SAP® MaxDB™ Expert Session

SAP® MaxDB™: Introduction into I/O concept 7.8

Heike Gursch May 13, 2014

Public

SAP® MaxDB™ – Expert Session

Introduction into I/O Concept of SAP ® MaxDB™ (7.8.)

Heike Gursch
Christiane Hienger
IMS MaxDB/liveCache Development Support
May 13, 2014

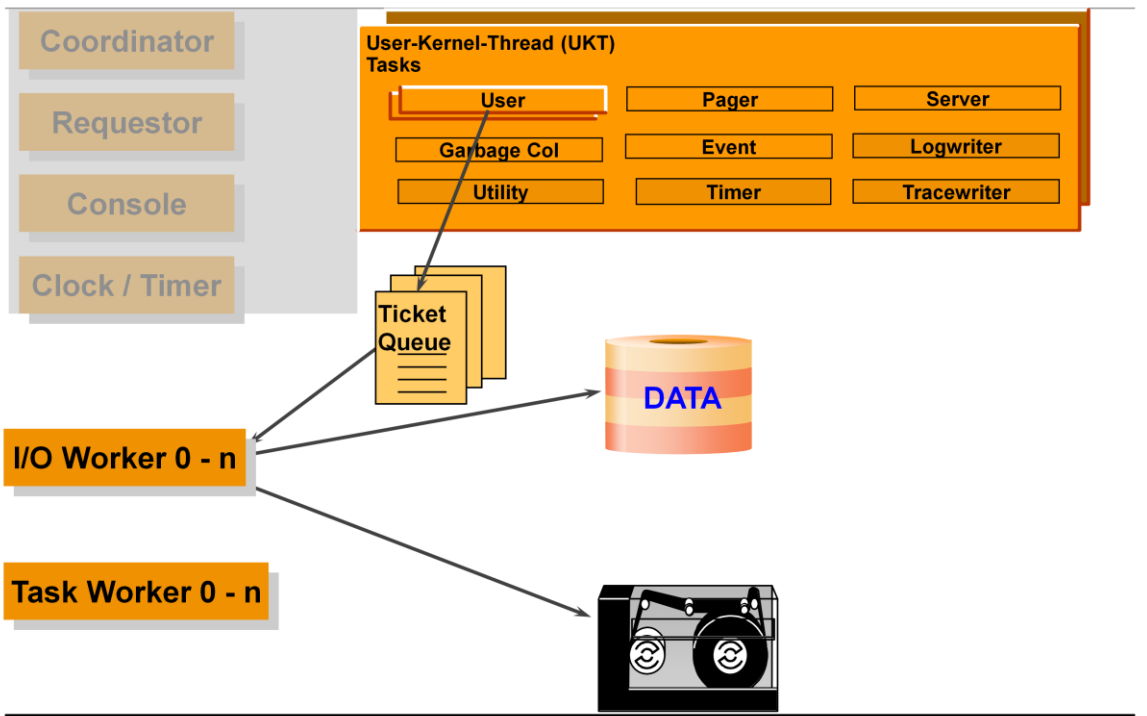
Agenda

- Introduction (Basic Parameters)
- Motivation for change of I/O concept
  - Disadvantages of Legacy I/O
- Transition to current concept
- I/O concept in Detail
  - General
  - Parameters
  - Cluster (Parameters)
  - Prefetch Mechanisms - Asynchronous Read (Parameters)
  - DB-Analyzer logfiles
  - x_cons output
  - System tables

Agenda

1. **Introduction** (Basic Parameters)
2. Motivation
3. Transition to Current Concept
4. I/O Concept in Detail
5. Q&A and Outlook

I/O Worker Threads



I/O threads are responsible for processing the write and read requests to and from data and log volumes that are requested by the corresponding tasks. MaxDB supports asynchronous I/O requests.

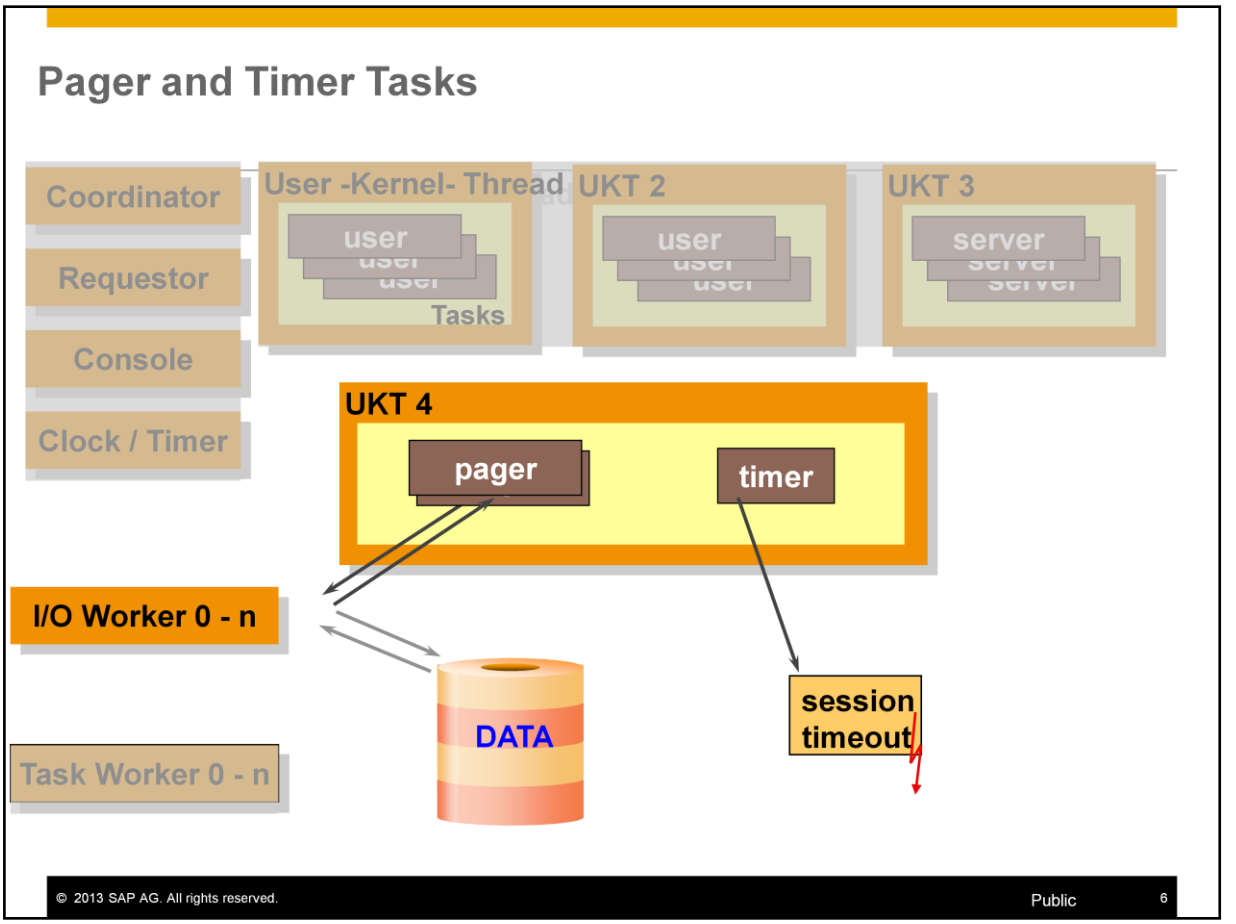
Until version 7.6 the number of **I/O threads** is primarily dependent on the number of volumes in the database instance. As a rule, two **I/O threads** are activated for each data and log volume and one for the writing of the database trace

As of version 7.7 **I/Oworker threads** are taken from a pool and activated on request; I/O is done asynchronously. After finishing the I/O requests the workers are returned to the pool.

As of version 7.8 the user tasks utilize the asynchronous I/O for scans. The user tasks send several parallel I/O orders to the I/O systems. They don't wait until the I/O system has read every single block from disk.

The **Task Worker Threads** are not used for I/O requests. User tasks use task Worker threads to execute orders asynchronously e.g. in hot standby environment to send the log position to the standby node.

MaxDB uses an own I/O concept and does not use the concepts for aynchronous I/O of the operating system (Windows) any longer.



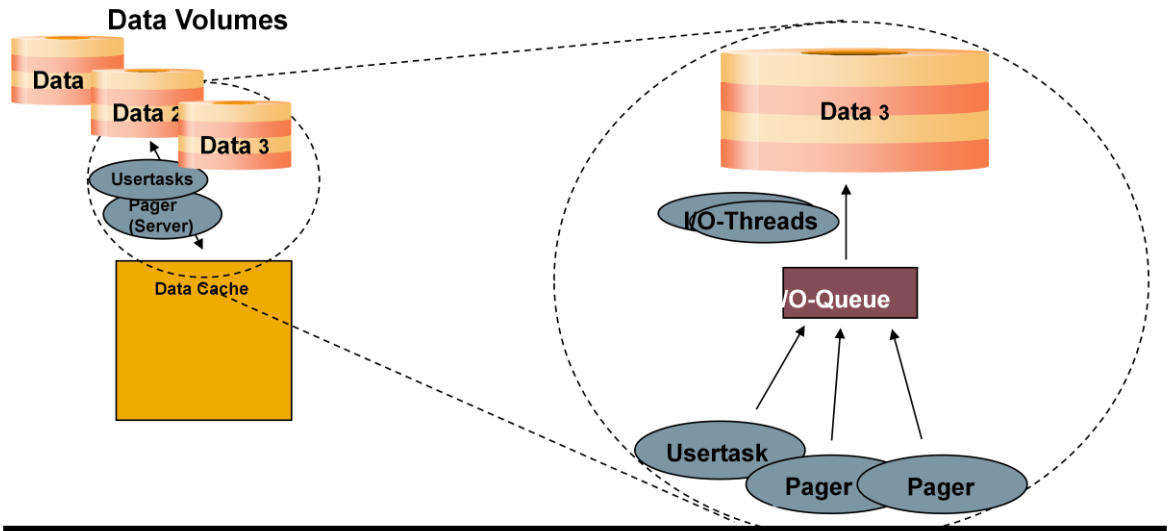
Pager tasks are responsible for writing data from the data cache to the data volumes. They become active when a savepoint is being executed.
The system calculates the number of pagers. It depends primarily on the data cache size and the number of data volumes.

The pager tasks use I/O worker threads to execute their I/O requests.

Asynchronous I/O

EnableSynchronousTaskIO (_USE_IOPROCS_ONLY)

The parameter specifies, if I/O may exclusively be done by special I/O threads or may also be done by the UKT itself.



UKTs can themselves call I/O operations if

- the parameter **EnableSynchronousTaskIO** is set to “YES" and
- only one user task in the UKT is not in "Connect Wait" status or only one task is running in a UKT (e.g. log writer)

Then the I/O request is not put into a queue and processed directly by the I/O thread.

The individual I/O operation can be executed more quickly if the UKT does not need to request an I/O thread.

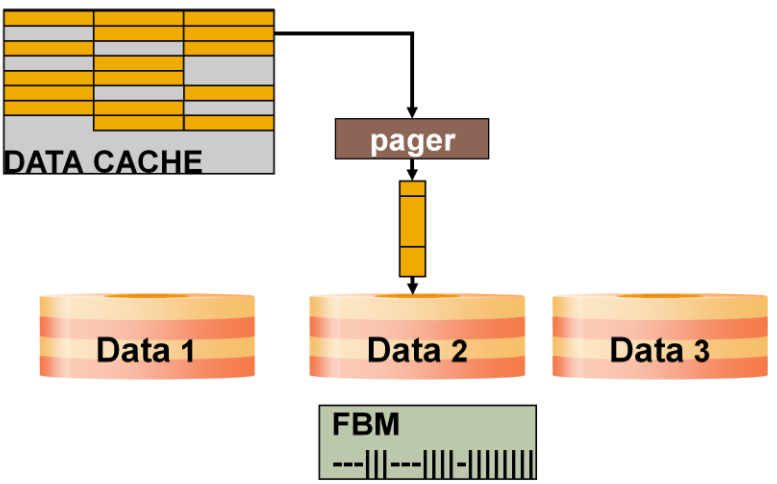
If a user task executes an I/O request by itself, other tasks cannot work until it is finished. The UKT is blocked and waits for the reply of the I/O request. This option can compromise performance in parallel operation.

Values: Default: YES
Online change: YES

Blocked I/O (data)

DataIOClusterSize (DATA_IO_BLOCK_COUNT)

Size (in 8 KByte pages) of an I/O operation by the pagers.



Pagers can combine data pages and write them with an I/O operation (vector I/O).

If for a table the cluster flag is switched on the database builds groups of pages that belong together according to the B* tree chains before writing it to the disks. Thus data pages are kept together logically to improve the use of prefetch algorithms of the storage systems during scan operations.

This parameter also influences the block sizes for read / write operations to data backups. Backup templates provide a mechanism to adapt block sizes used for writing to the backup media.

If the cluster flag is used for tables the block size for writing a backup should be set according to the **DataIOClusterSize** or should be a multiple of it. Otherwise the clusters would be dissected during restore.

Values: Default: 64
Min: 4, Max: 128
Online change: No

Blocked I/O (Log)

LogIOClusterSize

- Size (in 8 KByte pages) of a LOG I/O operation
- Usually set to 8 (upper limit 32)
- Windows 32 Bit : set to 4

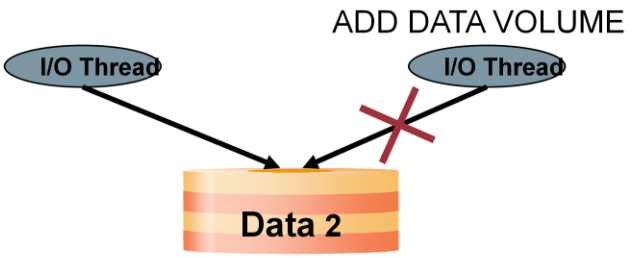
Lock on Attached Volumes

UseVolumeLock (SET_VOLUME_LOCK)

An enforced lock prevents an ADD DATA VOLUME with a volume that is already in use by this or another instance.

In case of NFS mounted volumes (e.g. using Network Attached Storage NAS) it may be reasonable not to set the lock.

Normally set; not set for hot standby systems and shared repository



From version 7.5, in the standard setting MaxDB sets a lock on open volumes that are in the file system. You can change this behavior using the parameter **UseVolumeLock**.

Values:

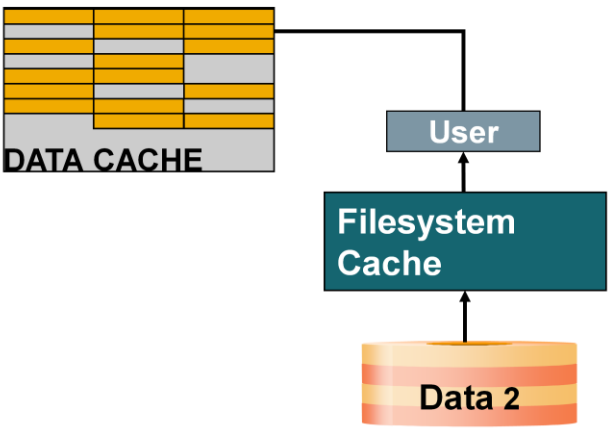
Default: YES The database requests a lock when a volume is opened.

Online change: NO

Bypassing the File System Cache

UseFilesystemCacheForBackup
UseFilesystemCacheForVolume (USE_OPEN_DIRECT)

Use of I/O flag O_DIRECT to by-pass filesystem cache



If the parameter **UseFilesystemCacheForVolume** is set to NO then the database opens the volumes by using the flag O_DIRECT, i.e. all volume read and write operations directly access the physical disks.

If it is set to YES a write-through-file-system-caching is enabled for volume I/O operations. Data is still immediately written to the disk but a copy is done to the file system cache.

If the parameter **UseFilesystemCacheForBackup** is set to NO then – for a backup - the database opens the volumes and backup media (in case of files) by using the flag O_DIRECT.

The database kernel cannot open the files in the volumes upon starting if one of the parameters is set to NO although the option O_DIRECT is not supported by the file system. The mount options should force direct I/O for the file system in those cases.

Please additionally have a look at note 993848 which gives recommendations concerning mount options for different file systems. Note 977515 describes file system behaviors during backups and provides special recommendations for the settings of these parameters.

Attention: By renaming the parameter it now got the inverse meaning.

Values:

UseFilesystemCacheForVolume

Default: NO
Online change: NO

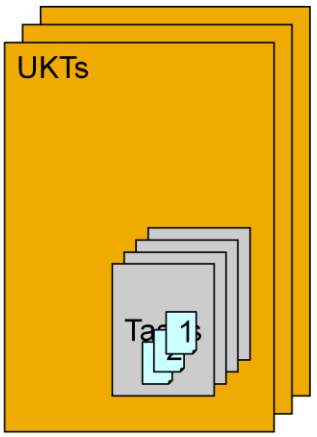
UseFilesystemCacheForBackup

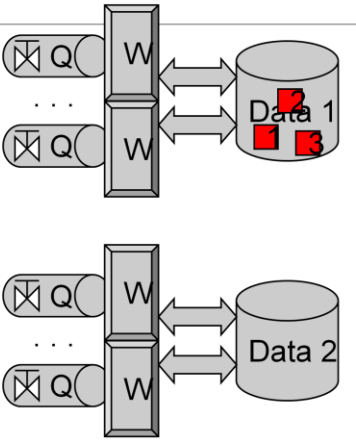
Default: YES
Online change: NO

Agenda

1. **Introduction** (Basic Parameters)
2. Motivation
3. Transition to Current Concept
4. I/O Concept in Detail
5. Q&A and Outlook

Legacy I/O Components in MaxDB <= 7.6





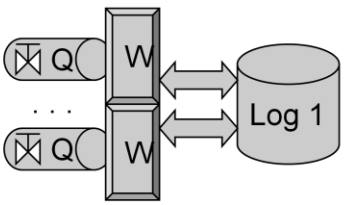
I/O Workers with Queues

Example:

Two queues per volume
(_IOPROCS_PER_DEV = 2)

Two requests in queue before switch to next
(_IOPROCS_SWITCH = 2)

Three jobs started in three tasks in parallel



For a better understanding of the current I/O concept we go back to the past and have a look at the former concept which was used up to version 7.6.

The I/O queues are associated with the data volumes. If **_IOPROCS_PER_DEV** is set to 2 there are 2 I/O threads per volume.

Animated slide: Three I/O requests have to be processed.

1.The first 2 I/O jobs are sent to the first queue as the parameter **_IOPROCS_SWITCH** is set to 2. I/O job 3 is sent to the second I/O thread.

2.I/O 1 and 3 can be completed at the same time; 2 is still waiting.

3.Afterwards 2 can be completed.

Legacy I/O: System Views

System views:

IOTHREADSTATISTICS / IOTHREADSTATISTICSRESET

- I/O worker statistics, per volume queue
- I/O counts
- I/O times (if enabled)

BACKUPTHREADS

- Statistics for backup workers, per backup

Legacy I/O: Parameters

_IOPROCS_PER_DEV (2)

- Number of I/O workers/queues per data volume
- Log volumes and backup media use each a single worker

_IOPROCS_SWITCH (2)

- Minimum request count in a queue before trying next queue

_IOPROCS_FOR_PRIO (0)

- Number of reserved workers for “high priority” tasks

_IOPROCS_FOR_READER (0)

- Number of reserved workers for read jobs

_USE_IOPROCS_ONLY (NO)

- If set to NO, task may do I/O directly, if alone in UKT

PREALLOCATE_IOWORKER (NO)

- If NO, I/O workers will be created on-demand, otherwise at startup

SET_VOLUME_LOCK (YES)

- Prevent attach to a volume from another kernel (NO for hot standby)

SIMULATE_VECTORIO (NEVER)

- Workarounds for buggy writev()/readv() implementation

USE_OPEN_DIRECT

- Open volumes with O_DIRECT flag to bypass buffer cache

New I/O: Reasons for Rewrite

Shortcomings of Legacy I/O

- Too many workers needed
- Lock-based operations
- Poor parallelism when accessing single volume
  (Solvable by adding more volumes and thus more workers)
- No support for asynchronous operations (i.e., at most one active job/task)
- Poor priority handling

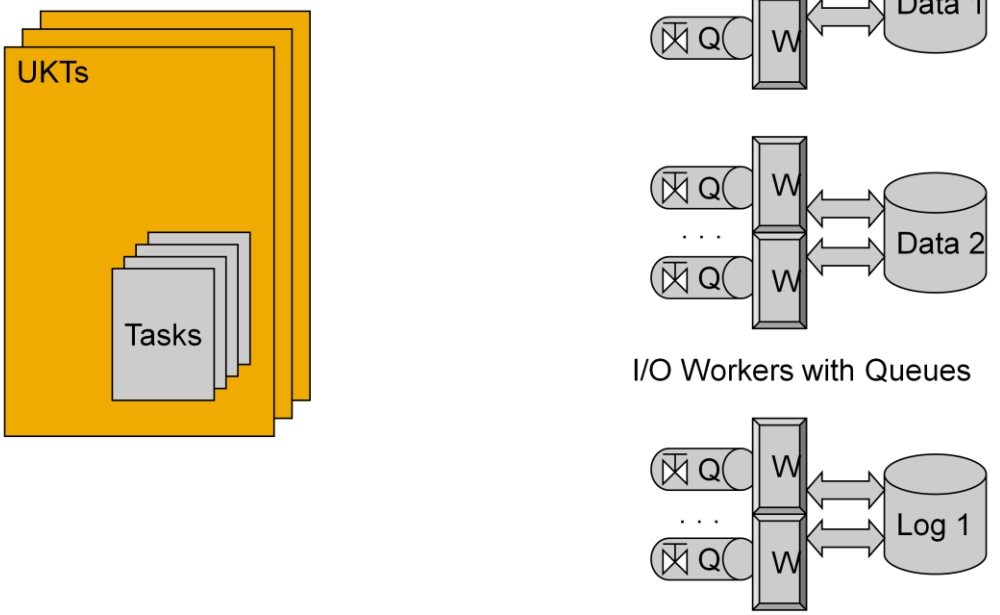
New I/O

- Different queuing mechanisms allowing for worker pool
- Lock-free operation
- Asynchronous operation
  Task-asynchronous operation (task continues running and dequeues the result later; more than one job can be active at a time)
  Completion port (central handler for certain types of I/O requests)
  Prepared for Unix aio usage
- Priority handling
- New console commands and system views to inspect I/O

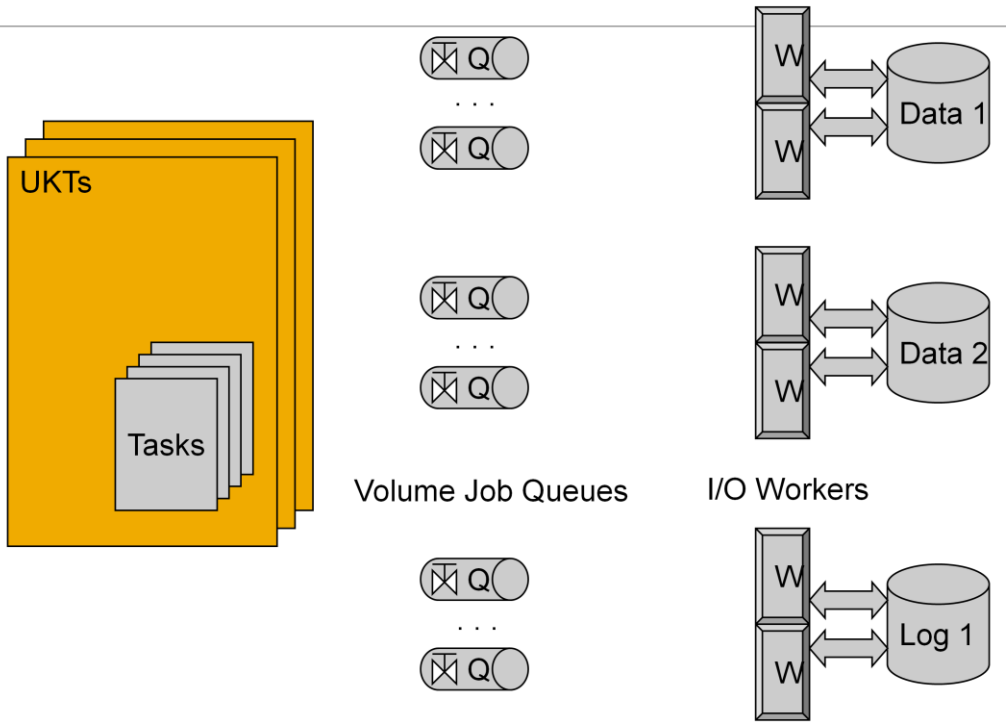
Agenda

1. **Introduction** (Basic Parameters)
2. Motivation
3. Transition to Current Concept
4. I/O Concept in Detail
5. Q&A and Outlook

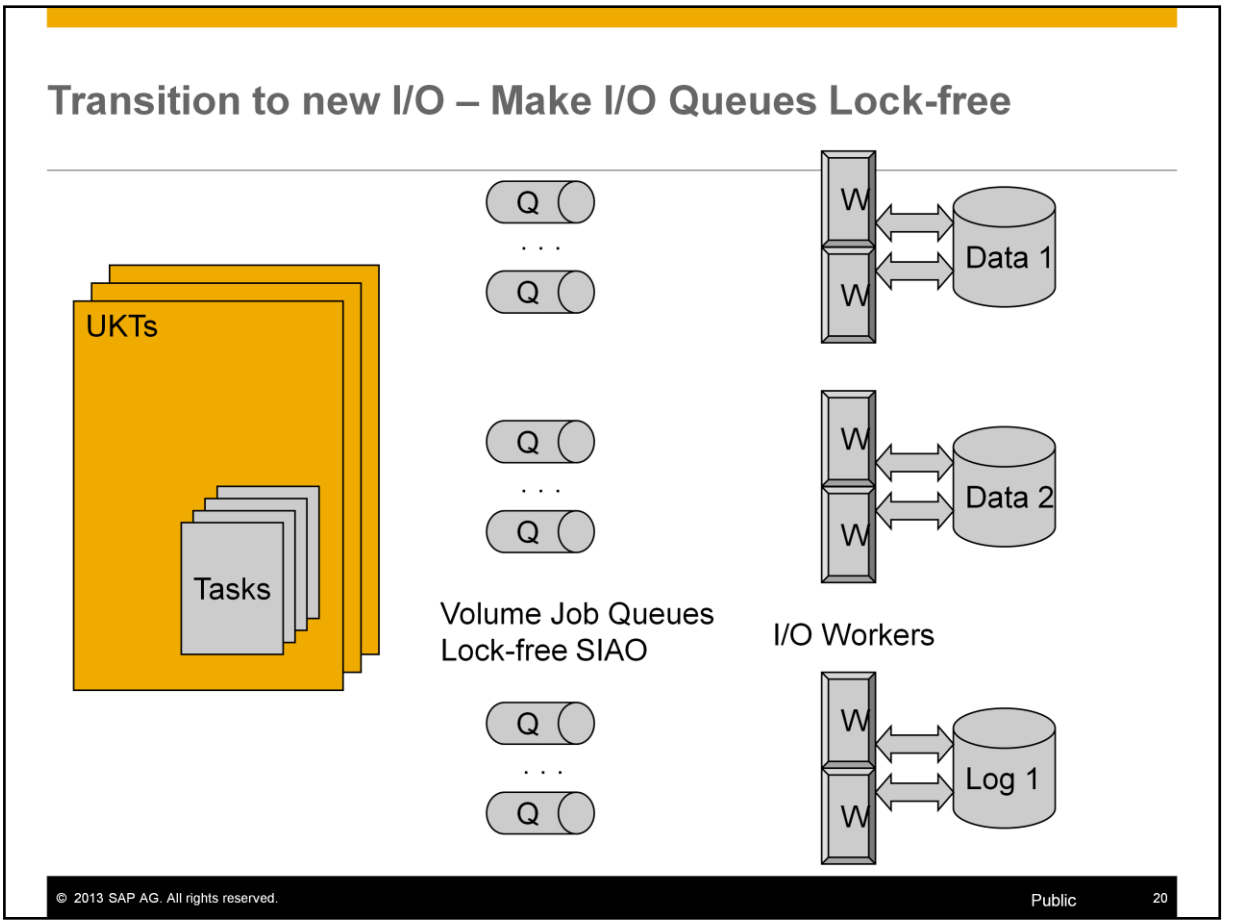
Legacy I/O Components in MaxDB <= 7.6



Transition to new I/O – Separate I/O Queues

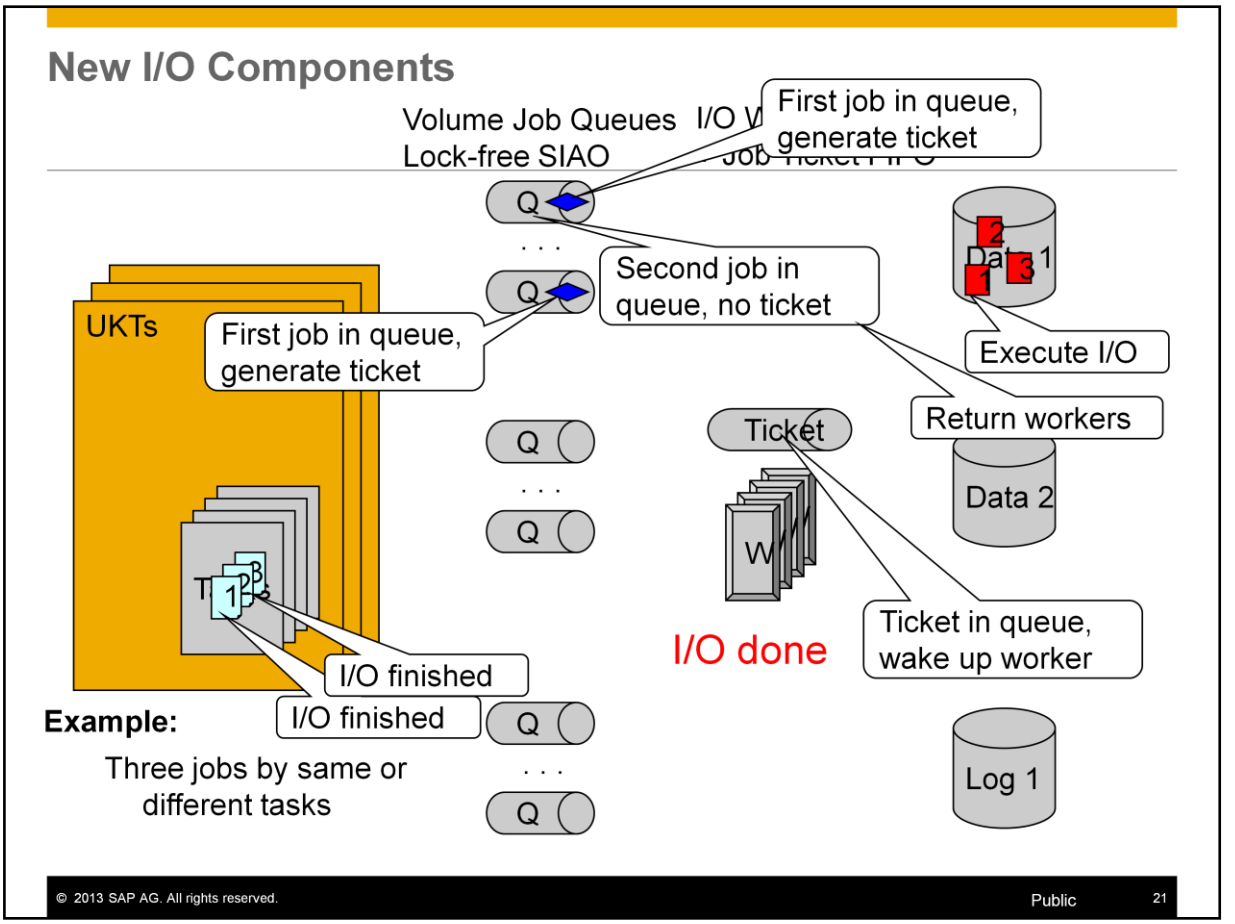


As the first measure the I/O queues were separated from the I/O workers.



The next step was to make sure that the I/O queues can work lock-free.

SIAO – single in all out



Animated slide: Three I/O jobs have to be processed by the same or by different tasks

1.The first job is sent to a job queue; a ticket is generated

2.The ticket is sent to the job ticket fifo and wakes up an I/O worker.

3.In the meantime a second I/O job is sent to the (same) queue; no ticket required. At the same time job 3 is put into another queue; generation of a ticket required.

4.I/O jobs 1 and 2 are sent to I/O worker; job3 creates a ticket

5.I/O for 1 is executed. For 3 an I/O worker thread is opened and 3 is sent to it.

6.The I/O of job1 successfully done in data area.

7.I/O 2 and 3 successfully done in data area.

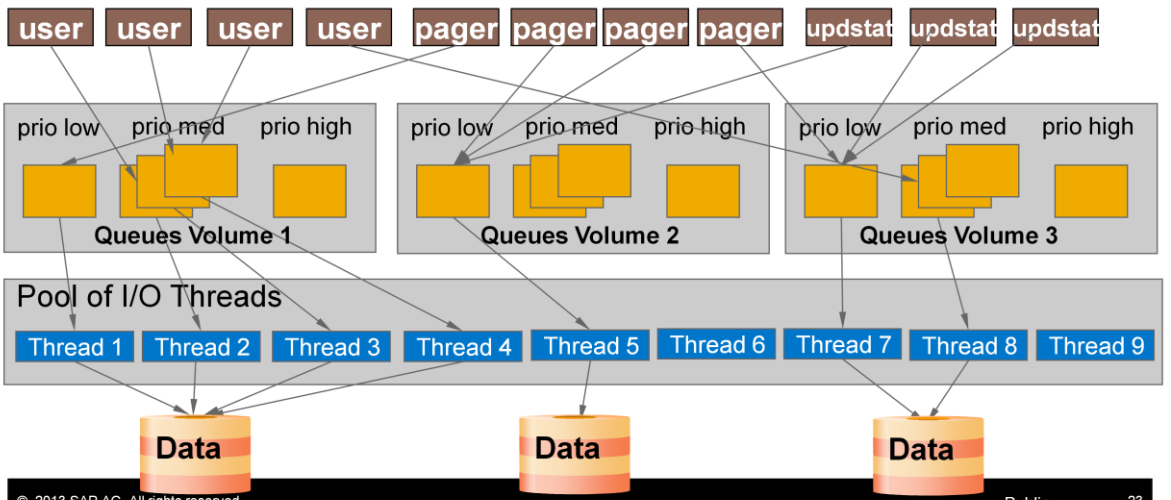
8.If there is no more request the I/O workers are returned to the pool.

Agenda

1. **Introduction** (Basic Parameters)
2. Motivation
3. Transition to Current Concept
4. I/O Concept in Detail
5. Q&A and Outlook

I/O Thread Implementation

Scalability of asynchroneous I/O has been improved in version 7.7 significantly. In older versions the I/O threads were directly associated with the volumes. As of version 7.7 I/O threads can send their requests to different volumes. There is a configurable number of queues per volume. It is possible to assign priorities to I/O requests. Tasks don‘t have to wait for the result of the I/O but can send the request asynchroneously and continue their work.



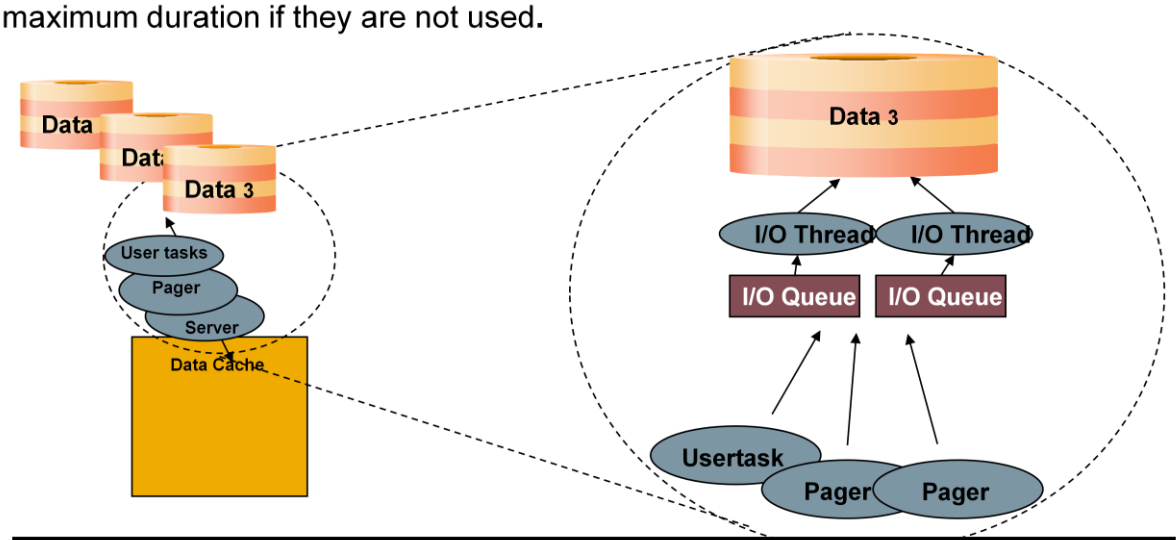
With version 7.7 the I/O interface to the operating system has been reimplemented. Version 7.7 uses different parameters than version 7.6. The new I/O system has the following essential advantages:

- No direct assignment of a I/O worker thread to a volume. This implies a better scalability of I/O.
- I/O worker threads can be started on request. This prevents the use of unnecessary resources.
- The synchronization of accesses to the I/O queues has been changed. The access is done collision free. This additionally improves the scalability of I/O.
- Prioritization of special I/O requests. Dedicated jobs within the database (f.e. CHECK DATA) can run with lower priority. Online operation is stressed less.
- Tasks can send I/O requests asynchroneously to the I/O system. They don‘t have to wait until the I/O request has been fulfilled but can continue their work.
- Support of multiple database instances.

I/O Threads per Database

EnablePreAllocateIOWorker
MinIOPoolWorkers, MaxIOPoolWorkers
IOPoolIdleTimeout, IOWorkerStackSize

Minimum and maximum number of I/O worker threads within a database and their maximum duration if they are not used.



Usually it is not necessary to adapt these parameters. The database can start additional I/O worker threads on request.

The parameter **EnablePreAllocateIOWorker** defines if I/O worker threads are already generated during startup phase. As a default it is set to NO meaning that threads are only started when needed. This is usually more effective. Be aware that if the configuration in near machine resource limits it may happen that I/O worker thread resources are not available during runtime. F.e. this might prevent the execution of a successful backup.

MinIOPoolWorkers defines the minimum number of I/O worker threads that were allocated during the startup phase. If the parameter is set to a value smaller than the number of priorities, then at least as many workers are started as priorities are defined.

With setting the parameter **MaxIOPoolWorkers** it is possible to restrict the number of I/O worker threads.

(The value for **MaxIOPoolWorkers** is identical to **MinIOPoolWorkers** if **EnablePreAllocateIOWorker** is set to YES.)

IOPoolIdleTimeout describes the maximum time in seconds an I/O pool worker is allowed to be idle before it is released and the thread resources are returned to the operating system.

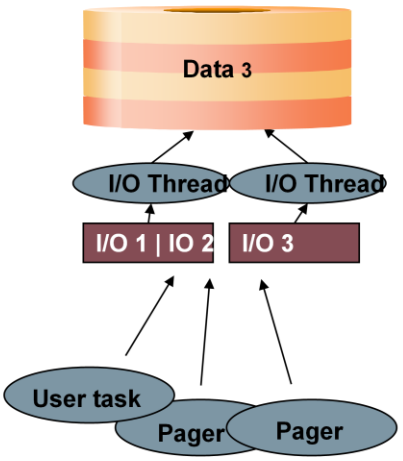
IOWorkerStackSize specifies the stack size for I/O worker threads in kilobytes which is as default the platform-specific minimum stack size.

(The parameters shown above were introduced with the implementation of a multiple database concept. This allows the use of several MaxDB databases within one instance. The parameters can be used to restrict I/O resources per database within one instance. This concept did not come into effect so far.)

Threshold value for the use of additional I/O queues

IOQueueFillingThreshold (_IOPROCS_SWITCH)

Defines the threshold value from which number of I/O requests it is changed to another I/O queue.



As soon as there are more than **IOQueueFillingThreshold** requests in the queue of an I/O thread the system tries to put each additional I/O request to another I/O queue.

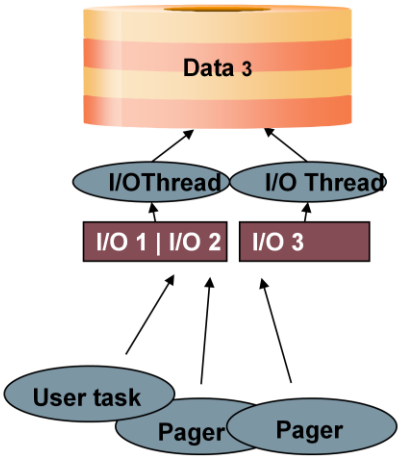
Values: Default: 1 (recommended)
Min: 0
Online change: YES

If there are not enough I/O worker threads available to handle all filled queues then the system automatically starts an additional thread.

I/O Queues per volume

VolumeIOQueuesForLowPriority
VolumeIOQueuesForMediumPriority
VolumeIOQueuesForHighPriority

Number of I/O queues for requests with low, medium and high priority per volume



By defining the number of queues per volume and per priority you can influence the priorities of I/O for certain requests.

VolumeIOQueuesForLowPriority:
Default: 1
Min: 1
Max: 10
Online Change: NO

VolumeIOQueuesForMediumPriority:
Default: 4
Min: 0
Max: 20
Online Änderung: NO

VolumeIOQueuesForHighPriority:
Default: 5
Min: 0
Max: 10
Online Änderung: NO

You can watch the states of current I/O requests in the system by the use of the console (x_cons) and the system view IOJOBS.

New I/O: Queuing Parameters

IOQueueFillingThreshold (1)

- How many I/O jobs must be in the queue before selecting another queue
- Maps to old parameter _IOPROCS_SWITCH

VolumeIOQueuesForHighPriority (0)

- Number of queues per volume reserved exclusively for high-priority jobs

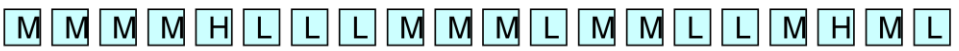
VolumeIOQueuesForMediumPriority (9 Unix, 3 Win)

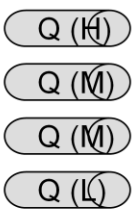
- Number of queues per volume for medium priority jobs (can be used by high prio jobs, but not by low prio jobs)

VolumeIOQueuesForLowPriority (1)

- Number of queues per volume for low priority jobs (can be used by all prio jobs)

Example: Enqueue following jobs for one volume (H=high, M=medium, L=low priority):





Note:

- Usually, there will be much more medium-priority queues
- Priorities are given by kernel code

Animated slide: Use of I/O queues with different priorities.

Jobs with higher priority can also be processed in queues which are designated for lower priority. You see in the example that the first job with medium priority will be handled by the IOQueueForLowPriority when it is currently idle.

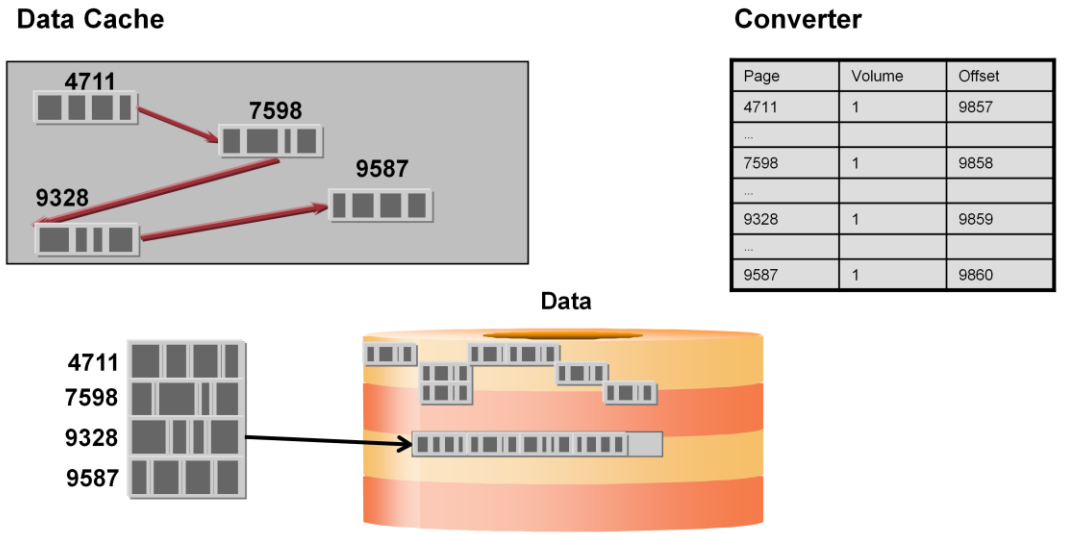
When all L and M Queues are filled the next job with medium priority will not be handled by the H Queue as this one is reserved for jobs with highest priority.

The priorities are provided by the kernel code. As am example jobs like CHECK DATA will run with low priority.

Use of Cluster Areas (I)

ClusterWriteThreshold (CLUSTER_WRITE_THRESHOLD)

Value in percent beginning with which cluster size the data of cluster tables is written to separate FBM sectors even if the sector is not completely filled.



MaxDB builds clusters for tables with the cluster flag to improve read performance for scans.

If blocks are written for cluster tables the pager tasks are looking for logically clustered blocks. Logically clustered blocks are those with successive cluster keys. The cluster key is defined by the primary key or another logical key which must not be unique on application side (f.e. time characteristic). Pager tasks write those blocks adhesively to the data area.

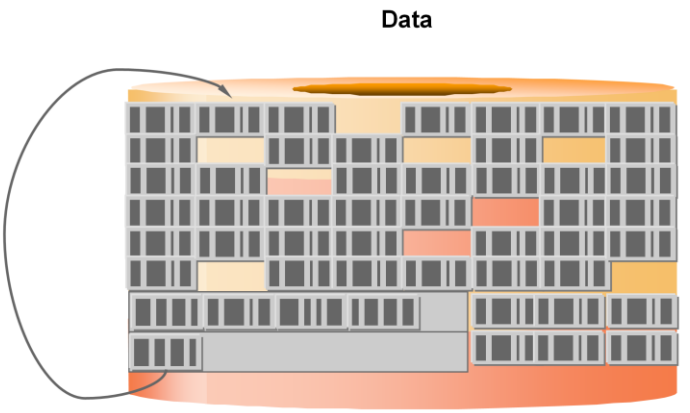
A cluster built by pager tasks is only written to a separate FBM section if the number of blocks within the cluster is at least **ClusterWriteThreshold** % of **DataIOClusterSize** and a free section in the data volumes is available. During backup and restore the clustering is not lost. If the percentage falls below **ClusterWriteThreshold** and no more free section is available the cluster is splitted and written to different free blocks.

Values: Default: 80%
Min: 0, Max: 100
Online Änderung: Ja

Use of Cluster Areas (II)

ClusterCompressionFillThreshold

If a cluster only holds ClusterCompressionFillThreshold % of a FBM section then this cluster is read, marked as changed within the cache and will be written to another position with the next savepoint. This generates space for larger clusters that can use a section in a better way.



If the database is filled to a high amount there is increased risk of writing too small clusters because there are no more free FBM sections for bigger clusters. So the scan performance of the system will be restricted.

FBM sections are released if they are only filled with a few blocks.

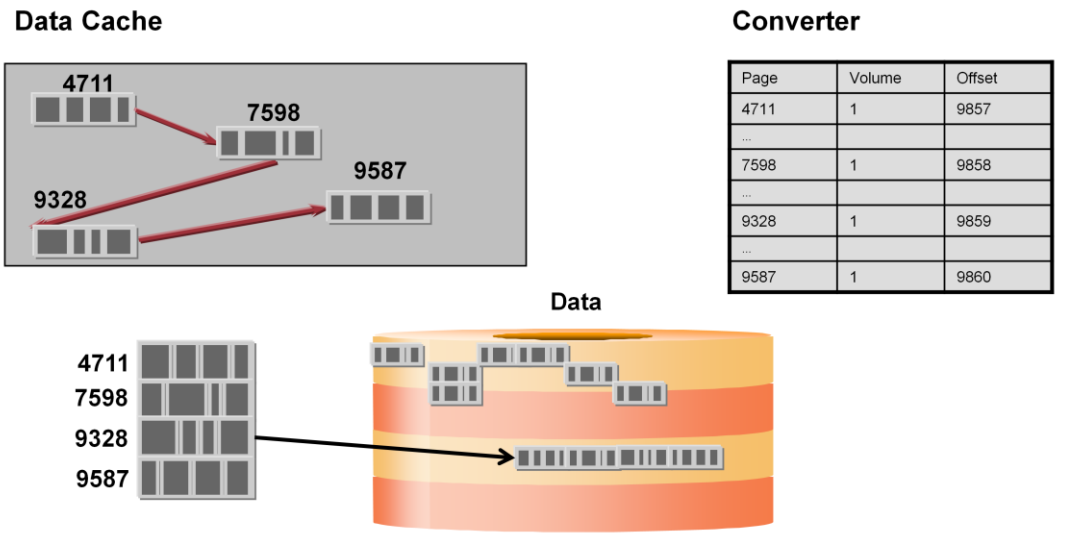
At the end of a savepoint it is checked by pager tasks if there are FBM sections with a low filling grade. Server tasks read the affected blocks to the data cache and mark it as modified. The blocks are written to other positions in the data area at the latest with the next savepoint. The FBM sections are now free for large table clusters.

Values: Default: 10%
Min: 0, Max: 50
Online Change: YES

Building Clusters for LOBs

UseLobClustering (CLUSTERED_LOBS)

Storage of blocks with LOB values in clusters.



Pager tasks also build clusters for LOB values if the parameter **UseLobClustering** is set to YES.

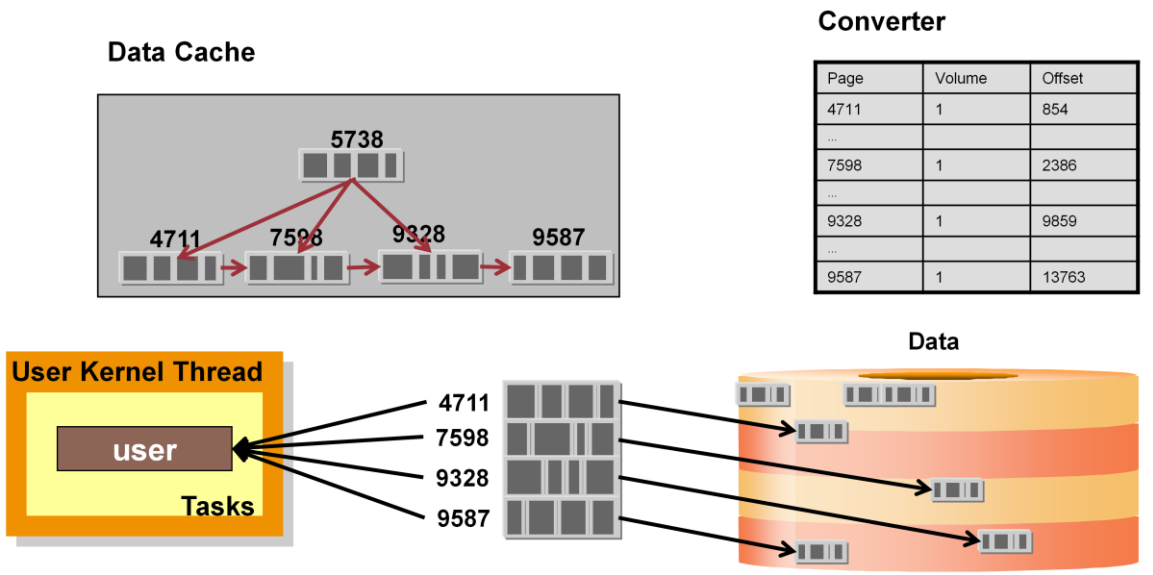
The storage of blocks for LOB data is also influenced by the parameter **ClusterWriteThreshold**.

Values: Default: YES
Online Change: YES

Asynchronous Read of table Pages

ReadAheadTableThreshold

User tasks utilize asynchronous read requests to scan tables or ranges of tables with parallel I/O



As of version 7.8 MaxDB uses parallel I/O requests to speed up table scans and table range scans. User tasks read index level 1 pages into the cache, determine the volume block positions of the pages stored in the separators by reading the converter and send asynchronous I/O requests to the I/O system. The user task doesn‘t wait for every single I/O before sending the next I/O request.

User tasks use asynchronous I/O requests if the size of the scan exceeds the number of pages specified as value of the parameter **ReadAheadTableThreshold.** The query optimizer evaluates the range size before the statement execution starts.

The database uses asynchronous I/O for scans only, if the number of current running I/O requests is below the value of **MaxDataAreaReadAheadRequests**. The determination of the current running I/O requests happens during the operation on the index level 1 page. This operation prevents the system from I/O overload situations. I/O orders for short SQL commands should not be blocked by asynchronous parallel I/O.

Asynchronous I/O read requests have the priority low.

Values:

ReadAheadTableThreshold: Default: 100 Pages

Min: 0
Max: 2147483647

Online Change: Yes

MaxDataAreaReadAheadRequests:

Default: 2 x Number of Data Volumes
Min: 0

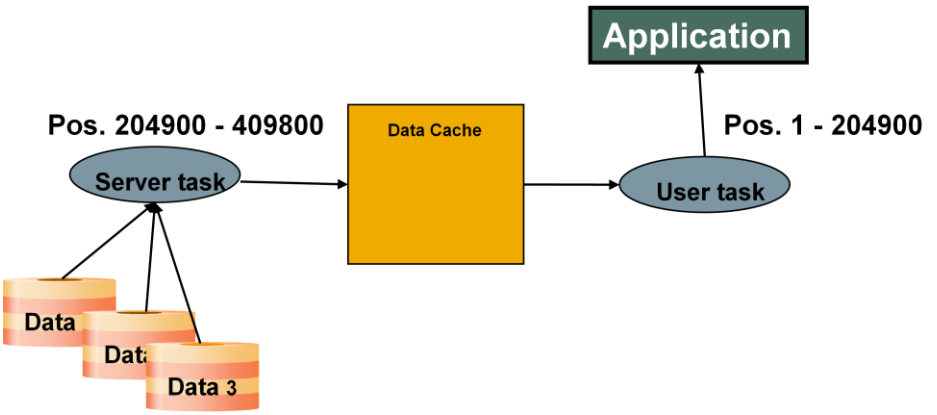
Max: 2147483647
Online Change: Yes

Monitoring: select * from monitor_pages where description like 'Read ahead%'

Asynchronous Read of LOB Pages

ReadAheadLobThreshold (_READAHEAD_BLOBS)

A server task reads data pages from the data volumes during read of BLOB pages. In the meantime the user tasks may deliver previously read pages to the application.



When reading larger LOB values, it is a good idea to continue reading asynchronously from the volumes while sending a read package to the application. This is done by a server task if the length of the LOB value to be read exceeds the value of the parameter **ReadAheadLobThreshold**.

Values:

Default: 25

Min: 2 * CommandBufferSize / 8196 Max: 262144

Online change: YES

Debug Parameters

EnableIOTimeStatistic (YES)

- Collect statistics about I/O times; monitoring I/O throughput
- Enabled by default, as this doesn't cause much performance impact
- In <=7.6, I/O times would be measured only when measuring all times (x_cons time enable)

TraceIOManager

- Defines which level of tracing for I/O manager is activated
- As default is it switched off (0)
- trace levels from 1 (only tracing of high priority information) to 9 (tracing of all information)

RetiredIOJobCount

- Specifies the number of retired I/O job descriptions per context to keep for analysis purposes

More Parameters (I)

UseVectorIOSimulation (on Linux IF_OPEN_DIRECT_OR_RAW_DEVICE)

(in general NEVER, on Linux IF_OPEN_DIRECT_OR_RAW_DEVICE)

- Simulate Unix writev()/readv() to work around bugs

UseIOCompletionPort

- Specifies if I/O via Windows I/O completion port is used (YES)
- As default the Windows specific mechanism is used instead of own I/O worker pool

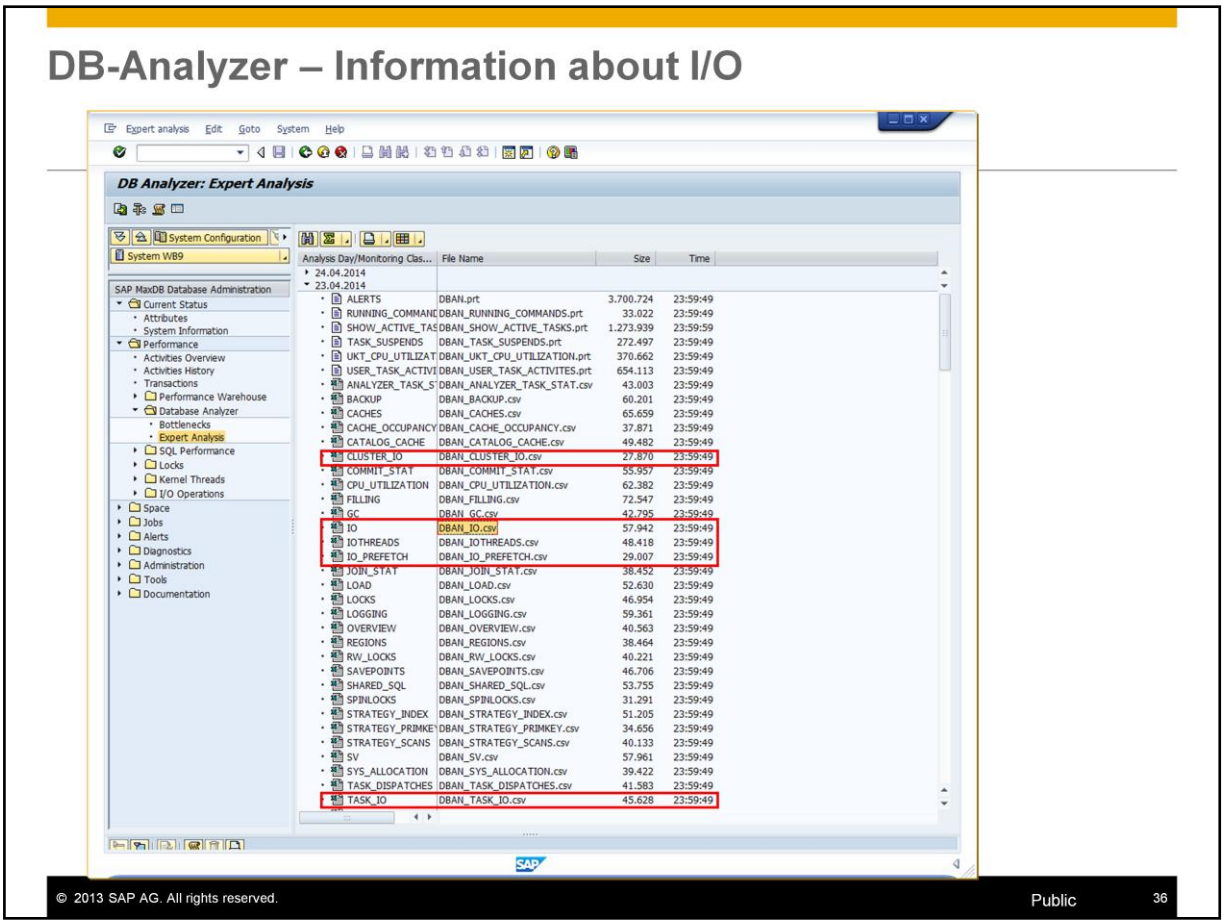
More Parameters (II)

FormatDataVolume (YES)

- Actually reserve disk space by writing empty blocks
- Raw volumes are not formatted
- NO makes sense only for demo/test installations to speed up DB creation

VolumeFormattingMode (PARALLEL)

- recommended that each volume is located on its own device
- PARALLEL (default) formats all volumes in parallel in several threads
- SERIAL formats all volumes one after another



DBAN_CLUSTER_IO.csv (as of 7.9):

Information about clustered read and cluster compression operations

DBAN_IO.csv:

Read and write operations to cache pages and data pages

DBAN_IO_PREFETCH.csv:

Statistics about prefetching (requests, ignored requests, LOB requests)

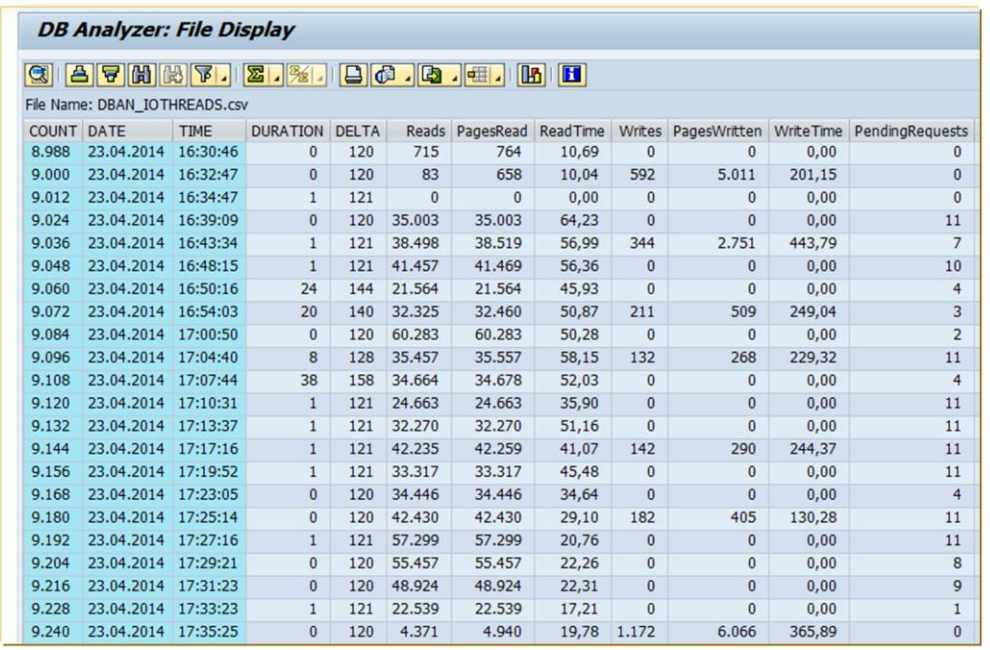
DBAN_IOTHREADS.csv:

Number and duration of physical write and read operations (I/O threads)

DBAN_TASK_IO.csv:

Number and duration of physical writes and reads from perspective of the log writer, the user task and the pager.

DBAN_IOTHREADS



DB Analyzer: File Display

File Name: DBAN_IOTHREADS.csv

COUNT	DATE	TIME	DURATION	DELTA	Reads	PagesRead	ReadTime	Writes	PagesWritten	WriteTime	PendingRequests
8.988	23.04.2014	16:30:46	0	120	715	764	10,69	0	0	0,00	0
9.000	23.04.2014	16:32:47	0	120	83	658	10,04	592	5.011	201,15	0
9.012	23.04.2014	16:34:47	1	121	0	0	0,00	0	0	0,00	0
9.024	23.04.2014	16:39:09	0	120	35.003	35.003	64,23	0	0	0,00	11
9.036	23.04.2014	16:43:34	1	121	38.498	38.519	56,99	344	2.751	443,79	7
9.048	23.04.2014	16:48:15	1	121	41.457	41.469	56,36	0	0	0,00	10
9.060	23.04.2014	16:50:16	24	144	21.564	21.564	45,93	0	0	0,00	4
9.072	23.04.2014	16:54:03	20	140	32.325	32.460	50,87	211	509	249,04	3
9.084	23.04.2014	17:00:50	0	120	60.283	60.283	50,28	0	0	0,00	2
9.096	23.04.2014	17:04:40	8	128	35.457	35.557	58,15	132	268	229,32	11
9.108	23.04.2014	17:07:44	38	158	34.664	34.678	52,03	0	0	0,00	4
9.120	23.04.2014	17:10:31	1	121	24.663	24.663	35,90	0	0	0,00	11
9.132	23.04.2014	17:13:37	1	121	32.270	32.270	51,16	0	0	0,00	11
9.144	23.04.2014	17:17:16	1	121	42.235	42.259	41,07	142	290	244,37	11
9.156	23.04.2014	17:19:52	1	121	33.317	33.317	45,48	0	0	0,00	11
9.168	23.04.2014	17:23:05	0	120	34.446	34.446	34,64	0	0	0,00	4
9.180	23.04.2014	17:25:14	0	120	42.430	42.430	29,10	182	405	130,28	11
9.192	23.04.2014	17:27:16	1	121	57.299	57.299	20,76	0	0	0,00	11
9.204	23.04.2014	17:29:21	0	120	55.457	55.457	22,26	0	0	0,00	8
9.216	23.04.2014	17:31:23	0	120	48.924	48.924	22,31	0	0	0,00	9
9.228	23.04.2014	17:33:23	1	121	22.539	22.539	17,21	0	0	0,00	1
9.240	23.04.2014	17:35:25	0	120	4.371	4.940	19,78	1.172	6.066	365,89	0

By default User Tasks do not execute the I/O itself, the I/O request is put into a queue and processed by the I/O thread. To analyze the I/O performance DBAN_IOTHREADS.csv can be used.

In this example we see that we have

- high number of read IO (*PagesRead*)
- very bad I/O times for reading (*ReadTime* in ms)
- Write I/O especially every 10 minutes (*PagesWritten*) – could be savepoint
- Very bad I/O times for writing (*WriteTime* in ms)
- I/O threads got a bottleneck (*PendingRequests* > 0), the I/O threads could not write/read the data fast enough to avoid any wait situation in the I/O threads.

- In this example we should check the database disk configuration and the disk performance on hardware level.

DBAN_TASK_IO

DB Analyzer: File Display

File Name: DBAN_TASK_IO.csv

COUNT	DATE	TIME	AvgRTime_UserPTask	User_PRThread	User_PRThreadPg	AvgAbsRTime_UserPThread	AvgRelRTime_UserPThread	User_PWTask
8.988	23.04.2014	16:30:46	0	708	748	10,84	10,79	0
9.000	23.04.2014	16:32:47	0	68	68	9,43	9,38	0
9.012	23.04.2014	16:34:47	0	0	0	0,00	0,00	0
9.024	23.04.2014	16:39:09	0	1-	1-	1,00-	1,00-	0
9.036	23.04.2014	16:43:34	0	32	51	175,93	175,89	0
9.048	23.04.2014	16:48:15	0	163	172	1,00-	1,00-	0
9.060	23.04.2014	16:50:16	0	1	1	413,81	413,76	0
9.072	23.04.2014	16:54:03	0	2	2	788,00	787,94	0
9.084	23.04.2014	17:00:50	0	0	0	0,00	0,00	0
9.096	23.04.2014	17:04:40	0	1	1	338,43	338,40	0
9.108	23.04.2014	17:07:44	0	26	40	98,88	98,84	0
9.120	23.04.2014	17:10:31	0	2	2	519,56	519,54	0
9.132	23.04.2014	17:13:37	0	2	2	8.500,81	8.500,77	0
9.144	23.04.2014	17:17:16	0	6	6	101,79	101,75	0
9.156	23.04.2014	17:19:52	0	8	8	37,87	37,82	0
9.168	23.04.2014	17:23:05	0	5	5	374,58	374,51	0
9.180	23.04.2014	17:25:14	0	33	33	75,46	75,40	0
9.192	23.04.2014	17:27:16	0	92	92	72,28	72,23	0
9.204	23.04.2014	17:29:21	0	168	168	81,44	81,39	0
9.216	23.04.2014	17:31:23	0	89	89	71,76	71,70	0
9.228	23.04.2014	17:33:23	0	0	0	0,00	0,00	0
9.240	23.04.2014	17:35:25	0	0	0	0,00	0,00	0
9.252	23.04.2014	17:37:25	0	0	0	0,00	0,00	0
9.264	23.04.2014	17:39:26	0	1	1	277,34	277,26	0

User_PRThread – Physical user reads via I/O threads
User_PRThreadPG – Number of pages read via I/O threads

AvgAbsRTime_UserPThread – Average absolute time (ms) for user reads via I/O threads
AvgRelRTime_UserPThread – Average relative time (ms) for user reads via I/O threads

New I/O: Console Commands

Original console commands still working

x_cons show io

- No changes

x_cons show aio

- Shows statistics of running and past backups/formatting

New commands:

x_cons show iopending

- Shows running I/O jobs in whole system

x_cons show cport

- Shows registered completion ports and pending I/O on them

Modified commands:

x_cons show rte

- Shows different list for I/O threads – workers of I/O worker pool
- Shows I/O on volume I/O queues instead of I/O worker queues

x_cons show t_cnt

- Additionally shows pending I/Os on task (including I/Os in progress on completion port if waiting on completion port)

On the following slides some examples of x_cons output are shown. They are simplified as the complete output would be difficult to survey.

This presentation will not provide a detailed description of x_cons output but can only be regarded as introduction and gives an impression which information can be found here.

Console Commands: Examples (I)

x_cons show active

```
SERVERDB: LC5

ID   UKT  Win   TASK       APPL Current         Timeout Region     Wait
          tid   type        pid state          priority cnt try    item
T2     2 0x1430 Logwr           Vvectorio             0 0              45102(s)
T132   8 0x1590 User       5537* Running              0 72             1486353(r)
T235  10 0x1508 User       2079* LogIOwait(234)       0 0              43283(s)

Console command finished (2007-06-04 11:15:34).
```

x_cons show io

```
SERVERDB: LC5

Volume                          Devs.    Read(s)       Read   Write(s)      Write
Name                             No.      Count      Pages      Count      Pages
knltrace                           0          0          0          1          1
d:\sapdb\lc5\devspaces\DATA_00     1      57916      57916       1002       3802
d:\sapdb\lc5\log\LOG_001           2        401       1448      34288      58506
                     total I/O:           58317      59364      35291      62309

Console command finished (2007-06-04 11:19:09).
```

x_cons show t_cnt (only log writer)

```
--------- T2                  LOGWRITER        ( pid = ?          ) ---------
 dispatcher_cnt: 902359                              command_cnt    : 0
 exclusive_cnt : 107165                              self_susp_cnt : 0
 Resume count 0  total 25130      History [ T176 T176 T176 ]
 self_write_io : 1182                                self_write_pg : 1182
 dev_write_io  : 40423                               dev_write_pg  : 70520
 state_vwait   : 0          state_vsleep  : 0        state_vsusp   : 25130
-------------------------------------------------------------------------------
```


x_cons show active

- Delivers a list of currently active tasks
- If the task is performing an I/O operation an appropriate status is displayed

Possible states (concerning I/O):

AsynClose (Stream Close):
Task closes I/O ports after saving or restoring

AsynCntl:
Task determines parameter or initializes a backup device

AsynIO(Stream IO):
Task executes asynchronous I/O operation (when saving or restoring)

AsynOpen(Stream Open):
Task opens I/O ports for saving or restoring

AsynWaitRead/Write(Stream Wait(R)/(W)):
Task waits for the end of an I/O operation during a save or restore operation

IO wait (W/R):
Task waits for the result of an I/O operation (W:write, R:read)

Vdetach(Detach Volume):
Task closes I/O ports (volumes, normal operation)

Vdualvectorio(Dual Vector IO):
Task executes a vector I/O operation (write or read) on two volumes in parallel

Vvectorio (Vector IO):
Task executes a vector I/O operation (write or read)

x_cons show io

- Counters for I/O on volumes

x_cons show t_cnt

- Counters for I/O on this task

Console Commands: Examples (II)

x_cons show rte

```
SERVERDB: LC5

Kernel Threads:
 Thread              Win  State
 Name                Tid
 . . .
 DEV0              0xF54  Sleeping
 ASYNC0           0x1378  Sleeping
 . . .

IO Worker Threads:
 Thread                Win  State        IO
 Name                  Tid              Counter
 IO-WORKER0         0x10A4  Sleeping     0
 IO-WORKER1         0x13F8  Sleeping     109677
 IO-WORKER2         0x10B8  Sleeping     0

 1 IO Worker threads executing concurrently

User Kernel Threads:
 . . .
Processor information:
 . . .
I/O via UKTs and I/O Threads:
 Thread          Win  Volume                Devs.      Read     Write    Queue
 Name            Tid  Name                   No.     Count     Count  Len. Max.
 UKT2         0x1430  d:\sapdb....\LOG_001     2         0     56569   --  (--)
 UKT4         0x11A8  d:\sapdb....ATA_0001     1     52126       255   --  (--)
 UKT4         0x11A8  d:\sapdb....\LOG_001     2       357         9   --  (--)
 UKT5         0x115C  d:\sapdb....ATA_0001     1       637         0   --  (--)
 UKT6         0x16B0  knltrace                 0         0         1   --  (--)
 UKT6         0x16B0  d:\sapdb....ATA_0001     1        62      1229   --  (--)
 UKT7          0x5EC  d:\sapdb....ATA_0001     1       980         0   --  (--)
 UKT7          0x5EC  d:\sapdb....\LOG_001     2        45         2   --  (--)
 UKT8         0x1590  d:\sapdb....ATA_0001     1      4111         0   --  (--)
 UKT9         0x13D8  d:\sapdb....ATA_0001     1         2         0   --  (--)
 I/O0         0x17A4  knltrace                 0         0         0    0  ( 0)
 I/O0         0x1234  d:\sapdb....ATA_0001     1         0         0    0  ( 0)
 I/O0         0x15FC  d:\sapdb....\LOG_001     2         0         0    0  ( 0)

Console command finished (2007-06-04 11:27:50).
```


x_cons show rte

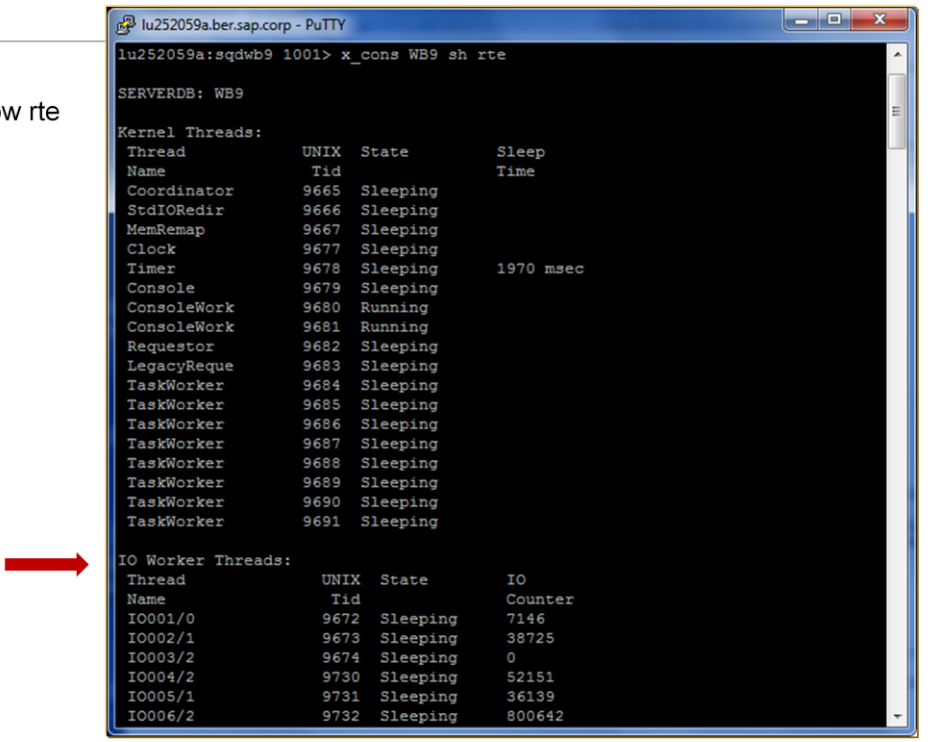
- Old appearance
- Displays the status of RTE threads, including I/O threads
- Counters of I/O on UKT and I/O threads

x_cons show aio (no slide)

- Counters for I/O on stream media (backup)
- Shows statistics of running and past backups/formatting

New I/O: Console Commands: Examples (I)

x_cons show rte



x_cons show rte

- Same command but with a difference appearance
- Displays the status of RTE threads, including I/O threads
- Counters of I/O on UKT and I/O threads

New I/O: Console Commands: Examples (II)

x_cons show rte
(continuation)

lu252059a.ber.sap.corp - PuTTY

```
 53 IO Worker threads executing concurrently

User Kernel Threads:
 Thread        UNIX  State       Dispatch  TaskSwitch Active Total Task
 Name           Tid              Counter     Counter  Tasks Tasks Cluster
 UKT1          9692  Sleeping          4           0      1     1 TW
 UKT2          9693  Sleeping          4           0      1     1 LW
 UKT3          9694  Sleeping          2           0      0     1 UT
 UKT4          9695  Sleeping   21874354    14071223     53    53 53*SV
 UKT5          9696  Sleeping    1687234           1      2     9 IDL,8*FS
 UKT6          9697  Sleeping      25528       25402     10    10 10*GC
 UKT7          9698  Sleeping     436248           0      1     1 TI
 UKT8          9699  Sleeping    5804932     2715976      9    35 PLW,33*US,ID
 UKT9          9700  Sleeping    2861695     2112697      9    35 PLW,33*US,ID
 UKT10         9701  Sleeping    6642410     3392782      9    36 PLW,34*US,ID
 UKT11         9702  Sleeping          2           0      1     1 [IDL]

Processor information:
 Processors     : 8
 Processor cores: 2

I/O via UKTs and I/O Threads:
 Thread        UNIX  Volume               Devs.      Read     Write    Queue
 Name           Tid  Name                  No.      Count     Count  Len. Max.
 I/O0             0  knltrace                1          0         1    0  ( 1)
 I/O0             0  /sapdb/W....ISKD0001    2       1578       939    0  (19)
 I/O1             0  /sapdb/W....ISKD0001    2        741     20455    0  (20)
 I/O2             0  /sapdb/W....ISKD0001    2        268      2217    0  (20)
 I/O3             0  /sapdb/W....ISKD0001    2        643      1601    0  (19)
 I/O4             0  /sapdb/W....ISKD0001    2       1386      1398    0  (19)
 I/O5             0  /sapdb/W....ISKD0001    2     966118        11    0  ( 2)
 I/O6             0  /sapdb/W....ISKD0001    2     295712         1    0  ( 2)
 I/O7             0  /sapdb/W....ISKD0001    2      91485         1    0  ( 1)
 I/O8             0  /sapdb/W....ISKD0001    2      24015         0    0  ( 1)
 I/O9             0  /sapdb/W....ISKD0001    2       5576         0    0  ( 1)
```


The status of each concurrently active I/O worker thread and the number of I/O operations are displayed with the name and priority. Each I/O worker can serve any queue of the same or higher priority jobs. Workers are picking the next queue to process from ticket queue.

New I/O: Console Commands: Example (III)

x_cons show iopending

```
SERVERDB: TEST1_CL
I/O jobs processed by job workers

 O Flg Device                          BlockNr.   Cnt State
 r AC  ...data\TEST1_CL\data\DISKD0001        1957   1 Pending
 r AC  ...data\TEST1_CL\data\DISKD0001        5306   1 Success(1)
 r AC  ...data\TEST1_CL\data\DISKD0001        3465   1 Pending
Console command finished (2007-06-04 14:34:13).
```

x_cons show cport

```
SERVERDB: TEST1_CL
I/O jobs on completion ports

I/O jobs on completion port 'MultiIOTest 3':
 O Flg Device                          BlockNr.   Cnt State
 r AC  ...data\TEST1_CL\data\DISKD0001        4314   1 Pending
 r AC  ...data\TEST1_CL\data\DISKD0001        2525   1 Pending
 r AC  ...data\TEST1_CL\data\DISKD0001        5138   1 Pending
 r AC  ...data\TEST1_CL\data\DISKD0001        3466   1 Pending
 r AC  ...data\TEST1_CL\data\DISKD0001        2255   1 Pending
 r AC  ...data\TEST1_CL\data\DISKD0001        5105   1 Pending
 r AC  ...data\TEST1_CL\data\DISKD0001        1138   1 Success(1)
 r AC  ...data\TEST1_CL\data\DISKD0001        4783   1 Pending
 r AC  ...data\TEST1_CL\data\DISKD0001        4566   1 Pending
 r AC  ...data\TEST1_CL\data\DISKD0001         650   1 Pending
```


x_cons show iopending

Shows running I/O jobs in the system.

x_cons show cport

Shows registered completion ports and pending I/O on them. The display is similar to „show iopending“ but only shows I/O on completion ports.

The first column „O“ shows the operation (f.e. ‚F‘ format, ‚w‘ write, ‚r‘ read).

Flg indicates ‚A‘ asynchronous I/O, ‚C‘ I/O on completion port, priorities of I/O.

State shows the I/O job status (f.e. pending, in progress, success, failed)

Input/output completion port (**IOCP**) is an API for performing multiple simultaneous asynchronous input/output operations. An input/output completion port object is created and associated with a number of sockets or file handles.. The I/O completion port manages multiple threads and their concurrency.

“I/O completion ports provide an efficient threading model for processing multiple asynchronous I/O requests on a multiprocessor system. When a process creates an I/O completion port, the system creates an associated queue object for requests whose sole purpose is to service these requests. Processes that handle many concurrent asynchronous I/O requests can do so more quickly and efficiently by using I/O completion ports in conjunction with a pre-allocated thread pool than by creating threads at the time they receive an I/O request.” (info from Microsoft Developer Network)

New I/O: Console Commands: Example (IV)

x_cons show t_cnt – synchronous I/O

```
--------- T89                        MultiIOTest 0 ( pid = ?          ) ---------
 dispatcher_cnt: 6179                                  command_cnt    : 0
 exclusive_cnt : 144                                   self_susp_cnt : 0
 Resume count 0  total 11          History [ T88 T88 T88 ]
 dev_read_io    : 6209        rel_dev_rd_tm : 5.1639      abs_dev_rd_tm : 5.1640
 dev_read_pg    : 6209                                   pages_per_io  : 1.0
 state_vwait    : 0           state_vsleep  : 0          state_vsusp    : 11
 same_ukt_coll : 0
 O Flg Device                                  BlockNr.   Cnt State
 r     ...data\TEST1_CL\data\DISKD0001          5995     1 Pending
-------------------------------------------------------------------------------
```

x_cons show t_cnt – I/O on completion port

```
--------- T89                        MultiIOTest 0 ( pid = ?          ) ---------
 dispatcher_cnt: 1031                                  command_cnt    : 0
 exclusive_cnt : 114                                   self_susp_cnt : 0
 Resume count 0  total 6           History [ T88 T88 T88 ]
 I/O on cport: MultiIOTest 0
 dev_read_io    : 1029        rel_dev_rd_tm : 5.1473      abs_dev_rd_tm : 5.1474
 dev_read_pg    : 1029                                   pages_per_io  : 1.0
 state_vwait    : 0           state_vsleep  : 0          state_vsusp    : 6
 same_ukt_coll : 0
 O Flg Device                                  BlockNr.   Cnt State
 r AC  ...data\TEST1_CL\data\DISKD0001          5996     1 Pending
 r AC  ...data\TEST1_CL\data\DISKD0001          1546     1 Pending
 r AC  ...data\TEST1_CL\data\DISKD0001          1957     1 Pending
 r AC  ...data\TEST1_CL\data\DISKD0001          5306     1 Success(1)
 r AC  ...data\TEST1_CL\data\DISKD0001          3465     1 Pending
 r AC  ...data\TEST1_CL\data\DISKD0001          5772     1 Success(1)
 r AC  ...data\TEST1_CL\data\DISKD0001           904     1 Pending
 r AC  ...data\TEST1_CL\data\DISKD0001           485     1 Pending
```


Displays detailed measurement values for individual database tasks.

If time measurement is switched on (x_cons time enable) you can get more information about the relative/absolute duration of I/O operations.

dev_read_io: number of I/Os via I/O threads (dev)

dev_read_io: number of pages written via I/O (threads (dev)

New I/O: System Views

Legacy I/O system views still supported:

IOTHREADSTATISTICS / IOTHREADSTATISTICSRESET

- I/O statistics per volume queue (not directly related to I/O threads)

BACKUPTHREADS

- Statistics for last backup
- Statistics held after backup media closed, until next backup of same type

New system view:

IOJOBS

- List of all pending I/O jobs in progress
- Displays all information about a job (e.g., start block, block count, operation, job type, ...)
- Inherently volatile

Questions

SAP® MaxDB™ I/O Concept

SAP® MaxDB™ – Expert Sessions Learning Map (1)

SAP® MaxDB ™ Features	SAP® MaxDB ™ Administration	SAP® MaxDB ™ Problem Analysis
Session 1: Low TCO with the SAP MaxDB Database	Session 2: Basic Administration with Database Studio	Session 5: SAP MaxDB Data Integrity
Session 6: New Features in SAP MaxDB Version 7.7	Session 3: CCMS Integration into the SAP System	Session 14: SAP MaxDB Tracing
Session 8: New Features in SAP MaxDB Version 7.8	Session 11: SAP MaxDB Backup and Recovery	Session 12: Analysis of SQL Locking Situations
	Session 13: Third-Party Backup Tools	
	Session 19: SAP MaxDB Kernel Parameter Handling	

SAP® MaxDB ™ Installation/Upgrade
Session 7: SAP MaxDB Software Update Basics

All Expert Sessions (recording and slides) are available for download
http://maxdb.sap.com/training/

SAP® MaxDB™ – Expert Sessions Learning Map (2)

SAP® MaxDB™ Architecture	SAP® MaxDB ™ Performance	SAP® MaxDB ™ & Content Server
Session 18: Introduction MaxDB Database Architecture	Session 4: Performance Optimization with SAP MaxDB	Session 23: SAP MaxDB & Content Server Architecture
Session 15: SAP MaxDB No-Reorganization Principle	Session 9: SAP MaxDB Optimized for SAP BW	Session 24: SAP MaxDB & Content Server Housekeeping
Session 17: SAP MaxDB Shadow Page Algorithm	Session 16: SAP MaxDB SQL Query Optimization (Part 1)	Session 25: SAP MaxDB & Content Server ODBC Tracing
Session 12: Analysis of SQL Locking Situations	Session 16: SAP MaxDB SQL Query Optimization (Part 2)	
Session 10: SAP MaxDB Logging	Session 22: SAP MaxDB Database Analyzer	
Session 20: SAP MaxDB Remote SQL Server		
Session 21: SAP MaxDB DBM Server		
Session 26: SAP MaxDB I/O Concept		

All Expert Sessions (recording and slides) are available for download
http://maxdb.sap.com/training/

Thank You!
Bye, Bye – And Remember Next Session

Feedback and further information:
http://www.scn.sap.com/irj/sdn/maxdb

Next Session: November, 2014
SAP MaxDB – Multitasking

Thank you

Contact information:

Heike Gursch
IMS MaxDB / liveCache
Heike.Gursch@sap.com

Christiane Hienger
IMS MaxDB / liveCache
Christiane.Hienger@sap.com