQL statements that are being executed" through an outer join in the following way:

```
with connect_agg as (
SELECT t.APPLICATION_HANDLE,
        count(t.member) as NUM_MEMBER,
        decimal(avg(t.TOTAL_CPU_TIME*1.00)/1000000, 12,2) as AVG_TOTAL_CPU_TIME_SEC,
        decimal(avg(t.TOTAL_WAIT_TIME*1.00)/1000, 12,2) as AVG_TOTAL_WAIT_TIME_SEC,
        decimal(avg(t.TOTAL_RQST_TIME*1.00)/1000, 12,2) as AVG_TOTAL_RQST_TIME_SEC,
        decimal(avg(t.TOTAL_SECTION_TIME*1.00)/1000,12,2) as AVG_TOTAL_SECTION_TIME_SEC,
        decimal(avg(t.TOTAL_SECTION_PROC_TIME*1.00)/1000, 12,2) as AVG_TOTAL_SECTION_PROC_TIME_SEC
FROM TABLE(MON_GET_CONNECTION(cast(NULL as bigint), -2)) AS t
WHERE t.MEMBER > 0
GROUP BY t.APPLICATION_HANDLE
)
select c.APPLICATION_HANDLE, NUM_MEMBER,
```

```
        AVG_TOTAL_CPU_TIME_SEC, AVG_TOTAL_WAIT_TIME_SEC, AVG_TOTAL_RQST_TIME_SEC,
        AVG_TOTAL_SECTION_TIME_SEC, AVG_TOTAL_SECTION_PROC_TIME_SEC,
        decimal(t0.TOTAL_APP_RQST_TIME/1000.0, 12,2) as TOTAL_APP_RQST_TIME_SEC,
        case when t0.TOTAL_APP_RQST_TIME > 0 then
decimal(AVG_TOTAL_CPU_TIME_SEC*100000/(t0.TOTAL_APP_RQST_TIME*2),12,2) end as CPU_USAGE_PCT,
        varchar(substr(STMT_TEXT, 1, 100), 100) as STMT_TEXT
FROM connect_agg c
inner join TABLE(MON_GET_CONNECTION(cast(NULL as bigint), -2)) AS t0 on (c.APPLICATION_HANDLE =
t0.APPLICATION_HANDLE)
left outer join SYSIBMADM.MON_CURRENT_SQL m on ( c.APPLICATION_HANDLE =
m.APPLICATION_HANDLE )
where t0.MEMBER = t0.COORD_MEMBER;
```

The left outer join is important, because with an inner join, the queries would disappear from the result set together with the computed final measures after their SQL statement terminates. If these queries are not running any more, determine the queries in the DB2 package cache with the overall highest CPU time consumption. These are candidates for query optimization.

You can use the Dynamic SQL view of the `db2top` command (`D` interactive command option) to determine where to look for the statements with the highest CPU time. To put the SQL statements with the highest CPU times at the top, you need to sort column 5 (CPU Time) in descending order, by using the `z` interactive option. You can also use the following query, which returns for the statements in the package cache the total and average CPU times in seconds per member or database partition:

```
SELECT  EXECUTABLE_ID,
        max(SECTION_TYPE) as SECTION_TYPE,
        integer(avg(NUM_EXEC_WITH_METRICS)) as AVG_NUM_EXEC_WITH_METRICS,
        decimal(sum(TOTAL_CPU_TIME)/(1000000.0*count(distinct member)),10,2) as
TOTAL_CPU_TIME_SEC_PER_MEMBER,
        decimal(sum(TOTAL_CPU_TIME)/sum(NUM_EXEC_WITH_METRICS)/1000000.0, 10,2) as
AVG_CPU_TIME_SEC_PER_MEMBER,
        max(varchar( substr(STMT_TEXT, 1, 100), 100)) as STMT_TEXT
FROM TABLE(MON_GET_PKG_CACHE_STMT ( 'D', NULL, NULL, -2)) as T
WHERE T.NUM_EXEC_WITH_METRICS > 0
GROUP BY EXECUTABLE_ID
ORDER BY TOTAL_CPU_TIME_SEC_PER_MEMBER;
```

High I/O waits

I/O waits occur when CPU I/O demand (due to table scans, for example) cannot be satisfied by the storage subsystem. The IBM Smart Analytics System is designed so that the storage subsystem offers sufficient I/O bandwidth for the typical workloads, such that the value I/O wait should be below the tolerable value of 25%.

High I/O waits might be due to table scans of broad tables, that is, tables with rows with a length of several hundred bytes, where only a few small columns are used in queries. In such a case, the DB2 product must read a huge amount of data from disk to process only a small fraction of it. Because this situation occurs when pages are accessed sequentially, it is characterized by high amounts of data read per second (for example, blocks read per second) from the physical storage.

High values for I/O waits can also occur when the DB2 product accesses pages of a table in random order. A combination of index scan and row fetch from a table can be the reason. In such a situation, the CPU must wait until the page with the selected row is loaded into the buffer pool. Because of the random page access, the amount of data that is read per second is low.

To determine possible sources of I/O waits, identify applications with a low ratio of number of rows that are read to number of physical read operations.

In the sessions view (`l` interactive option) of the `db2top` command, examine the values of the Actual RowsRead and Actual IOReads columns. The value in the Actual IOReads column is the sum of logical data, index, and temporary read operations for data pages. Not all of these items might be responsible for I/O waits. There might be applications with high logical I/O but low physical I/O. For other applications, the Actual RowsRead to Actual IOReads ratio might have been diluted by a high percentage of index or temporary reads for which one I/O read operation corresponds to one row that is read.

Figure 7 shows four applications. For the application with ID 300, the value of Actual RowsRead divided by Actual IOReads is less than 15 (380,928,039 / 25,756,192 = 14.79). That is, fewer than 15 rows are read per data page. For the other applications, this fraction is more than 300, which is why they are not responsible for an I/O wait.

Application Handle(Stat)	Cpu% Total	IO% Total	Mem% Total	Application Status	Application Name	Actual RowsRead	Actual RowsWritten	Actual IOReads
300(*)	5.32%	75.13%	0.73%	UOW Executing	db2batch	380,928,039	956,207	25,756,192
302(*)	29.97%	8.74%	63.93%	UOW Executing	db2batch	1,000,256,649	378,382	2,996,645
304(*)	31.74%	8.01%	6.36%	UOW Executing	db2batch	988,141,318	210,280	2,747,093
303(*)	32.97%	8.10%	27.44%	UOW Executing	db2batch	981,893,386	376,201	2,776,283

Figure 7. I/0 monitoring by using db2top command sessions view

After you determine the candidate applications, retrieve the SQL statement that is running by using the db2top command or the SYSIBMADM.MON_CURRENT_SQL administrative view, as described in the section "Determining the SQL statements that are being executed."

You can get the corresponding values at the connection level for the data partitions by using the following query. The query also reports the ratio of the number of rows that are read to the number of logical reads; the numbers of data, index, and temporary reads; and the SQL statements that are being executed. The query uses the following monitor elements of the `MON_GET_CONNECTION` table function:

- `rows_read`: The total number of rows that are read
- `pool_data_l_reads` and `pool_index_l_reads`: The total numbers of data and index pages that are requested from the buffer pool for regular or large table spaces
- `pool_temp_data_l_reads` and `pool_temp_index_l_reads`: The total numbers of data and index pages that are requested from the buffer pool for temporary table spaces

```
WITH READ_METRICS as (
```

```
SELECT  APPLICATION_HANDLE,
        count(*) as NUM_MEMBER,
        sum(ROWS_READ) as ROWS_READ,
        sum(POOL_DATA_L_READS) as POOL_DATA_L_READS,
        sum(POOL_INDEX_L_READS) as POOL_INDEX_L_READS,
        sum(POOL_TEMP_DATA_L_READS+POOL_TEMP_INDEX_L_READS) as POOL_TEMP_L_READS
FROM TABLE(MON_GET_CONNECTION(cast(NULL as bigint), -2)) AS m
where member > 0
group by application_handle
)
select  r.APPLICATION_HANDLE, NUM_MEMBER,
        r.ROWS_READ,
        case when POOL_DATA_L_READS+POOL_TEMP_L_READS > 0 then
decimal(r.ROWS_READ*1.00/(POOL_DATA_L_READS+POOL_TEMP_L_READS), 8,2) end as
ROWS_READ_PER_POOL_L_READ,
        POOL_DATA_L_READS, POOL_INDEX_L_READS, POOL_TEMP_L_READS,
        varchar(STMT_TEXT,100) as STMT_TEXT
from READ_METRICS r left outer join SYSIBMADM.MON_CURRENT_SQL s
ON r.APPLICATION_HANDLE = s.APPLICATION_HANDLE
order by pool_data_l_reads desc ;
```

For reducing I/0 waits, take the following actions:

- If broad tables are the probable reason for the high I/O wait, check the DDL statements of the referenced tables to determine those with the longest row lengths. You can compress the tables with long rows such that fewer I/0 operations are needed for reading the same number of rows. Alternatively, consider redesigning the table by moving the less-utilized long columns into a separate table, or consider defining an MQT for the most frequently used columns.

- In the case of random page accesses, check whether the execution plan of the query contains the combination of an index scan and fetch operator on the right side of a nested loop join. This is described in the section "Random lookups" on page 40 of the *Query optimization in a data warehouse* best practices paper. Take the actions described there.

High system CPU time

High system CPU time can be caused by swapping or a large number of context switches. The section "High memory usage" describes measures to avoid swapping.

A large number of connections might require high system CPU time because of a high number of context switches, particularly on the administration node or on the server/LPAR where the DB2 coordinator partition is located. Because the number of context switches is determined by the number of concurrent connections, you can use workload management to avoid having this number grow too high.

Intraparallelism increases the number of context switches. If system CPU times are constantly too high, consider decreasing the value of the `dft_degree` database configuration parameter or setting the value of the `intra_parallel` database manager configuration parameter to `NO`.

Imbalanced CPU load

In a partitioned database environment, during the execution of a query, you should ensure that the data server CPUs is equally used. However, even if all hash-partitioned tables are evenly distributed across all database partitions, the load on certain CPUs might be higher than on others. This might happen, for example, if certain where-conditions have different selectivity on the database partitions. The section "Data skew" describes how to monitor skews.

In the case of a CPU skew on the data nodes, check whether you obtain better results with another distribution key. Usually, more fine-grained distribution keys with more distinct values and a more balanced frequency distribution are less sensitive to skews on subsets. If a few values, such as NULL or default values with high frequencies are responsible for the skew, a multicolumn distribution key can help as well. If the skew occurs only with some specific where-conditions, you might decide to live with that situation.

A major percentage of the workload might be running on the administration node, that is, on the coordinator partition. This can happen if a query returns huge result sets or if a query compares values that are distributed across all partitions. An example of such a query is as follows. In this query, the table yourschema.yourtable is partitioned according to different columns.

```
select count (distinct col) from yourschema.yourtable
```

Another possible reason for such a skew is that a large table is not partitioned across the data nodes. In such a situation, it might be better to partition it or to replicate it across the data nodes. Guidelines on this topic are in the section "Determine how to partition the database" of the best practices paper *Query optimization in a data warehouse.*

High I/O rates

High I/O rates can have severe negative consequences on the performance of your database system.

High read I/O rates

High read I/O rates are typical for data warehousing applications because scans of large fact tables can occur frequently. However, only a percentage of the rows of a table might be selected through a WHERE clause. In such a case, data partitioning, multidimensional clustering (MDC) tables, or indexes can help to read the data in a more selective way.

To determine the candidate tables for this optimization, use one of the following approaches:

- For running queries determine the applications and associated SQL statements with the highest read activity (number of rows that are read and number of logical reads).
- For an ex-post analysis, use the `MON_GET_TABLE` monitoring table function to determine the tables with read activity and table scans since the last database activation, using the `rows_read`

and `table_scans` monitor elements. Next use the `MON_GET_PACKAGE_CACHE_STMT` function to determine the statements in the package cache containing those tables, and examine those statements with high numbers of rows read and logical reads.

The following query determines a superset of these table and SQL statement combinations, because of the use of the LIKE predicate that is used to determine whether a schema-table name combination occurs in an SQL statement. It assumes that the table names are fully qualified with the schema tabschema in the queries and returns apart from the table schema with name, the statement text, the number of table scans, the total number of rows that are read from the table, the number of executions of the statement, its number of rows read, logical reads, and its total activity time.

```
with montable as (
SELECT  varchar(tabschema,20) as tabschema,
        varchar(tabname,20) as tabname,
        max(table_scans) as table_scans,
        sum(rows_read) as table_rows_read
FROM TABLE(MON_GET_TABLE('','',-2)) AS t
WHERE tabschema not in ('SYSCAT', 'SYSIBM', 'SYSIBMADM', 'SYSPUBLIC', 'SYSSTAT', 'SYSTOOLS' ) and
rows_read > 0 GROUP BY tabschema, tabname
)
select  tabschema, tabname, max(table_scans) as table_scans, max(table_rows_read) as table_rows_read,
        count(member) as NUM_MEMBER,
        max(NUM_EXEC_WITH_METRICS) as STMT_NUM_EXECS_WITH_METRICS,
        decimal(sum(TOTAL_ACT_TIME)/1000.00,10,2) as STMT_ACT_TIME_SEC,
        sum(ROWS_READ) as STMT_ROWS_READ,
sum(POOL_DATA_L_READS+POOL_INDEX_L_READS++POOL_TEMP_DATA_L_READS+POOL_TEMP_INDEX_
L_READS) as STMT_POOL_L_READS,
        sum(POOL_DATA_L_READS) as STMT_POOL_DATA_L_READS,
        sum(POOL_INDEX_L_READS) as STMT_POOL_INDEX_L_READS,
        sum(POOL_TEMP_DATA_L_READS+POOL_TEMP_INDEX_L_READS) as STMT_POOL_TEMP_L_READS,
        varchar(substr(s.stmt_text, 1,100),100) as stmt_text
from    montable t,
        TABLE(MON_GET_PKG_CACHE_STMT('D', null, null, -2)) as s
where   rows_read > 0   and  lcase(s.stmt_text) like '%' || lcase(trim(t.tabschema)) || '%' || '.%' ||
lcase(trim(t.tabname)) || '%'
group by t.tabschema, t.tabname, varchar(substr(s.stmt_text, 1,100),100)
order by stmt_rows_read ;
```

High write I/O rates

For a query workload, write I/O activity can occur if the sort heap is too small for the temporary space that is needed for sorting a table or for performing a hash join. In such a case, a percentage of the temporary space is allocated in temporary tables that are written to the system temporary table space.

If the system temporary table space is located on its own file system (which is the case if you use SSD storage for temporary data) you can check for write I/O at the operating system level by using tools such as the `nmon` utility or `iostat` command.

In any case, increase the value of the `sortheap` database configuration parameter if hash loops occur frequently because they can slow down a query by a factor higher than 10. If the value of the `sortheap` parameter is below the one that is recommended for the IBM Smart Analytics System, increase the value of the parameter to the recommended value. Otherwise, increase it by 50% - 100%, and then check whether hash loops still occur.

In general, when turning the value of the `sortheap` parameter, consider the maximum number of applications (APPMAX) that perform sorts concurrently. Generally, the value of the `sortheap` parameter should not be greater than the value of the `sheapthres` database manager configuration parameter divided by the value of APPMAX.

Monitor the main memory usage for sort and hash joins after every increase of the `sortheap` parameter to check that no paging occurs. If the memory usage gets close to 100%, restrict the number of concurrent applications through workload management.

Be careful not to set the `sortheap` parameter to a value that is too high. Otherwise, just a few concurrently running applications with large sorts or hash joins can lead to such a large sort heap that the system memory starts paging.

High network traffic

In a partitioned database, there might be the need to transfer data between partitions. Best case, small result sets must be sent from data nodes to the coordinator node for consolidation only. However, large amounts of data might have to be exchanged between the data partitions when a query is being executed.

Because FCM controls the exchange of data between database partitions, you should check the FCM-related performance metrics even though it results in network traffic only if the partitions are on different hosts.

High network traffic between data nodes

In a partitioned database environment where the data is spread across different partitions, non-collocated joins are a common problem. These result in high network traffic through the FCM between the data partitions. Identify the applications that cause the highest network traffic by using the query in the section "FCM monitoring," by using the `MON_GET_CONNECTION` table function, and check the values for the `fcm_recv_volume_mb` and `fcm_send_volume_mb` monitor elements.

If you want to determine the SQL statement that is currently causing highest FCM traffic, you can use a query that is similar to the query in the section "High I/O wait," which is based on the `MON_GET_CONNECTION` table function and MON_CURRENT_SQL administrative view. Include the `fcm_recv_volume` and `fcm_send_volume` monitor elements.

An alternative is to use the session view of the `db2top` command. First, switch to the non-delta view by using the `k` option. Next, order the session view based on the column TQr+w, which shows the number of rows that are read from and written to table queues. For a descending sort of the view, use the `z` option and enter 10 for the column TQr+w.

Figure 18.8 shows that the session with application handle 36256 has a high value in the TQr+w column, which means there might be a problem with the collocation that is based on table queues:

```
Application            Actual
Handle(Stat)            TQr+w
------------ --------------
    36256(*)       2,326,712
    36243(*)               0
    36259(*)               0
```

Figure 18.8 Table queue monitoring with the db2top command

Finally, create the explain plans as described in the section "Determining the SQL statements that are being executed" for the currently running queries that generate network traffic. Within these access plans, you should see at least one directed table queue (DTQ). The *Query optimization in a data warehouse* best practices paper shows you how to avoid table queues and ensure collocation.

High network traffic toward the coordinator partition or administration node

If there is high network traffic toward the coordinator or administration node, you can proceed in a similar way to that explained in the previous section, but restrict your analysis to the FCM traffic for partition 0.

A possible reason for such high network traffic is that a big non-partitioned table is stored to all data partitions, which you can see as the broadcast table queue (BTQ) in the explain plan of the query. In this case, consider replicating or partitioning the table. Another reason can be that the query returns a huge result set that must be sent from the data partition to the coordinator partition.

High memory usage

IBM Smart Analytics Systems are preconfigured to avoid swapping. All database and instance parameters are set to values that should not require swapping under normal circumstances.

If swapping occurs or main memory usage gets close to 100%, determine the most important DB2 memory consumers.

These consumers are as follows:

- FCMBP shared memory: The amount of this memory is controlled by the DB2 product. You should not modify the amount of this memory.

- Buffer pools: The buffer pools are, in general, the most important main memory consumers. On an IBM Smart Analytics System 7700 or IBM PureData System for Operational Analytics, the buffer pool occupies a data node that uses approximately 30% of main memory. This huge size was calculated for the IBM Smart Analytics System under the assumption that only one database is active at a time.

  If the buffer pool occupies more main memory, check whether more than one database is active. If that is true, ensure that only one database can be active by setting the `numdb` database manager configuration parameter to 1, or consider reducing the size of the largest buffer pool, BP_16K.

- Utilities: The IBM Smart Analytics System default value for the util_heap_sz database configuration parameter should not cause paging if only one database is active. However, the size of the utility heap can grow above the size that is specified by the util_heap_sz configuration parameter if you run load jobs using the DATA BUFFER parameter with high values. Limit the number of concurrently running load jobs, and set the DATA BUFFER parameter for those jobs to lower values.

- Sort heap: The possibilities for tuning the sort heap are covered in the section "**Error! Reference source not found.**."

Underutilized system

An underutilized system is characterized on the DB2 side by high wait times. In such a situation, check the following wait time-related monitor elements. The values are in milliseconds.

- total_wait_time: The total wait time.
- total_act_wait_time: The total wait time within an activity such as execution of SQL statements, loads, or stored procedures.
- Fcm_recv_wait_time and fcm_send_wait_time: The time that is spent waiting to receive or send data through the FCM. Typically, the coordinator partition has a high value for the fcm_recv_wait_time monitor element because it waits for the results from the data partitions. High values for the data partitions indicate a skew of the data that is sent by the FCM.
- lock_wait_time: The time that is spent waiting for locks. In this case, determine the blocking applications as described in the section "Locking " and, if necessary, force the blocking applications.

To avoid having an underutilized system, see "Diagnosing and resolving locking problems" in the DB2 10.5 Information Centre (http://publib.boulder.ibm.com/infocenter/db2luw/v10r5/topic/com.ibm.db2.luw.admin.trb.doc/doc/c0055071.html).

Conclusion

This paper presented the main monitoring methods for operating system and database performance issues for warehouse systems that are based on DB2 software, such as the IBM Smart Analytics System and IBM PureData System for Operational Analytics. For these methods, this paper described how to monitor the system with the commands, utilities, or DB2 monitoring functions.

The recommended overall approach for finding the reason for a performance problem is to look first at the operating system measurements to check whether they have unusual values. Then, look at the corresponding DB2 measurements to determine the cause for the problem.

Best practices

- Monitor all servers of a cluster to detect anomalies that might impact overall performance. To run a command-line tool on all servers of a cluster in a single invocation, use the `rah` command.

- For multi-core and multithreaded CPUs, in addition to checking the overall CPU usage, check the usage at the thread or core level to determine core-related or thread-related bottlenecks.

- Examine the length of the run queue and the number of process switches to determine whether they are the reason for high system CPU usage.

- Understand the performance characteristics of your storage system. To determine the maximum possible I/O performance, generate a sample I/O workload that reads from and writes to raw physical devices by using the `dd` command on AIX operating systems and the `sg_dd` command on Linux operating systems.

- If the network performance is lower than you expect, look at the number of sent and received packages and the number of transmission errors. These errors should be less than 1% of the number of received and sent packages.

- Avoid paging. Short periods of small amounts of paging space usage, where the amount of paging space is less than 5% - 10 % of the physical main memory size, are tolerable. However, higher paging space usage can lead to a severe system slowdown, so that simple DB2 or operating system commands seem to hang.

- For monitoring DB2 performance, use the table functions that are based on the in-memory metrics. To monitor running queries, use the `MON_GET_UNIT_OF_WORK` and `MON_GET_ACTIVITY_DETAILS` functions. If a query ends and the connection is still open, you can use the `MON_GET_CONNECTION` function to return the aggregated performance measurements for all activities that were executed for that connection. Separately determine the measures for each DB2 partition (referred to as *member* by these functions) by specifying `-2` as the second parameter for these functions so that all members are considered. Then, aggregate the measures for the whole system.

- For SQL queries, review wait times, such as lock wait time, I/O wait time, and FCM wait time. These wait times should be as low as possible, except for the coordinator partition. The coordinator partition, which can wait on the results of the data partitions because they should do almost all the processing in a well-partitioned database setup.

- To see the effectiveness of the prefetchers, compare the number of asynchronous read and writes with the total number of physical read and write operations. If asynchronous reads and writes are considerably below 80% of the number of physical read and write operations, increase the value of the `num_ioservers` database configuration parameter.

- To ensure that the DB2 software does not exhaust the main memory resources, always monitor DB2 memory usage with operating system memory usage.

- Avoid post-threshold sorts. Otherwise, main memory might be exhausted, and paging might start, slowing down the system.

- When monitoring CPU skew, calculate the slowdown of a query for estimating the performance improvement with an even distribution of the data. Given that the execution time of a query is bounded by the slowest partition (with maximum CPU time), it is defined as max(*CPU_time*) / avg(*CPU_time*).

- If the volume of FCM traffic is more than 10% of the total volume of data that is read, check the reasons for the traffic.

- To determine possible sources of I/O waits, identify applications with a low ratio of number of rows that are read to number of physical read operations.

- To check that no paging occurs, monitor the main memory usage for sort and hash joins after every increase of the sort heap. If the memory usage gets close to 100%, restrict the number of concurrently running applications through workload management.

Appendix A. db2top command hints

Batch mode of db2top command

You can use the batch mode of the `db2top` command for recording the performance of a workload. For information about the batch mode and how to reproduce it, see the "DB2 problem determination using db2top utility" article (http://www.ibm.com/developerworks/data/library/techarticle/dm-0812wang/#batch_mode).

Monitoring a single partition

In a partitioned database environment, you might want to monitor a specific partition rather than the whole system. To monitor a specific partition, specify the `P` option and the partition id. To change the order of the columns, you can use the `c` option together with a list of the numbers of the columns to be displayed. A recommended column selection and order for the best overview of possible bottlenecks as shown in Figure 19 is:
"0,5,23,24,1,2,3,14,15,16,30,31,10,36,37"

Application Handle(Stat)	Application Name	Os User	DB User	Cpu% Total	IO% Total	Mem% Total	Lockwait (sec)	Locks Held	Sorts (sec)	Sorts Overflows	Hash Join Overflows	Delta TQr+w/s	Session Cpu
19079(*)	db2batch	bcuaix10	BCUAIX10	5.94%	8.16%	4.42%	0	107	1447	0	0	0	1453.99603
19311(*)	db2batch	bcuaix10	BCUAIX10	6.52%	13.33%	7.84%	0	106	1469	0	0	0	1457.51417
19350(*)	db2batch	bcuaix10	BCUAIX10	7.13%	11.87%	11.06%	0	100	1464	0	0	9,608	1456.25414
19343(*)	db2batch	bcuaix10	BCUAIX10	7.32%	12.09%	11.25%	0	100	1446	0	0	130,578	1443.69199
19254(*)	db2batch	bcuaix10	BCUAIX10	7.95%	16.07%	11.16%	0	100	1480	0	0	0	1456.22068
19341(*)	db2batch	bcuaix10	BCUAIX10	11.23%	16.11%	11.24%	0	100	1459	0	0	0	1455.90590

Figure 19. Rearranged db2top command view to show possible bottlenecks

If you use option `w` in the session, you can save this order in the `.db2toprc` `db2top` configuration file so that this setting is available every time that you issue the `db2top` command.

Collecting operating system measurements in the db2top command

You can modify the header of the `db2top` command screen to show operating system measurements by updating the `.db2toprc` configuration file. Include the following two lines:

```
cpu=vmstat 2 2 | tail -1 | awk '{printf("%d(usr+sys)",$14+$15);}'
io=vmstat 2 2 | tail -1 | awk '{printf("%d(bi+bo)",$10+$11);}'
```

They show the CPU usage of the server where you issued the `db2top` command. The CPU usage is displayed in the upper right of the `db2top` command output, and the I/O load is displayed in the upper left.

Linux operating system monitoring

CPU monitoring

For CPU monitoring, you can use the `rah` command with the `vmstat` command or "`sar -u`" command. You can obtain details for each core by entering "`sar -u -P ALL`". For details about the usage of the `sar` command see "Appendix B. Operating system monitoring for a cluster."

I/O monitoring

To display I/O performance by file system, use the `sar` command with the `-p` and `-d` options, as shown in Figure 20:

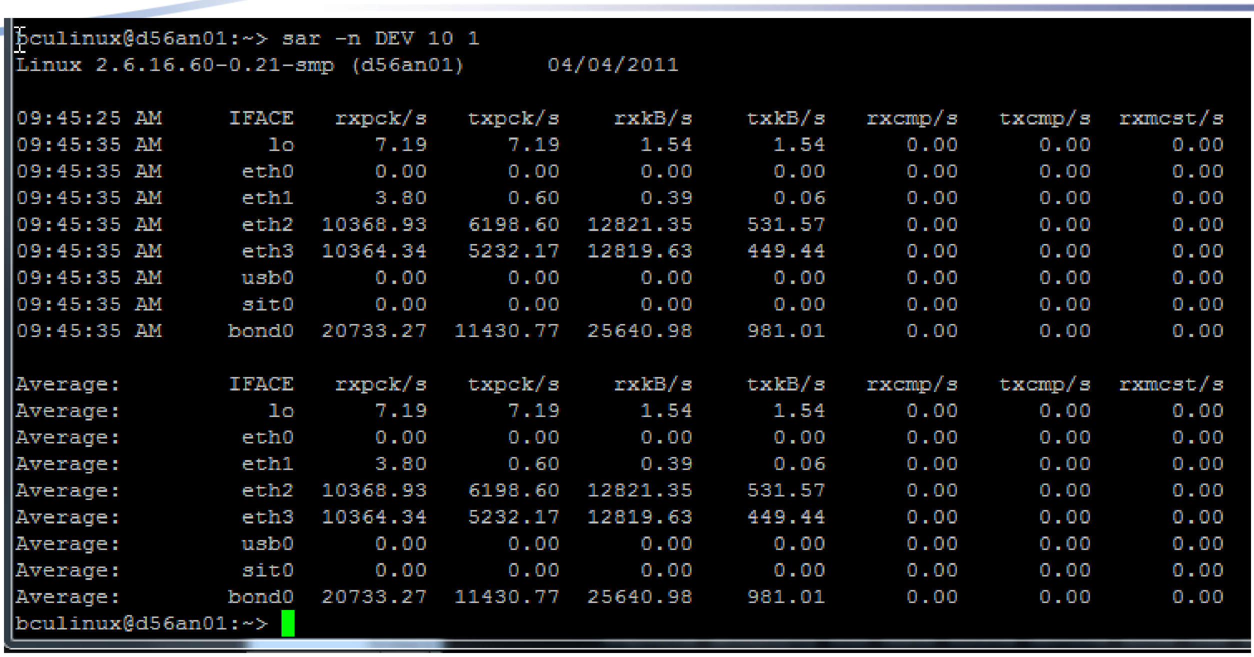
```
bculinux@d56an01:~> sar -p -d 10
Linux 2.6.16.60-0.21-smp (d56an01)      04/04/2011

09:16:59 AM       DEV       tps  rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz     await     svctm     %util
09:17:09 AM       sda      5.00      0.00    134.27     26.88      0.08     15.84      3.12      1.56
09:17:09 AM       sdb      1.90      0.00     17.98      9.47      0.01      7.79      7.37      1.40
09:17:09 AM       sdc      0.20      0.00      6.39     32.00      0.00     14.00     14.00      0.28
09:17:09 AM       sdd      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
09:17:09 AM       sde      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
09:17:09 AM system-optlv      1.90      0.00     15.18      8.00      0.06     29.26      2.11      0.40
09:17:09 AM system-tmplv      1.60      0.00     12.79      8.00      0.03     18.25      3.50      0.56
09:17:09 AM system-varlv      6.49      0.00     51.95      8.00      0.03      5.35      1.23      0.80
09:17:09 AM vgnfs-lvnfsdb2home      1.40      0.00     11.19      8.00      0.01      4.29      4.29      0.60
09:17:09 AM vgnfs-lvnfsvarlibnfs      2.30      0.00      4.60      2.00      0.01      4.00      3.48      0.80
09:17:09 AM vgdb2NODE0000-lvdb2NODE0000      0.70      0.00      5.59      8.00      0.00      4.00      4.00      0.28
09:17:09 AM vgstage-lvstage      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
09:17:09 AM system-db2dump      0.20      0.00      1.60      8.00      0.00      6.00      4.00      0.08
```

Figure 20. Output of the sar -p -d 10 command

The amount of data that is transferred is shown in 512 byte sectors, for example, as writes per seconds in the `wr_sec/s` column or reads per second in the `rd_sec/s` column. The output**Error! Reference source not found.** shows a mix of activity by physical device and logical volume. You can use output from the devices to check whether the activity is balanced. The file system view describes where activity happens, for example, activity about `lvstage` reading load files.

Network monitoring

The `sar -n DEV` command displays the network activity for all available interfaces, as shown in the example in Figure 21. The throughput for the bond0 network, which is the FCM network for IBM Smart Analytics Systems clusters that are based on Linux operating systems, is the sum of the eth2 and eth3 activity. Important for performance monitoring are the columns reporting the numbers of packages that are received or sent per second and the number of kilobytes that are received or sent per second.



```
bculinux@d56an01:~> sar -n DEV 10 1
Linux 2.6.16.60-0.21-smp (d56an01)      04/04/2011

09:45:25 AM     IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
09:45:35 AM        lo      7.19      7.19      1.54      1.54      0.00      0.00      0.00
09:45:35 AM      eth0      0.00      0.00      0.00      0.00      0.00      0.00      0.00
09:45:35 AM      eth1      3.80      0.60      0.39      0.06      0.00      0.00      0.00
09:45:35 AM      eth2  10368.93   6198.60  12821.35    531.57      0.00      0.00      0.00
09:45:35 AM      eth3  10364.34   5232.17  12819.63    449.44      0.00      0.00      0.00
09:45:35 AM      usb0      0.00      0.00      0.00      0.00      0.00      0.00      0.00
09:45:35 AM      sit0      0.00      0.00      0.00      0.00      0.00      0.00      0.00
09:45:35 AM     bond0  20733.27  11430.77  25640.98    981.01      0.00      0.00      0.00

Average:        IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
Average:           lo      7.19      7.19      1.54      1.54      0.00      0.00      0.00
Average:         eth0      0.00      0.00      0.00      0.00      0.00      0.00      0.00
Average:         eth1      3.80      0.60      0.39      0.06      0.00      0.00      0.00
Average:         eth2  10368.93   6198.60  12821.35    531.57      0.00      0.00      0.00
Average:         eth3  10364.34   5232.17  12819.63    449.44      0.00      0.00      0.00
Average:         usb0      0.00      0.00      0.00      0.00      0.00      0.00      0.00
Average:         sit0      0.00      0.00      0.00      0.00      0.00      0.00      0.00
Average:        bond0  20733.27  11430.77  25640.98    981.01      0.00      0.00      0.00
bculinux@d56an01:~>
```

Figure 11. Using the sar -n DEV command to monitor network activity

To monitor data transfer between servers in a cluster, issue the following command on the management node:

```
dsh –a  -s sar –n DEV 10 0 | egrep –I “iface|bond”
```

This restricts the output to the bonded FCM network device. If you need additional details for all network interfaces, exclude the `egrep` statement.

Appendix B. Operating system monitoring for a cluster

For operational system monitoring, use the `sar` command, which collects and stores operating system performance data in a space-efficient way. Collecting data for all system parameters (disk, CPU, network, and process queue) in 10-second intervals requires approximately 44 MB of space in binary format per server and per day. If you provide more space, you can store performance data in 1-second or 2-second intervals.

The following commands apply only to Linux operating systems:

A sample entry in the /etc/inittab file has the following sample content:

```
sar:2345:respawn:/usr/bin/sar -A -o 10 0 1>/dev/null
```

To add this entry to all /etc/inittab files in the cluster, issue the following command as root user from the management node, after editing the /etc/inittab file:

```
dsh -a "echo 'sar:2345:respawn:/usr/bin/sar -A -o 10 0 1>/dev/null' >> /etc/inittab"
```

Apply the change on all nodes by issuing the following command:

```
dsh -a "telinit q"
```

The output is written to the /var/log/sa/sa*DD* file, where *DD* is the number of the current day.

To extract the CPU performance data for the ninth day of the current month between 10:00 a.m. and 11:00 a..m. in 60-second intervals from a binary format, issue the following command on one node:

```
sar -f /var/log/sa/sa09 -u -s 10:00:00 -e 11:00:00 60
```

The following sample output is obtained:

```
10:33:02 AM     CPU     %user     %nice   %system   %iowait    %steal     %idle
10:34:02 AM     all      0.02      0.00      0.05      0.14      0.00     99.80
10:35:02 AM     all      0.01      0.00      0.06      0.25      0.00     99.68
10:36:02 AM     all      0.01      0.00      0.06      0.20      0.00     99.73
10:37:02 AM     all      0.01      0.00      0.05      0.14      0.00     99.80
10:38:02 AM     all      0.01      0.00      0.06      0.17      0.00     99.75
10:39:03 AM     all      0.02      0.00      0.06      0.14      0.00     99.78
10:40:03 AM     all      0.02      0.00      0.07      0.23      0.00     99.68
10:41:03 AM     all      0.02      0.00      0.07      0.16      0.00     99.75
```

Figure 22. Sample operating system cluster output

One benefit of `sar` command output compared to `vmstat` or `iostat` command output is that `sar` command output has a time stamp to allow the analysis of previous system situations.

The `sar` command does not overwrite old files. After one month, content is added to the corresponding file of the preceding month in the /var/log/sa folder. To avoid this, setup the following commands as cron job for the root user:

```
5 0 * * * find /var/log/sa -name sa?? -ctime +24 -exec compress {} \; 2>&1

10 0 * * * find /var/log/sa -name sa??.Z -ctime +24 -exec rm {} \; 2>&1
```

The first command identifies all files that are older than 24 days and compresses them. The second one removes all compressed files that are older than 24 days. These commands provide a space-efficient way of storing performance data for approximately 50 days.

To implement changes for a cron job, issue the following commands:

```
dsh -a "echo '5 0 * * * find /var/log/sa -name sa?? -ctime +24 -exec compress {} \; 2>&1' >>
/var/spool/cron/tabs/root"

dsh -a "echo '10 0 * * * find /var/log/sa -name sa??.Z -ctime +24 -exec rm {} \; 2>&1' >>
/var/spool/cron/tabs/root"
```

Displaying performance data in the cluster

The following command, issued from the management node, stored performance data from nodes as a formatted table, as shown in Figure 23:

```
dsh -N BCUDATA "sar -u -f /var/log/sa/sa10 10 -s 10:00:00 -e 10:01:00 | egrep -v 'Linux|^$|Average'"
```

```
d51mn01:~ # dsh -N BCUDATA "sar -u -f /var/log/sa/sa10 10 -s 10:00:00 -e 10:01:00 | egrep -v 'Linux|^$|Average'"
d51dn01: 10:00:06        CPU     %user     %nice   %system   %iowait    %steal     %idle
d51dn01: 10:00:16        all      0.02      0.00      0.05      0.30      0.00     99.63
d51dn01: 10:00:26        all      0.00      0.00      0.00      0.13      0.00     99.87
d51dn01: 10:00:36        all      0.10      0.00      0.12      0.17      0.00     99.60
d51dn01: 10:00:46        all      0.00      0.00      0.05      1.12      0.00     98.83
d51dn01: 10:00:56        all      0.03      0.00      0.05      0.12      0.00     99.80
d51dn04: 10:00:06        CPU     %user     %nice   %system   %iowait    %steal     %idle
d51dn04: 10:00:16        all      0.02      0.00      0.07      0.30      0.00     99.60
d51dn04: 10:00:26        all      0.00      0.00      0.03      0.13      0.00     99.85
d51dn04: 10:00:36        all      0.05      0.00      0.07      0.42      0.00     99.45
d51dn04: 10:00:46        all      0.00      0.00      0.00      0.33      0.00     99.67
d51dn04: 10:00:56        all      0.00      0.00      0.05      0.15      0.00     99.80
d51dn02: 10:00:04        CPU     %user     %nice   %system   %iowait    %steal     %idle
d51dn02: 10:00:14        all      0.00      0.00      0.05      0.25      0.00     99.70
d51dn02: 10:00:24        all      0.00      0.00      0.05      0.12      0.00     99.83
d51dn02: 10:00:34        all      0.00      0.00      0.00      0.18      0.00     99.82
d51dn02: 10:00:44        all      0.05      0.00      0.00      0.35      0.00     99.60
d51dn02: 10:00:54        all      0.00      0.00      0.05      0.13      0.00     99.82
d51dn03: 10:00:00        CPU     %user     %nice   %system   %iowait    %steal     %idle
d51dn03: 10:00:10        all      0.05      0.00      0.07      0.53      0.00     99.35
d51dn03: 10:00:20        all      0.05      0.00      0.03      0.15      0.00     99.78
d51dn03: 10:00:30        all      0.02      0.00      0.05      1.12      0.00     98.80
d51dn03: 10:00:40        all      0.00      0.00      0.03      0.33      0.00     99.65
d51dn03: 10:00:50        all      0.00      0.00      0.05      0.12      0.00     99.83
d51dn03: 10:01:00        all      0.03      0.00      0.03      0.15      0.00     99.80
d51mn01:~ #
```

Figure 23. Sample operating system cluster output

If you want to work with this data, you can copy and paste it into a spreadsheet. The fixed column format allows data to be easily formatted and displayed as a graph.

Further reading

- The Information Management Best Practices portal is the entry point to the best practices publications for all IBM Information Management products, including DB2 for Linux, UNIX, and Windows and DB2 Warehouse (http://www.ibm.com/developerworks/data/bestpractices).

- The *Managing data warehouse performance with IBM InfoSphere Optim Performance Manager* best practices paper describes how to use IBM InfoSphere Optim Performance Manager for monitoring the performance of a data warehouse.

- The *Query optimization in a data warehouse* best practices paper gives hints and tips on how to optimize queries in the data warehouse context.

- The *Tuning and Monitoring Database System Performance* best practices paper treats the monitoring a DB2 database from the OLTP perspective.

- Workload management is a key element in helping to achieve consistent data warehouse performance. The *Implementing DB2 workload management in a data warehouse* best practices paper gives guidance on how to implement workloads with the DB2 workload management functionality.

- The `db2top` utility is described in the "DB2 problem determination using db2top utility" IBM DeveloperWorks article

- To learn more about IBM Information Management products, see the Information Management website (http://www.ibm.com/software/data).

Contributors

Garrett Fitzsimons
Data Warehouse Specialist

Richard Lubell
Data Warehouse Information Developer

Serge Boivin
Senior DB2 Information Developer

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 does 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.

COPYRIGHT LICENSE: Copyright IBM Corporation 2013. All Rights Reserved.

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.

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.

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.

Contacting IBM

To provide feedback about this paper, write to db2docs@ca.ibm.com.

To contact IBM in your country or region, check the IBM Directory of Worldwide Contacts (http://www.ibm.com/planetwide).