MaxDB Internals
Performance Analysis
Version 7.8

Christiane Hienger
MaxDB provides various tools and methods for the analysis of performance bottlenecks and monitoring current database activities. Some of these tools were originally developed only for testing and analysis in MaxDB development, but can also be used by experienced database administrators for performance analysis.

The following are of particular importance for performance analysis:

- The **x_cons** console for monitoring current operations
- The **Database Analyzer** program for analyzing performance bottlenecks
- As of version 7.8 Shared SQL collects performance data with each statement in the cache. This substitutes the former Resource Monitor in older versions which had to be enabled explicitly.
- The Command Monitor provides a list of long-running or poorly-processed SQL statements

x_cons and Database Analyzer are stand-alone programs and are called from the operating system command line. The Command Monitor and Shared SQL are part of the core functions of the MaxDB kernel.

In SAP WebAS, all functions and results can be controlled and analyzed using transaction DBACockpit => Current Status or DBACockpit => Performance.

The former transactions DB50 and ST04 have been substituted by transaction DBACockpit in newer SAP WebAS versions.
DB Console: \textit{x\_cons}

Database console \textit{x\_cons} features:
- process overview
- configuration overview
- observing session activities and wait states
- watching I/O activities and wait queues
- measuring of detailed task specific times

Call:
- \texttt{x\_cons <servername> <command> [<interval>] [<repeat>]}
  e.g. \texttt{x\_cons WB5 show active 10 6}
  advantage: delta information using `interval` and `repeat`
- \texttt{dbmcli -d ... -u ... [-n <node>] db\_cons <command>}
  advantage: works per remote connection to database host

\textbf{DB Console \textit{x\_cons}}

The database console \textit{x\_cons} gives you a quick overview of the operating system resources that the database system is using, the distribution of the database session among the operating system threads, and the status of the active database sessions. You can also use other functions that are intended mainly for support employees and developers.

Start on shell level: \texttt{x\_cons <dbname> <command> [<interval>] [<repeat>]}
\texttt{x\_cons <dbname> help} returns a complete overview of all available command functions.

The database console can also be addressed remotely via the DBM server.
CANCEL  <index>    cancels the command executed by task <index>
KILL  <index>   kills the session of task <index>

SHOW

DEBUGLEV  level  set debug level for the kernel
DEBUGTASK  <index> writes back trace of task to knldiag
RESET  obj_cnt  resets counter about the following objects:
         IO T_CNT REGIONS (ALL)
         incl. local counters of any task
TIME  enable  enables time measurements
QUIT  exit console driver

The option –e before a SHOW command shows an extended output for tasks lists.
x_cons Process Configuration (1)

x_cons <dbname> show rte

Kernel Threads:

<table>
<thead>
<tr>
<th>Name</th>
<th>Tid</th>
<th>UNIX State</th>
<th>Sleep Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER</td>
<td>16660</td>
<td>Sleeping</td>
<td>2</td>
</tr>
<tr>
<td>COORDINATOR</td>
<td>16639</td>
<td>Sleeping</td>
<td></td>
</tr>
<tr>
<td>CLOCK</td>
<td>16654</td>
<td>Sleeping</td>
<td></td>
</tr>
<tr>
<td>CONSOLE</td>
<td>16655</td>
<td>Sleeping</td>
<td></td>
</tr>
<tr>
<td>REQUESTOR</td>
<td>16658</td>
<td>Sleeping</td>
<td></td>
</tr>
</tbody>
</table>

User Kernel Threads:

<table>
<thead>
<tr>
<th>Name</th>
<th>Tid</th>
<th>UNIX State</th>
<th>Dispatch Counter</th>
<th>TaskSwitch Active Counter</th>
<th>Tasks</th>
<th>Tasks Cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>UKT1</td>
<td>19067</td>
<td>Sleeping</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>UKT2</td>
<td>19068</td>
<td>Sleeping</td>
<td>10210</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>UKT3</td>
<td>19069</td>
<td>Sleeping</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>UKT4</td>
<td>19070</td>
<td>Sleeping</td>
<td>14525</td>
<td>393</td>
<td>43</td>
<td>43</td>
</tr>
<tr>
<td>UKT5</td>
<td>19071</td>
<td>Sleeping</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>UKT6</td>
<td>19072</td>
<td>Sleeping</td>
<td>12537</td>
<td>3533</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>UKT7</td>
<td>19073</td>
<td>Sleeping</td>
<td>70395</td>
<td>52103</td>
<td>65</td>
<td>65</td>
</tr>
<tr>
<td>UKT8</td>
<td>19074</td>
<td>Sleeping</td>
<td>239871</td>
<td>18583</td>
<td>9</td>
<td>46</td>
</tr>
<tr>
<td>UKT9</td>
<td>19075</td>
<td>Sleeping</td>
<td>457240</td>
<td>15572</td>
<td>10</td>
<td>55</td>
</tr>
</tbody>
</table>

Kernel parameters (don't change directly):

- TaskCluster01: tw/wc:ut;2000*sv;10*fs;10*gc;
- TaskCluster02: ti,100*pg;1*bup,50*us;
- TaskCluster03: equalize

Processor information:

- Processors: 8
- Processor cores: 2

---

x_cons <dbname> show rte

This shows the distribution of the MaxDB threads among the operating system processes. The DB threads coordinator, console, timer, requestor and Dev0 each have their own operating system threads. The entire database kernel runs in a single process.

However, multiple database tasks (user task, log writer, utility task, and so on) can be located together in an operating system thread, which is called a UKT (user kernel thread). The MaxDB runtime environment uses internal tasking to administer these database tasks. Internal MaxDB administration takes up less operating system time, and gives you more control over the scheduling and prioritization of individual database sessions.

The database parameters MAXCPU and UseableCPUs are normally used to distribute the tasks automatically to the UKTs; the (support) database parameter TASKCLUSTER (requires change in the control file cserv.pcf) can also be used for this purpose, but only in consultation with SAP support.
Abbreviations of the Database Tasks in TASKCLUSTER:

**Abbreviation**
- **tw**: Trace writer, writes kernel traces and dumps
- **ti**: Task for timeout monitoring
- **al**: Log writer
- **dw**: Tasks for cache monitoring and asynchronous cache displacement as well as savepoint I/O
- **ut**: Utility task for administration tasks (start backup, recovery, and so on).
- **sv**: Server processes for backup I/O and special operations such as parallel index generation
- **us**: User tasks for executing SQL statements
- **gc**: Garbage collector
- **ev**: Event task
- **fs**: Floating service for load balancing
x_cons Task Activity

`x_cons <dbname> [-e] show active [<interval>] [<repeat>]`

Presents an overview of the states of all active tasks.

**Appl pid**
- Process ID of the application program linked to the task. An asterisk (*) before the PID indicates that the process ID is on a separate computer and is being accessed remotely.

**Region**
- `cnt`: Displays the number of times the region has been accessed since the task has been running.
- `try`: The number of the queried or held region

**UKTsleep**
- Number of semaphore waits per UKT

<table>
<thead>
<tr>
<th>ID</th>
<th>UKT</th>
<th>UNIX</th>
<th>TASK type</th>
<th>APPL Current</th>
<th>Timeout</th>
<th>Region</th>
<th>Wait item</th>
</tr>
</thead>
<tbody>
<tr>
<td>T146</td>
<td>7</td>
<td>-1</td>
<td>User</td>
<td>28069 Running</td>
<td>0 220</td>
<td>99</td>
<td>741131(r)</td>
</tr>
<tr>
<td>T147</td>
<td>7</td>
<td>-1</td>
<td>User</td>
<td>28072 Runnable</td>
<td>48 0</td>
<td>111</td>
<td>741131(r)</td>
</tr>
<tr>
<td>T152</td>
<td>8</td>
<td>-1</td>
<td>User</td>
<td>28071 Runnable</td>
<td>56 0</td>
<td>76</td>
<td>424309(r)</td>
</tr>
<tr>
<td>T154</td>
<td>8</td>
<td>-1</td>
<td>User</td>
<td>28070 Running</td>
<td>0 55</td>
<td></td>
<td>424309(r)</td>
</tr>
<tr>
<td>T2</td>
<td>2</td>
<td>-1</td>
<td>Logwr</td>
<td>-1 IO Wait (W)</td>
<td>0 1</td>
<td>5</td>
<td>1978(e)</td>
</tr>
<tr>
<td>T152</td>
<td>8</td>
<td>-1</td>
<td>User</td>
<td>28069 LogIOwait (234)</td>
<td>0 0</td>
<td></td>
<td>424800(e)</td>
</tr>
<tr>
<td>T66</td>
<td>6</td>
<td>-1</td>
<td>Pager</td>
<td>-1 Vvectorio</td>
<td>0 0</td>
<td></td>
<td>3258(e)</td>
</tr>
<tr>
<td>T67</td>
<td>6</td>
<td>-1</td>
<td>Pager</td>
<td>-1 IO Wait (W)</td>
<td>0 0</td>
<td>1</td>
<td>3258(e)</td>
</tr>
<tr>
<td>T87</td>
<td>4</td>
<td>-1</td>
<td>Savepnt</td>
<td>-1 PagerWaitWrite</td>
<td>0 0</td>
<td></td>
<td>234617(e)</td>
</tr>
<tr>
<td>T75</td>
<td>4</td>
<td>-1</td>
<td>BUPvol</td>
<td>-1 AsyncWaitRead</td>
<td>0 0</td>
<td></td>
<td>11368(e)</td>
</tr>
<tr>
<td>T76</td>
<td>4</td>
<td>-1</td>
<td>BUPvol</td>
<td>-1 AsyncWaitWrite</td>
<td>0 0</td>
<td></td>
<td>11368(e)</td>
</tr>
<tr>
<td>T159</td>
<td>8</td>
<td>-1</td>
<td>User</td>
<td>28072 IO Wait (R)</td>
<td>0 0</td>
<td>2</td>
<td>429215(e)</td>
</tr>
<tr>
<td>T152</td>
<td>8</td>
<td>-1</td>
<td>User</td>
<td>28069 InvRootExcl</td>
<td>0 0</td>
<td>74573</td>
<td>24185(r)</td>
</tr>
<tr>
<td>T154</td>
<td>8</td>
<td>-1</td>
<td>User</td>
<td>28070 Running</td>
<td>0 55</td>
<td></td>
<td>438561(r)</td>
</tr>
<tr>
<td>T142</td>
<td>7</td>
<td>-1</td>
<td>User</td>
<td>0* Vwait</td>
<td>0 0</td>
<td></td>
<td>745843(e)</td>
</tr>
<tr>
<td>T157</td>
<td>8</td>
<td>-1</td>
<td>User</td>
<td>0* IO Wait (R)</td>
<td>0 0</td>
<td>1</td>
<td>852579(e)</td>
</tr>
</tbody>
</table>
In a system with one CPU, only one task can be running at a given time. If x_cons nevertheless shows two tasks running, this is due to unprotected access.

<table>
<thead>
<tr>
<th>Task State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsynClose</td>
<td>closes an I/O port after backup or recovery</td>
</tr>
<tr>
<td>Asyncntl</td>
<td>determines parameter or initialises a backup device</td>
</tr>
<tr>
<td>AsynIO</td>
<td>asynchronous I/O (during backup or recovery)</td>
</tr>
<tr>
<td>AsynOpen</td>
<td>opens an I/O port for backup or recovery</td>
</tr>
<tr>
<td>AsynWaitRead</td>
<td>waits for an I/O operation to end, then read (backup or recovery)</td>
</tr>
<tr>
<td>AsynWaitWrite</td>
<td>waits for an I/O operation to end, then write (backup or recovery)</td>
</tr>
<tr>
<td>Command reply</td>
<td>delivers a result to the application</td>
</tr>
<tr>
<td>Command wait</td>
<td>task is waiting for a new request</td>
</tr>
<tr>
<td>Connect wait</td>
<td>task is free for a new session</td>
</tr>
<tr>
<td>DcomObjCalled</td>
<td>a DB-procedure or a COM-object is currently executed</td>
</tr>
<tr>
<td>Diaginit</td>
<td>initialises the database internal trace files</td>
</tr>
<tr>
<td>Inactive</td>
<td>task is in initial state and has no resources yet</td>
</tr>
<tr>
<td>InsertEvent</td>
<td>creates an event</td>
</tr>
<tr>
<td>IO Wait (R)</td>
<td>waiting for I/O (R=read)</td>
</tr>
<tr>
<td>IO Wait (W)</td>
<td>waiting for I/O (W=write)</td>
</tr>
<tr>
<td>IO2 Wait (R)</td>
<td>waiting for I/O for mirrored disk (log only)</td>
</tr>
<tr>
<td>IO2 Wait (W)</td>
<td>waiting for I/O for mirrored disk (log only)</td>
</tr>
<tr>
<td>Locked</td>
<td>task is locked during kernel shutdown (to prevent rescheduling)</td>
</tr>
</tbody>
</table>
### x_cons: Task States (2)

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not Connected</td>
<td>brief wait, continues automatically</td>
</tr>
<tr>
<td>RescheduleMsec</td>
<td>immediately runnable</td>
</tr>
<tr>
<td>Runnable</td>
<td>running, using CPU time</td>
</tr>
<tr>
<td>Running</td>
<td>suspended by kernel and waiting to proceed running</td>
</tr>
<tr>
<td>Stopped</td>
<td>task or database session has been canceled</td>
</tr>
<tr>
<td>Terminated</td>
<td>task state unknown</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td></td>
</tr>
<tr>
<td>Vattach</td>
<td>opens I/O ports (volumes, normal operation)</td>
</tr>
<tr>
<td>Vbegexcl</td>
<td>waiting for protected memory access</td>
</tr>
<tr>
<td>Vblockio</td>
<td>runnable after protected memory access</td>
</tr>
<tr>
<td>Vdetach</td>
<td>closes I/O-ports (volumes, normal operation)</td>
</tr>
<tr>
<td>Dual Vector I/O</td>
<td>performs a vector-I/O-operation on two volumes in parallel</td>
</tr>
<tr>
<td>Vendexcl</td>
<td>leaving a protected area</td>
</tr>
<tr>
<td>Enter ExclLock</td>
<td>waiting to access a protected region with an exclusive lock</td>
</tr>
<tr>
<td>Enter ShareLock</td>
<td>waiting to access a protected region with a share lock</td>
</tr>
<tr>
<td>Task State</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Leave ExclLock</td>
<td>leaves a protected region</td>
</tr>
<tr>
<td>Leave ShareLock</td>
<td>leaves a protected region</td>
</tr>
<tr>
<td>ShutDown</td>
<td>database is shut down (changing state from ONLINE to ADMIN)</td>
</tr>
<tr>
<td>Connect close</td>
<td>ends the database session</td>
</tr>
<tr>
<td>Vsleep</td>
<td>brief wait, continues automatically</td>
</tr>
<tr>
<td>Vsuspend</td>
<td>suspended and waiting to be explicitly activated by another task (e.g. for B*-Tree locks (very brief) or log I/O)</td>
</tr>
<tr>
<td>Vvectorio</td>
<td>performs a vector-I/O-operation (reading or writing)</td>
</tr>
<tr>
<td>Vwait</td>
<td>waiting to be explicitly activated by another task (e.g. waiting for an SQL-lock)</td>
</tr>
<tr>
<td>WaitForEvent</td>
<td>waiting for an event</td>
</tr>
<tr>
<td>Yielding</td>
<td>Briefly cedes control of CPUs during Busy Waiting</td>
</tr>
</tbody>
</table>
x_cons Task Detail

Kernel Parameter: UseExtendedTimeMeasurement=YES
x_cons <dbname> show t_c t<task_index>

--- T25 User (pid = 23163) ---
remote_node: myserver
dispatcher-cnt: 127292
total_excl-cnt: 9110558
dev_write_io: 19
dev_write_pg: 19
state_vwait: 11
state_vsusp: 682
csv_rpl_count: 2296
csv_rpl_long: 46
rpl_rcv_count: 2296
dev_que_len_0: 18
dev_que_len_1: 0
dev_que_len>1: 0
remote_pid: 23163
command-cnt: 30477
self_susp-cnt: 433
avg_dev_wr_tm: 0.0895
avg_vwait_time: 4.1446
avg_vsusp_time: 0.0684
avg_rcv_rpl_t: 0.1077
avg_rpl_rcv_t: 0.0222

#Statements > 1 second
Avg. I/O time for
Dev process

---
x_cons <dbname> show t_c t<task_index> displays highly-detailed measurement values for individual database tasks. In this way, you can, for example, monitor the DB activity of an application while it remains connected to a database task (no permanent release/connect).

As of version 7.8 the database should run with the default setting UseExtendedTimeMeasurement=YES. The kernel collects time values of most wait situations. Shared SQL stores time values belonging to SQL statements.

Much of the output of the 'show t_c' function was developed exclusively for developers, however, some of the values are of more general interest in special situations.

dispatcher-cnt: Count of how often the task passed control to the UKT dispatcher, because it could not run, its time slot had expired, or another task was prioritized.
total_excl-cnt: Number of region accesses
command-cnt: Communication count between application and kernel
self_suspend-cnt: Number of task suspensions in which the task remained executable but still gave up control
<dev/self>_<read/write>_io: Number of I/Os via UKT (self) or DEV threads (dev)
<dev/self>_<read/write>_tm: Duration of an I/O via UKT (self) or DEV threads (dev)
state_vwait: Number of waits on SQL locks
avg_vwait_time: Average wait time for an SQL lock
avg_rcv_rpl_t: Average processing time of an SQL statement in the database kernel
rcv_rpl_long
second

Number of SQL statements with a processing time of more than one second
The command `show io` displays the number of read and write operations per volume as well as the number of 8 KB pages read. These numbers are independent of whether the I/O are synchronous or asynchronous.

`Show dev_io` displays the number of read and write operations of the I/O threads and the average I/O times.
Much of the data generated with x_cons is also accessible through tables. Thus this performance data can also be displayed by other tools (SQLStudio, SAP WebAS->DBACockpit).

The columns of the respective tables largely correspond to those of the database console.
Performance tables: Task Activities (1)

**SYSMON_TASK**
- Shows all tasks
- analog to „x_cons <DBNAME> show tasks“

**SYSMON_US**
- shows all User Tasks
- analog to „x_cons <DBNAME> show tasks us“

**SYSMON_DW**
- shows all DataWriter Tasks
- analog to „x_cons <DBNAME> show tasks dw“

**SYSMON_SV**
- shows all Server Tasks
- analog to „x_cons <DBNAME> show tasks sv“
Performance tables: Task Activities (2)

SYSMON_ACTIVE_TASK / SYSMON_RUNNABLE
- shows all active tasks
- analog to „x_cons <serverdb> show [active|runnable]“

SYSMON_USACTIVE / SYSMON_US_RUNNABLE
- shows all active User Tasks
- analog to „x_cons <serverdb> show [active|runnable] us“

SYSMON_DW_ACTIVE / SYSMON_DW_RUNNABLE
- shows all active DataWriter Tasks
- analog to „x_cons <serverdb> show [active|runnable] dw“

SYSMON_SV_ACTIVE / SYSMON_SV_RUNNABLE
- shows all active Server Tasks
- analog to „x_cons <serverdb> show [active|runnable] sv“
The task manager in transaction DB50 displays all database tasks and their current status. The system displays an overview of the database tasks and information about the current state of each individual task.

The following views are available: **Active Tasks, Executable Tasks, User Tasks (task type User), System Tasks, All Tasks**.

We use the task manager to analyze the following:

- For this database MAXCPU is set to 2. Thus the database can use 2 CPUs in parallel. Task T168 is running in another UKT (see Thread Overview, thread ID:19074) than task T120 and T163 (thread ID:19075). Tasks T120 and T168 can both have the Running status.

- We see a command (T163) that reads data from the disk to the cache - IO-WAIT (R).

- The task manager show the SQL statements executed by the sessions as of MaxDB version 7.8.

In the Application column we see the process ID of the work process and via the Application Server column we see the SAP application server.

With transaction SM50, we can identify the application that caused the long-running command using the application PID (20296).
The task manager can stop running SQL commands and terminate sessions.

With the task manager it is possible to terminate the respective task (T163) directly on the database level. The information in the process overview can then be used to examine the application for possible programming errors or missing indexes.
Database Analyzer
Consider a Typical Problem ...

Customer

... is reporting performance issues he thinks are database related

Support

... analyses the situation

- configuration? (caches, MAXCPU…)
- collisions? (SQL/BD locks, regions …)
- strategies? (used strategies, bad indices, current statistics, …)
- I/O system? (log / data accesses?)
- …

... gathers data from system tables / x_cons

- tedious work
- time consuming
CSN note 530394 describes bottleneck analysis with the Database Analyzer.

The DBAnalyzer is available as of version 7.3.00 Build 25.

Enhanceability

The logic and rules for monitoring with the Database Analyzer are defined by way of a configuration file (ASCII text). In case of changes or enhancements, you only have to change the configuration file in the directory INSTROOT/env.

Release independence

As accesses to the system tables are defined in the configuration file, adjustments for new releases only require adjusting the configuration file. Consequently, this is release-independent, but the Database Analyzer itself is not. The configuration file takes account of the instance type (OTLP/LVC).

Remote capability

The Database Analyzer uses only system tables. The data generated by “x_cons” can be queried via the SYSMON_..., system tables, which means they can be called remotely (e.g. via OSS).
Database Analyzer

For routine monitoring of database operation in the production system, an interval of 15 minutes (-t900) is adequate. For short-term monitoring of database operation, a measuring interval of 10-30 seconds is recommended.

If class W3 warnings occur frequently, you should certainly try to remove the bottleneck. W3 warnings generally indicate that the runtime behavior of the database is severely compromised. If you suspect poor search strategies (rows read/rows qualified), a more precise analysis is unavoidable. Shared SQL and the command monitor are available for this purpose.

Not all Database Analyzer outputs are necessarily caused by actual bottlenecks. For example table scans can be useful in certain situations, long runtimes of statements can automatically occur with large datasets etc.
Configuration File: dbanalyzer.cfg

- Describes the data to be collected or calculated (parameters). These parameters are either taken from the database (system tables) or calculated from the data taken from the database. As the manual evaluation of parameters is time-consuming, the Database Analyzer formats the logged data.

- Describes the evaluation rules (monitors) for the parameters. The monitors have up to four conditions (Information and Warnings 1 through 3) and are logged in a way that takes account of the conditions. For logging the monitors, in the configuration file you can store a verbal assessment or even concrete instructions for the user.
Up to four conditions for triggering the monitor. Conditions are boolean expressions that refer to parameters.

The top-level message is stored in the label. The label is an expression that is calculated when the monitor is activated. This enables references to the parameters.

User-selected texts for Description and UserAction.
General warnings on

- low cache hitrates (data-/catalog-cache)
- high I/O rate
- low hitrates on Selects, Updates und Deletes (ratio found/read rows; optimizer strategy)
- log queue filling level too high / overflows
- lock list escalations
Task specific warnings on

- poor I/O-times
- high lock waits (vwait/vsuspend)
- long command runtimes (receive/reply)
- high read activity (reads)
- a Usertask blockades in a certain state (e.g. Vwait, Vbegexcl… )
Database Analyzer: Program Start

Calling the Database Analyzer
- from a UNIX- or DOS-Shell
  - start: dbanalyzer
  - -n <server>
  - -d <database>
  - -u <user,pwd>
  - -f <configfile>
  - -t <interval>,<number>
  - -o <outputdir> -c <level>
  - stop: dbanalyzer
  - -n <server> -d <database> -u <user,pwd> -f <configfile> -o <outputdir> -stop
- with the DBMCLI command dban_start

- per WebAS
  - manually via DBACockpit ->Performance->Database Analyzer
  - implicit start with SAP WebAS 6.20 Basis SP 37
  - using the SAP CSS Support connection (SAP DB Connection -> SAPDBCON)
  - Enables SAP support to collect and store data on a host of their choice

You can also call the Database Analyzers with the DBMCLI command dban_start. The Database Analyzer is then implicitly started in the background. The Database Analyzer call can be supplemented with various options.

-n <server>
Name of the computer on which the database instance is running. If you enter this argument, you have to specify a directory for logging with the -o switch.

-d <database>
Name of the database instance that is to be examined.

-u <user,pwd>
User name and password for authorization on the database server.

-f <configfile>
Indicates the name of the configuration file to be used. The standard setting specifies the file dbanalyzer.cfg in the directory $INSTROOT/env.

-t <interval>,<number>
Defines the time interval (in seconds) between two evaluations. If <number> is specified, the Database Analyzer ends automatically when it has reached the specified number.

-o <outputdir>
Specifies the directory in which the log files of the Database Analyzer are written. If you specify -n <server> at the time of the call, you also have to specify a log directory. If you fail to specify a log directory, logging is done in the RUNDIRECTORY of the database instance in the subdirectory analyzer.

-c <outputlevel>
Specifies that Database Analyzer output also be written to the console. In the standard setting, no output is written to the console. With <outputlevel> you can specify how much is to be output. The possible values are 1, 2, 3 and 4.

-i Deletes (initializes) any pre-existing log files. This enables the logging of data from different databases in the same directory, which is otherwise prohibited. The data of the previously analyzed database are deleted in the process.
For routine monitoring of database operation in the production system, an interval of 15 minutes (-t900) is adequate. Logging should be activated with -p to obtain a retrospective overview of DB activities. For short-term monitoring of database operation, a measuring interval of 10-30 seconds is recommended.
As of support packages 6.20 SP37, the Database Analyzer starts automatically when the SAP WebAS system is started.

You can call the Database Analyzer from transaction DBACockpit ->Performance->Database Analyzer. You can also stop and restart it from there.

The default time interval for determining measurement data is 15 minutes. You can override this configuration stopping and restarting the Database Analyzer.

Each time the Database Analyzer is started, information about the configuration and performance-relevant data from system tables is output, including, for example, the number of tables that require an Update Statistics. You can determine the table names with a Select on the system table sysupdstatwanted.

Detected bottlenecks are output in text form to rapidly provide database administrators with an overview of the possible causes of performance problems. The analysis can be performed just once or at regular intervals.
Under *Expert Analysis* you can view into the logs of a particular day.

Logs are implicitly deleted periodically via the program *RSDBANCONTROL*. You can configure how long logs are kept using transaction DB59 in the integration data for the respective system. (6.20 as of basis SP 37).
You can build aggregates on a daily, weekly, monthly or quarterly basis for the journalized data. Data can be prepared furthermore by the list viewer building sums, min, max and average values, can be loaded to the local desktop or graphically displayed.
The Database Analyzer writes snapshot data like the running commands and the task activities (x_cons show active) into log files as of version 7.8.
Storing performance data in the logs is useful when checking runtime behavior later.

The collected data is stored as "csv" files in the directory/YYYYMMDD specified with "-o".

If you start the Database Analyzer on the DB server, you can omit the "-o" entry.
In that case, logging is done in the run directory/YYYYMMDD

A directory contains the data from one day.

The data is grouped by contents and stored in different files. You can display the day in a table with MS Excel and from the WebAS.
DBAN.prt
- quick overview; records monitor data including all rule based values

DBAN_BACKUP.csc
- physical reads/writes for backup, read/write time (ms) for backup

DBAN_CACHES.csv
- accesses, successful, failed and hit rates of all caches (DATA, CATALOG,...)

DBAN_FILLING.csv
- database filling level (size, permanently/temporarily occupied...)

DBAN_IO.csv
- virtual/physical reads/writes (common, permanent, temporary, long)

DBAN_LOAD.csv
- accesses / selectivity of selects and fetches, inserts, updates, deletes
Database Analyzer
Log Files (3)

DBAN_LOGGING.csv
- number of actual log writes, log queue overflows, max log queue used

DBAN_OVERVIEW.csv
- summarizing the other protocols key points

DBAN_REGIONS.csv
- Region accesses, collisions, waits and dispatches

DBAN_SPINLOCKS
- spinlock collisions, read/write locks

DBAN_STRATEGY_INDEX.csv
- accesses / selectivity of index, index ranges and isolated index / index ranges

DBAN_STRATEGY_PRIMKEY.csv
- accesses / selectivity of primary key and primary key ranges
Database Analyzer
Log Files (4)

DBAN_STRATEGY_SCANS.csv
- accesses / selectivity of table and isolated index scans

DBAN_TASK_ACTIVITY.csv
- SQL commands, task statistics (active, running, runnable…)

DBAN_TASK_IO.csv
- I/O number / duration for logwriter, user und datawriter Tasks

DBAN_TASK_STATES.csv
- number and elapsed time of processed commands
- number and used time in task states Vsuspend, Vwait, Vsleep

DBAN_TRANSACTIONS.csv
- number commands, prepares, executes, commits, rollbacks, subtrans, lock request timeouts and lock request escalations
Database Analyzer
Log Files (5)

RUNNING_COMMANDS.prt
- Running SQL commands at the time of the data collection

SHOW_ACTIVE.prt
- Active tasks at the time of the data collection

DBAN_UKT_CPU_UTILIZATION.prt
- CPU Usage of the User Kernel Threads at the time of the data collection

DBAN_USER_TASKS_CMDS_EXECUTED.prt
- Dispatcher Counts per Usertask serving a session
Via transaction DB59 -> **Integration Data**-> **Automatic Monitoring**, you can define the time interval at which Database Analyzer logs are deleted.

By default, the logs are stored for 93 days.

The corresponding information in the database table SDBCCMS, however, is kept for 15 weeks. For more information, see note **530394**.

You can make your own personal settings by choosing Display/Change.
Database Analyzer: Data Cache

Low data cache hit rate: 
<percentage> % <number> accesses, <number> successful, <count> unsuccessful

Cause:
- data cache too small
- SQL statements creating a lot of page reads (unselective commands, missing indices)

Action:
- Finding cause, e.g. with the Diagnose Monitor and pay attention to further Database Analyzer messages
- If nothing indicates an application or design problem: increase cache size to reduce risk of I/O sequentialization

Database Analyzer: Data Cache

Low data cache hit rate : <percentage> %

<number of> accesses, <number> successful, <number> unsuccessful

Explanations

The hit rate is too low when accessing the database cache. The data cache hit rate for a running database application should not be less than 98%; otherwise, too much data has to be read physically. For a short time, lower hit rates may occur; e.g., when reading tables for the first time, or when the table does not fit into 10% of the data cache with repeated table scans (only with UseDataCacheScanOptimization/LRU_FOR_SCAN = NO). Data cache hit rates under 98% for intervals of 15 minutes or more must be avoided.

User response

In addition to enlarging the data cache (note the paging risk in the operating system), search for the cause of the high read activity. Frequently, individual SQL statements cause a high percentage of the total logical and physical read activities. Enlarging the cache only transfers the load from the disk to the CPU although an additional index, for example, could transform a read-intensive table scan into a cheap direct access.
Database Analyzer: cache displacements

Cache displacements: <number of> pages/second

Explanations

Modified pages are displaced from the data cache to disk because the data used by the applications cannot be completely kept in the data cache. If the size of the data cache were sufficient, the physical write would be delayed until the next SAVEPOINT and then be done asynchronously. Cache displacements result in synchronous I/O and should be avoided, if possible.

User response

Enlargement of the data cache. Particularly with larger data imports, the so-called pagers should be activated for regular asynchronous buffer flushes between the SAVEPOINTS database parameter DataCacheIOAreaSize, DataCacheIOAreaFlushThreshold, DataCacheLRUAreaFlushThreshold or in earlier versions _DW_IO_AREA_SIZE, _DW_IO_AREA_FLUSH, _DW_LRU_TAIL_FLUSH).
Database Analyzer: selectivity

Explanations

The relationship between read and found (qualified) rows is poor for a certain access strategy applied by the MaxDB Optimizer. This indicates a poor search strategy, caused either by the application (missing or insufficient indexes) or by poor formulation of SQL statements. Searching large quantities of data can seriously compromise the performance of the system as a whole due to the numerous negative effects (I/O, CPU load, etc.).

User response

First of all, see if MaxDB Optimizer is able to find a more suitable strategy after updating the internal database statistics. The update should be done directly from the SAP system with transaction DB13.

If this does not produce the desired result, search for the statement that triggers the unfavorable search strategy. The easiest way to do this is with the data in the Shared SQL Cache or with the Command Monitor.
As of version 7.8 MaxDB collects runtime data for the known SQL commands. These figures are available if Shared SQL is turned on and the parameter UseExtendedTimeMeasurement has the value YES.

The runtime data collection has a small impact on the performance of the system. The statement text is available in Shared SQL and doesn’t need to be stored in specific monitoring tables.

The Resource Monitor in the DBACockpit works downward compatible. It shows the data from the diagnose analyze tool in older version and the data from Shared SQL with MaxDB 7.8 and newer versions.

The runtime data includes information about number of executions, overall – minimum – maximum and average execution time, number of page accesses in memory and on disk, number of read and qualified rows, wait situations and more.

Shared SQL collects the execution times in microseconds. The Resource Monitor shows the execution times as milliseconds which is incorrect in the current version of the DBACockpit.

The Shared SQL data helps analyzing the overall load in the systems. The Resource Monitor filters the commands to be displayed by the output criteria.

This example shows a select reading data from table E071K. It runs quite often with a high execution time. It reads all records but doesn’t find any according to the WHERE condition. An optimization of the application or an index on table E071K for the field FLAG could reduce the load in the system significantly.

A double-click on the command guides you to the complete statement. The Resource Monitor can jump to the table definition and to the ABAP program. The DBACockpit supports the explain command in the Command Monitor. MaxDB needs the values of the input parameters for the explain. Shared SQL doesn’t store input value parameters of the single executions.
A select from the view with the condition "WHERE CURRENTEXECUTECOUNT > 0" shows all currently running commands.

A select from the view SYSINFO.COMMANDSTATISTICS returns the runtime values stored in the Shared SQL Cache. The column CURRENTEXECUTECOUNT shows the number of sessions currently executing the command.

The view SYSINFO.COMMANDSTATISTICSRESET show the runtime values of commands executed after a reset. The SQL command “diagnose analyze clear all” performs the reset.

The following SQL statement shows all command executions after a reset:
```
select c.statement, r.* from commandstatisticsreset r, commandstatistics c where c.commandid = r.commandid order by r.executetime desc
```
The Command Monitor finds long and/or expensive SQL command executions. Different input parameter values can lead to very different execution times. MaxDB can use different search strategies according to the input parameter values.

The Command Monitor stores the command ID, the monitoring values and the input parameter values when a command execution exceeds (page accesses and execution time) or falls below (selectivity) the given filter value.

The transaction DBACockpit and the Database Studio can run an Explain with the logged command using the input parameter values. Explain shows the search strategy of the statement.

The DBACockpit and the Database Studio use SQL statements to enable the Command Monitor and to display the results of the monitoring. Administrators can use those SQL statements manually, as well.

The Command Monitor has been re-implemented with MaxDB version 7.8. It uses commands stored in the Shared SQL Cache. It doesn’t log the statement text anymore but only the command ID known by Shared SQL. This reduces the memory footprint of the Command Monitor significantly.

Furthermore the new implementation has a very small impact on the performance of the system as long as users define reasonable filter values.
Users can turn on the Command Monitor with the desired filter value in the Performance section of the DBACockpit. The DBACockpit displays the logged SQL commands with the runtime values.

A double click on the line of a command shows more detail information about the command.

The layout definition allows the selection of user defined output columns. Important columns are:

- **Table**: Table used by the statement
- **Program**: ABAP Program calling the SQL command
- **Routine**: Runtime of the SQL command in seconds
- **#P Accesses**: Number of page accesses in cache
- **#R Read**: Number of read records
- **#R Qualified**: Number of records matching the where clause
- **#P/R**: Number of page accesses per qualified record
- **#Fetched**: Number of records transported to the application
- **#Disk I/O**: I/O accesses from and to disk
- **SQL Waits**: Number of lock collisions
- **Task Suspends**: Number of internal memory access collisions
- **No. Fetch Orders**: Number of fetch order for record transportation
- **Result is Copied.**: YES: an internal result set was created
- **Date**: Execution date
- **Time**: Execution time
The Command Monitor collects the input parameter values belonging to the command executions. The DBACockpit inserts the values into the command text and executes an Explain.

Explain show the search strategy found by the optimizer at the time of the explain execution.

The DBACockpit can jump into the ABAP program calling the SQL command. It can also jump into the table definition view with the tables referenced by the SQL command. The table definition view shows the definition in the database, not the data dictionary definition of the WebAS.
EXPLAIN shows the search strategy of the command

- Search strategy at this time
- No SQL command execution
- Explain with kernel trace supported
- Explain with hints supported

The chapter Query Optimization provides more detailed information about the Explain command.
The database interfaces transport the name of the ABAP program and the call position with a prepare of a statement to the database. Shared SQL stores this information with the command.

The DBACockpit uses this information to jump directly into the ABAP program to the call position of the statement. Both, the Command and the Resource Monitor support this functionality.

ABAP SQL statements often look different to the commands sent to the database. The ABAP database interfaces generate the SQL commands depending on profile settings and the used database management system.
Resource and Command Monitor allow to jump from the statement to the table definition view. The DBACockpit here shows the definition and more detailed information about the chosen table. The table size, optimizer statistics and the table data can be displayed in this view. Optimizer statistics can be updated as well.
The DBACockpit provides further diagnose functions supporting users when analyzing the performance of a system:

- Caches
  - Cache sizes and usage, incl. hitrates
- Missing Tables and Indexes
  - Jump into transaction DB02
- EXPLAIN
  - Enter a command to get the search strategy
- Select-Editor
  - Input and execution of SQL commands
- Critical Regions
  - Accesses to synchronized memory areas (x_cons SHOW REGION)
- Database Console
  - x_cons output
- SYSINFO Views
  - Direct access to all views of the schema SYSINFO
- Database Objects
  - Show database object definitions and observe table growth
The DBACockpit can access to remote database. It can use the Command and Resource Monitor on databases without ABAP WebAS data.

The Database Studio supports SQL command analysis as well. This is very helpful if no DBACockpit is available. The Database Studio combines Resource and Command Monitor in one “SQL Performance Analysis” Editor. It allows the detailed definition of filter values for the Command Monitor in a WHERE condition.
The Database Studio shows all commands stored in the Shared SQL Cache. The commands are sorted by the execution time by default.

Different variants show groups of runtime figures. This insures a proper overview. The variant “Caller Details” show the commands logged by the Command Monitor only.

A double click on a statement line guides to the detailed statement view.
The view "Bottleneck Candidate Details" shows all single command executions collected by the Command Monitor. It shows the input parameter values and the Explain output according to the chosen command execution.

A right click on the statement opens a new SQL editor with this statement. It can open the table definition editor for marked tables as well.
The latest entry in the table SYSINFO.COMMANDMONITORCONSTRAINTS defines the filter of the Command Monitor. The WHERE condition uses any column of the table COMMANDMONITOR. It can use AND and OR conditions.

With this approach the Command Monitor as of version 7.8 can set more detailed filters than the monitor in older versions.

The performance impact of the Command Monitor depends on the number of command executions to be logged. A high number of logged commands can lead to high memory consumptions.

The Command Monitor settings remain after a database restart as of version 7.8; i.e. the table COMMANDMONITORCONSTRAINTS stores persistent data.
The Command Monitor stores all runtime figures, the timestamp and the SESSIONID of a command execution matching the filter criteria in the table SYSINFO.COMMANDMONITOR. The COMMANDID references to the COMMANDID in the view SYSINFO.COMMANDSTATISTICS.

The table SYSINFO.COMMANMONITORPARAMETERS stores the input parameter values belonging to the command executions together with the COMMANDID, execution number and the parameter position. This table can become big with many logged command executions specially if the commands have many input parameters. An SQL DELETE command can remove the records from the table.

This select joins the data from the relevant system tables and shows the command executions with their parameter values sorted by the execution runtime:

```sql
select m.executecount, c.statement, p.value, m.*
from sysinfo.commandstatistics c, sysinfo.commandmonitor m, sysinfo.commandmonitorparameters p
where c.commandid = m.commandid
and m.commandid = p.commandid
and m.executecount = p.executecount
order by m.runtime desc, m.executecount, p.inputparameternumber asc
```
As of version 7.5 MaxDB uses statistics data only for joins and single table selects with a result set limit like "WHERE ROWNUM <= n".

The Update Statistics collects the table size and number of rows only if they are not available in the File Directory. This can happen with tables created by older database version than 7.6.

Update Statistics collects and calculates statistic values for primary- and secondary key columns. It collects the statistic values for other columns if statistics already exists.

The database kernel inserts table names into the table SYSUPDSTATWANTED during command execution if it recognizes outdated statistics. The DBM command sql_updatestat_per_systemtable executes an Update Statistics for all tables logged in SYSUPDATESTATWANTED.

The pre-defined database procedure SYSDBA.SYSCHECKSTATISTICS checks the size of all tables in the given schema. It inserts table names into the table SYSUPDSTATWANTED if the actual table size differs to the size in the statistics by more percentage than the threshold.

The database procedure SYSCHECKSTATISTICS checks the size of all tables in the given schema. It inserts table names into the table SYSUPDSTATWANTED if the current table size differs from the statistics by more than the given threshold in percent.

The automatic update statistics function starts a DBM event process. The database kernel sends an event to the DBM process when it inserts a table name into the table SYSUPDSTATWANTED. The DBM process then starts the command sql_updatestat_per_systemtable.

A DBM event process receives an event from the database kernel about outdated statistics.

The DBM command sql_updatestat executes an Update Statistics for all permanent tables of the database.

The Update Statistics speeds up the read of the table records by using parallel I/O orders.
Sampling with Update Statistics

Sample rates for Update Statistics can be configured as

Rows per table:
- UPDATE STATISTICS ... ESTIMATE SAMPLE <n> ROWS

Percentage per table
- UPDATE STATISTICS ... ESTIMATE SAMPLE <p> PERCENT

Advantage of sampling:
- Shorter runtime of update statistic job

Disadvantage of sampling:
- Sample values are only estimated. If they do not resemble the actual data distribution, the optimizer might chose a suboptimal access strategy

Sampling with Update Statistics

Database statistics can be created on the basis of samples. The basis for the statistics can be either a number of rows of your choice or a percentage of the table. While the statistics are not exact, there are generally sufficient for a correct calculation of the SELECT strategy since this depends less on precision than on distinguishing between selective and non-selective columns.

Especially when creating an additional index for an inefficiently processed SQL command, the selectivity of all columns of a table can be determined relatively quickly using 'UPDATE STATISTICS COLUMN (*) ESTIMATE SAMPLE 20000 ROWS'. The selectivity of a column is an important criterion when selecting index columns.

50,000 rows have been proven as adequate sampling quantities for column statistics.

As of version 7.6, the sampling procedure in the standard uses a new algorithm for calculating the statistics data. You can determine the algorithm to be used with the parameter UPDATESTAT_SAMPLE_ALGO. The new algorithm generates more accurate statistics with fewer records read.
Thank you!