SAP® MaxDB™ Expert Session

SAP® MaxDB™: Multi Tasking
Christiane Hienger September 2, 2014

Public

SAP® MaxDB™ Expert Session

SAP® MaxDB™ Multi Tasking

Christiane Hienger
Heike Gursch
IMS MaxDB/liveCache Development Support

September 2, 2014

Agenda

- CPU Usage
- Load Balancing

Process Structure, CPU Usage

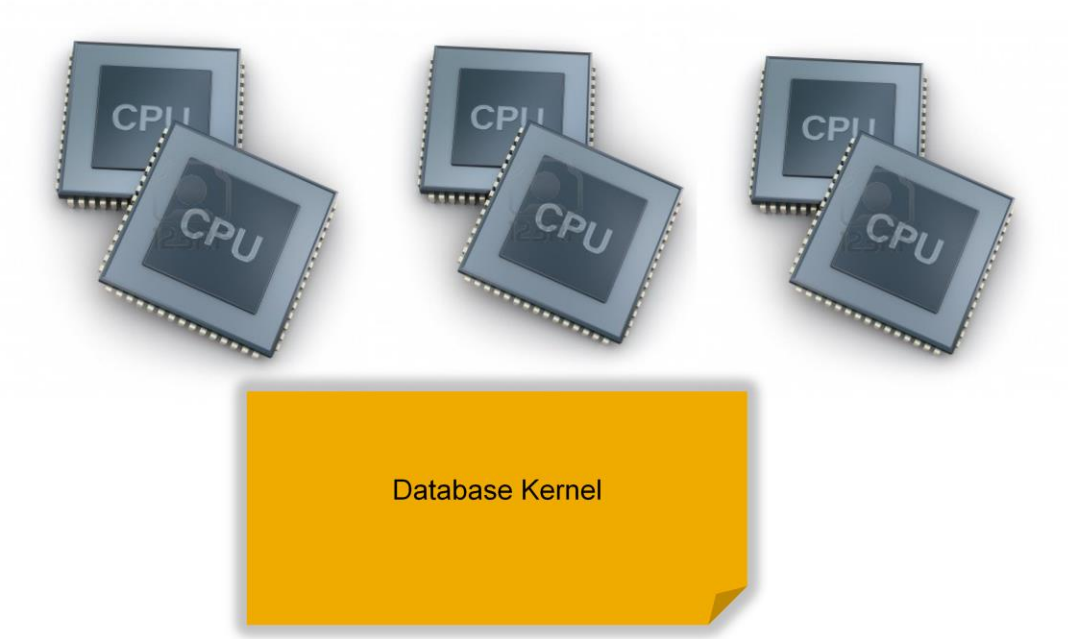
LoadBalancingCheckInterval	Load balancing between UKTs
LoadBalancingWorkloadThreshold	Threshold value for load balancing
LoadBalancingWorkloadDeviation	Precision of measurement for load balancing
EnableMultipleServerTaskUKT	Distribution of server tasks to UKTs
ExclusiveLockRescheduleThreshold	Stop running tasks
MaxExclusiveLockCollisionLoops	Busy Waiting (BW) during collisions
TaskSchedulerRetryLoops	Dispatcher loops
Task…Priority	Base priority per task state
TaskNiceElevationValue	Additional priorities for lockholders
ForceSchedulePrioritizedTask	Interruption of tasks

Agenda

- CPU Usage

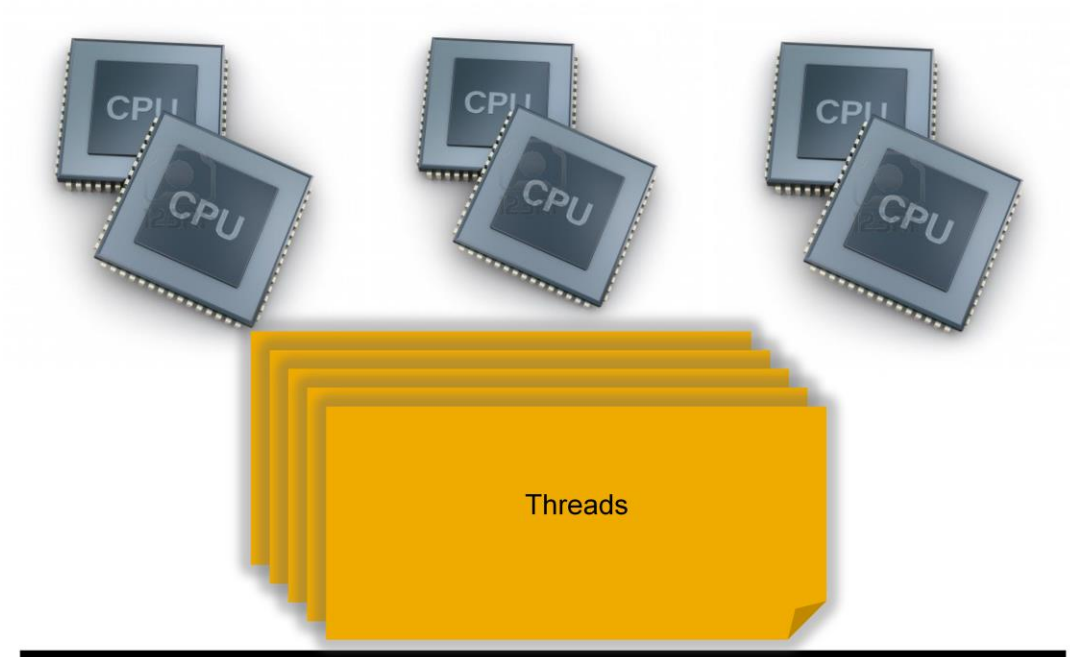
TaskCluster01 ... 03	Distribution of tasks to threads
MaxCPUs	Maximum number of intensely used processors
UsableCPUs	Maximum number of currently intensely used processors

Process Structure (1)



The database kernel runs as one process divided into threads.

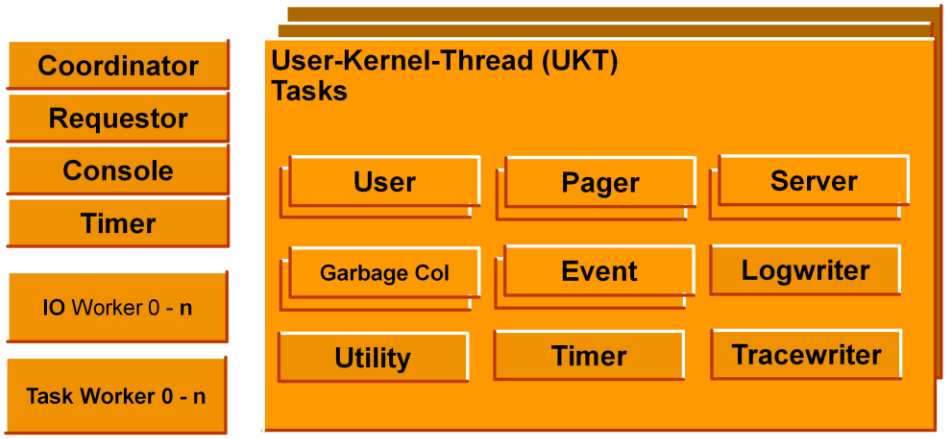
Process Structure (2)



Threads can be active in parallel on several processors within the operating system.

Threads perform various tasks.

Process Structure (3)

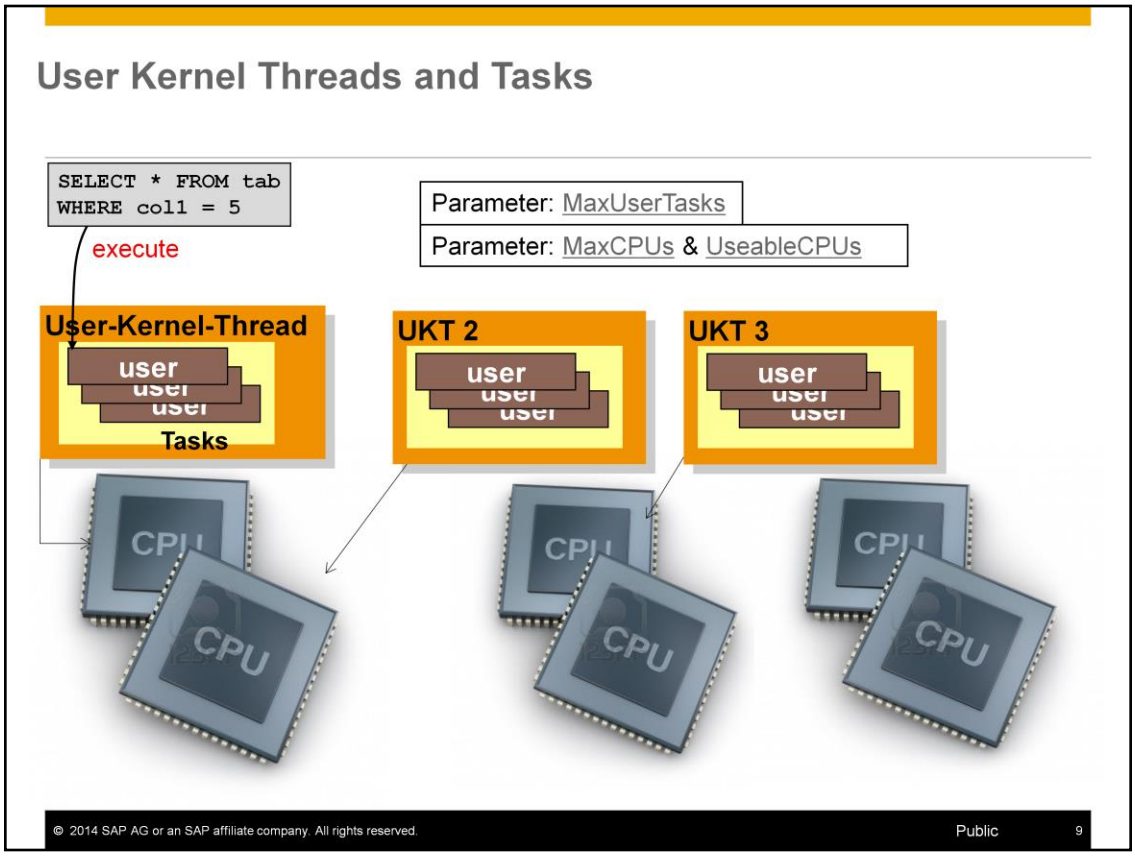


The runtime environment (RTE) defines the structure of the process and the threads.
When the runtime environment is started, that is, when the database instance is started in the Admin state, first the **coordinator thread** is generated. This thread is of particular importance.

- When started, the coordinator thread uses database parameters to determine the configuration of the memory and process of the database instance. For this reason, changes to database parameters take effect only after you have restarted the database instance.
- The coordinator thread also coordinates the start procedures of the other threads and monitors them while the database instance is in operation.
- If operating errors occur, the coordinator thread can stop other threads.

The **requestor thread** receives logon requests from the user processes to the database. The logon is assigned to a task within the user kernel thread.
The **console thread** collects information about the internal states of the database kernel when x_cons is being used.
The **clock thread** and the **timer thread** calculate internal times, for example to determine how much time was required to execute an SQL statement.

A user kernel thread forms a subset of all tasks (internal tasking).
User kernel threads (UKT) consist of several tasks that perform various tasks.
This tasking enables more efficient coordination of tasks than operating system tasking that employs individual threads.



Each user session (application process) is assigned exactly one **user task** at logon.
The maximum number of available user tasks is determined by the database parameter **MaxUserTasks**. This parameter also restricts the number of user sessions that can be logged on to the database system simultaneously.
The database parameter **MaxTaskStackSize** determines the memory usage of the user tasks.
The general database parameter **MaxCPUs** specifies the number of user kernel threads among which the user tasks are distributed. Other tasks and global threads use very little CPU time. The parameter **MaxCPUs** allows you to specify how many processors the database should use in parallel.

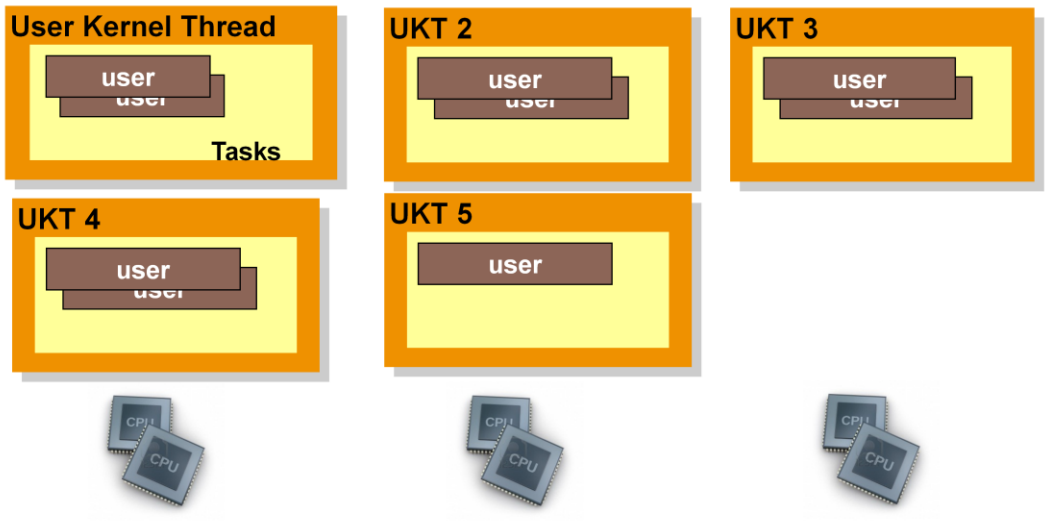
The parameter **UseableCPUs** allows an online adjustment of the number of used user kernel thread. This makes dynamic configuration changes according the available CPUs in the system possible.
As of version 7.4.03, user tasks can switch from one UKT to another if the previously-responsible UTK is overburdened. This results in better scaling for multiprocessor servers (SMP). To use this function, set the parameter **LoadBalancingCheckInterval** to a value greater than 0.
As of version 7.8 Load Balancing is released for MaxDB and liveCache instances and used by default. The scheduler immediately moves the task to

an idle user kernel thread if the current thread is overloaded.

Maximum Number of Intensively Used CPUs

MaxCPUs (MAXCPU)

Maximum number of CPUs used by the database instance for concurrent processing of UKTs containing user tasks.



This parameter serves to inform the database kernel that multiple CPUs can be used as maximum.

At the same time, it allows the database system to restrict CPU usage. Such a restriction only applies to UKTs that contain user tasks. Other UKTs continue to access any number of CPUs even if the value for MAXCPU is reduced.

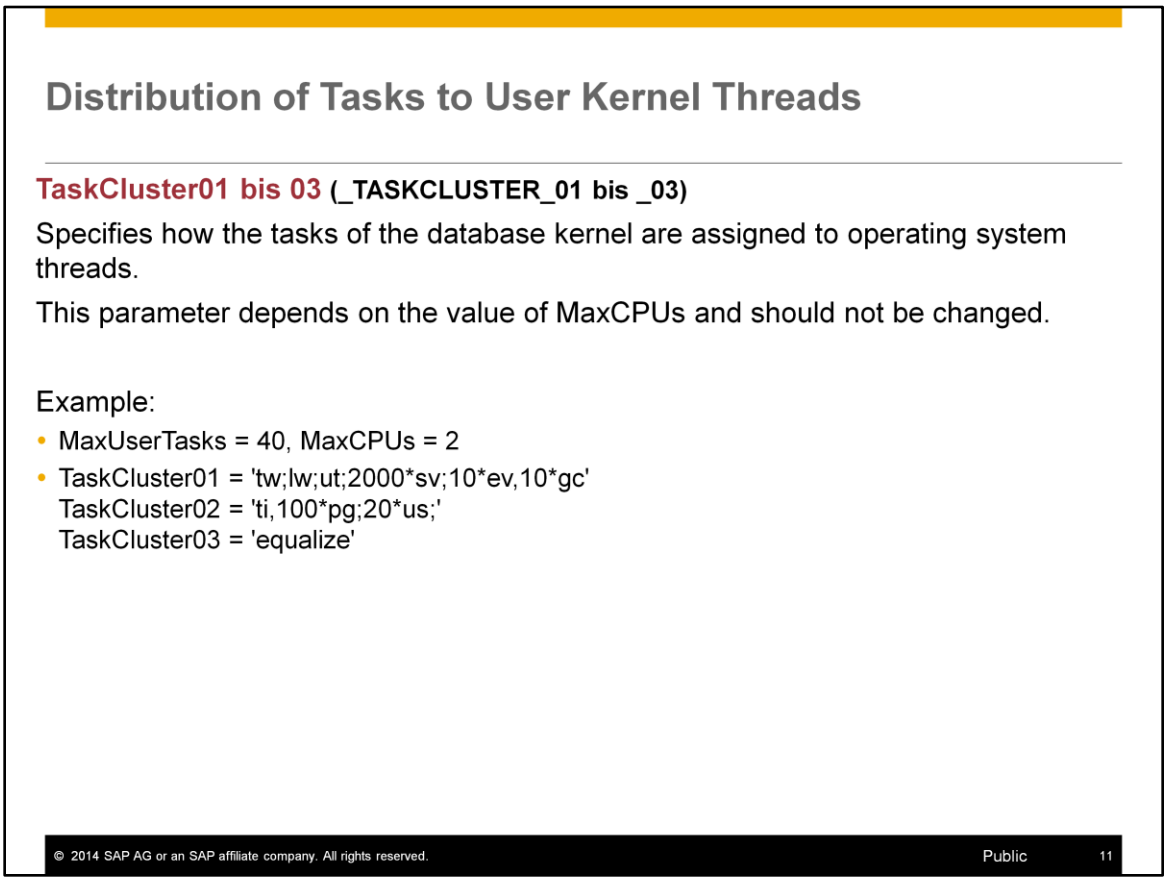
Generally speaking, MAXCPU indicates the number of CPUs simultaneously subject to intensive usage.

The value for MAXCPU strongly influences the distribution of database kernel tasks to the operating system threads (parameter TaskCluster).

If the computer is used exclusively as a database server, MaxCPUs should correspond to the number of CPUs the computer has; otherwise the value should be reduced to free up some CPUs for other applications.

Values: Default: 1

SAP central system: 1/3 - 1/5 of available CPUs
Dedicated database server with up to 7 CPUs: 100% of available CPUs
Dedicated database server with more than 7 CPUs: 100% of available CPUs -1
Online change: NO



Tasks between two semi-colons are combined to make a thread.

Meaning of the example:

Trace writer, log writer und utility task each run individually in their own thread.

Up to 2000 (practically all) server tasks are combined in a single thread.

Garbage collectors and event tasks run together in a thread.

Timer and up to 100 pagers run in one thread.

The number of threads containing user tasks is limited to MaxUserTasks/2.

"equalize":
User tasks are distributed as evenly as possible across different threads.

"compress":
The maximum possible number of user tasks (in this case 20) is processed in a thread before a new thread is started.

"allinone":
All tasks run in one thread.

Online change: NO

Warning:

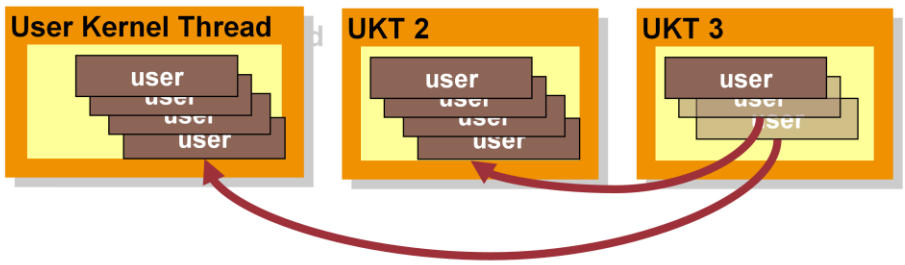
Do not change these parameters without the explicit recommendation of MaxDB Support. Any changes would be reset the next time the parameters are calculated.

Maximum Number of currently used CPUs

UseableCPUs

Limitation of the number of User Kernel Theads currently used by user tasks

The dispatcher moves user tasks from inactive User Kernel Threads to active UKT if the values of UseableCPUs is smaller than MaxCPUs



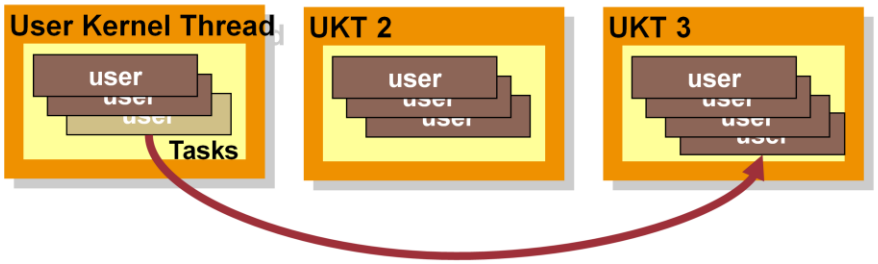
As of version 7.8 MaxDB can dynamically adjust the number of CPU cores to use. The dispatcher moves user tasks out of the inactive user kernel threads when the tasks become active.

Values: Default: MaxCPUs
Min: 1
Max: MaxCPUs
Online Change: Yes

Load Balancing Between UKTs

LoadBalancingCheckInterval (LOAD_BALANCING_CHK)

Time interval for checking the load on User Kernel Threads. If the value of the parameter is bigger than 0, tasks of an UKT with high workload can be moved to an UKT with lower load.



Load balancing enables optimal exploitation of all threads and thus of all the CPUs allocated to the database.

After the time interval of LoadBalancingCheckInterval seconds, the database kernel searches for a task to move to another UKT. This is helpful when one UKT has a particularly heavy load and another UKT has a smaller load.

Between the checks after LoadBalancingCheckInterval seconds, statistics are collected. The greater the time for gathering the data, the more meaningful the UKT load statistics that result. With smaller values, the shifting that occurs may not be optimal.

Load balancing is particularly useful for liveCache instances. These often run very CPU-intensive LCA routines over long periods. Multiple LCA routines should not work sequentially on one CPU if another CPU is free.

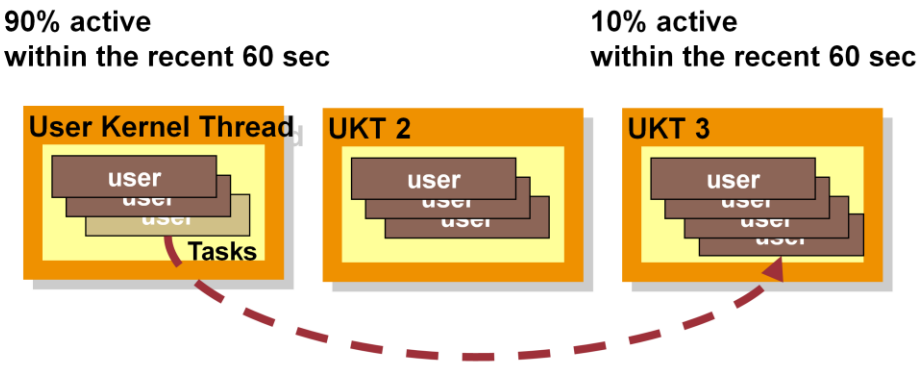
In OTLP operation, unbalanced load distribution among the UKTs is usually due to poorly-optimized statements with high CPU loads for a single job. For this reason, such statements should be identified and optimized before load balancing is employed.

Values: Default: 4
Min: 0, Max: 600
Online change: Yes

Threshold Value for Load Balancing

LoadBalancingWorkloadThreshold (LOAD_BALANCING_DIF)

Percentage of workload difference, when the distribution of tasks is started.



To assess the load on a UKT, the database kernel determines the time in which the thread is active.

The parameter LoadBalancingWorkloadThreshold indicates, in percent, the minimum difference between the CPU loads of two threads above which tasks are shifted.

From version 7.5, the database console provides information about tasks that have been moved:

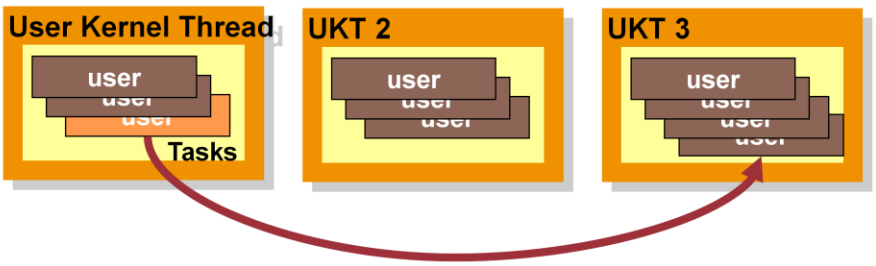
```
x_cons <dbname> show moveinfo
x_cons <dbname> show t_move t<taskid>
dbmcli –d <dbname> -u <dbmuser,passwd> -n <server> show …
```

Values: Default: 10
Min: 0, Max: 99
Online change: Yes

Precision of Measurement for Load Balancing

LoadBalancingWorkloadDeviation (LOAD_BALANCING_EQ)

Specifies the percentage the internal time values must differ before they are not treated as equal.



Moving tasks (task moving) is time-consuming. It should only be done when it is expedient.

Recorded time data for activities are considered equal if the difference between them does not exceed a certain percentage. The parameter LoadBalancingWorkloadDeviation defines this percentage.

Values: Default: 5
Min: 0, Max: 50
Online change: Yes

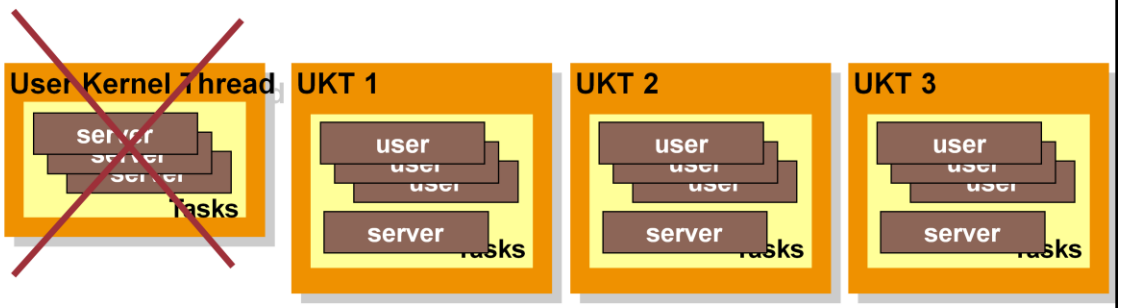
Distribution of Server Tasks to UKTs

EnableMultipleServerTaskUK (ALLOW_MULTIPLE_SERVERTASK_UKTS)

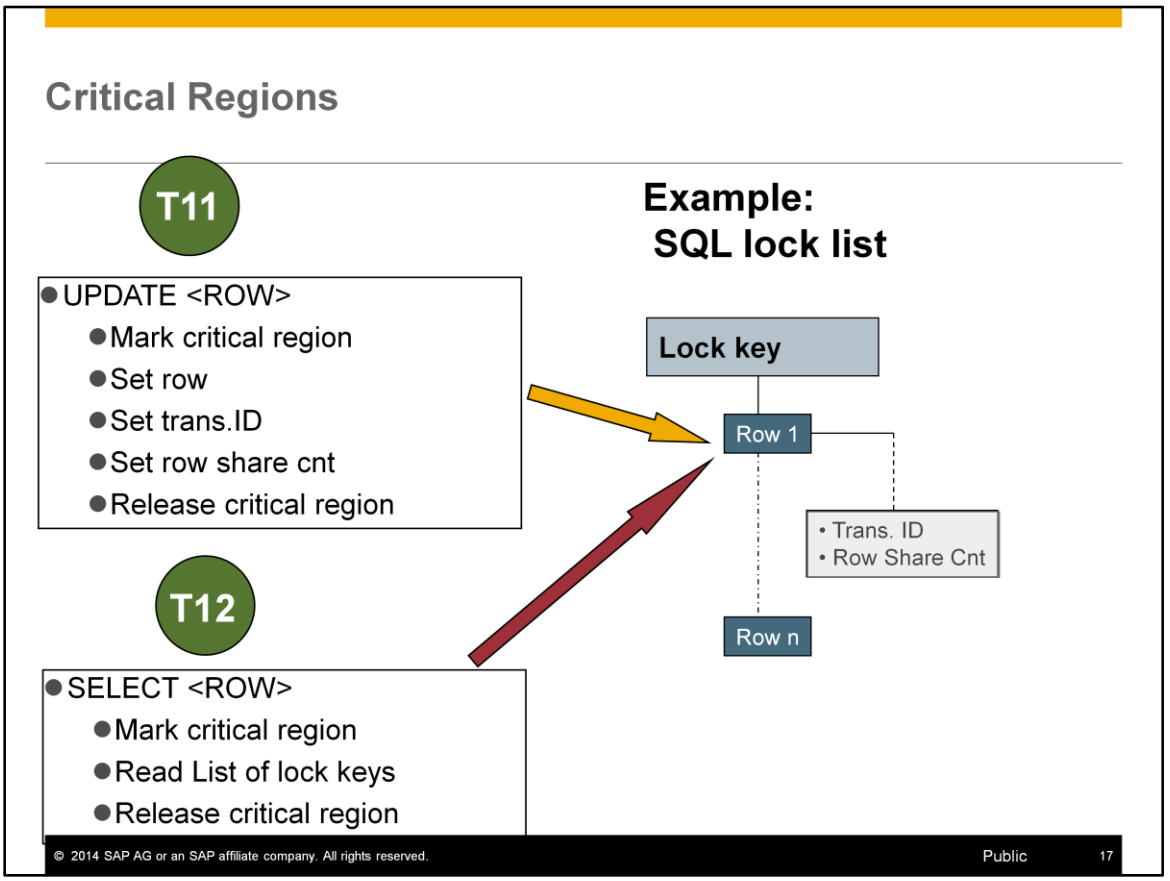
Specifies if server tasks can be distributed to UKTs containing user tasks.

Values:

- YES: Distribution of server tasks to UKTs with user tasks
- NO: Server tasks are configured within their own thread
- Default: NO

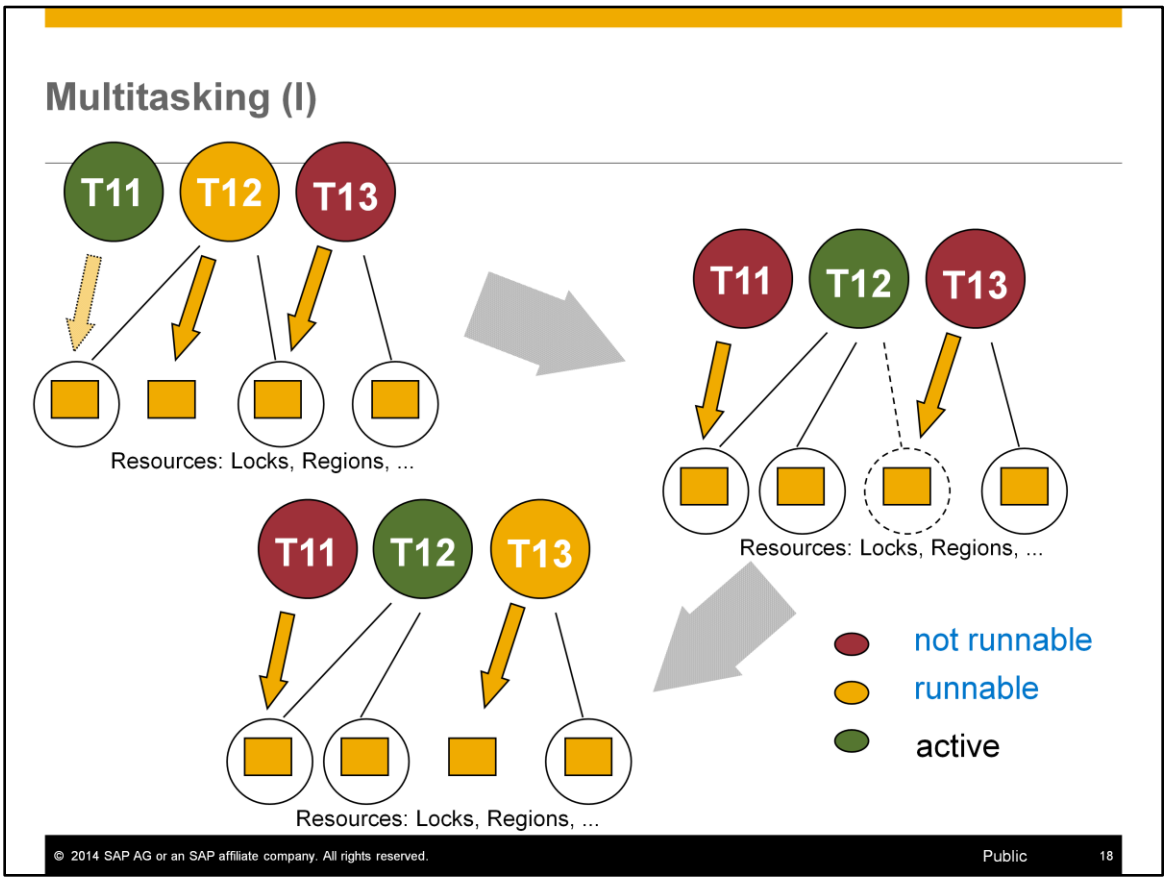


With the standard setting, server tasks are assigned to a UKT. In systems with many data volumes and fast I/O, the CPU load generated by the server tasks can lead to a bottleneck within the thread.
In liveCache instances in particular, such a bottleneck can lead to poor log recovery performance.
With the setting EnableMultipleServerTaskUKT=YES, server tasks are distributed to the UKTs for user tasks. This can prevent a CPU bottleneck in the single UTK for all server tasks.
When the server tasks are distributed among the user task UKTs, users have to share the load per CPU with the server tasks. This can lead to compromised performance in online operation, for example while a backup is in process.
Online change: NO



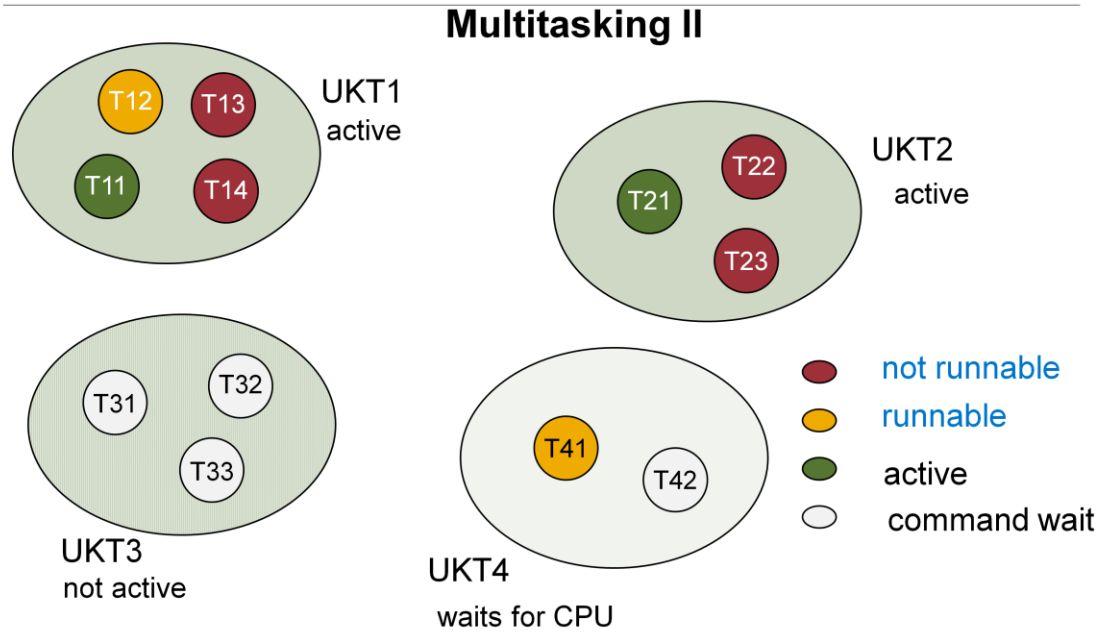
The present example should clarify the process of accessing a critical region.
Task 11 processes an update of a data record. Before it can be changed, a record has to be locked.
The lock is entered in the lock management, that is, in the lock key list, which is determined for this record by way of a hash algorithm.
During the change in the lock key list, values are changed and pointers set. While the change is taking place, the lock key list is not consistent.
Task 12, on an SMP machine, could want to read an entry from the lock key list at precisely the same time that task 11 is carrying out the change. The database does not allow this process because task 12 would be reading from an inconsistent list. So task 12 has to wait until task 11 has completed the change.
The source code, which can only be run by one task at a time, is designated a critical region. Defining regions enables synchronized access to resources that can be read or changed by multiple tasks in parallel.
Regions are only kept for a short time. The more tasks that want to access a region in parallel, the higher the risk of poor scaling. Scaling can be improved by shorter critical regions. Scaling can also be improved by shortening wait times and by defining multiple regions that allow parallel access (for example multiple lock key lists).
The command

- x_cons <SERVERDB> show regions
- dbmcli –d <dbname> -u <dbmuser,passwd> -n <server> show regions
  shows which regions have been accessed how many times.



Cooperative multitasking
Multitasking in MaxDB means "cooperative multitasking." There is no central entity responsible for dispatching tasks. Tasks and threads independently decide on activation and prioritization using a number of simple rules.
A user task stops if it has nothing to do (e.g. it is waiting for an SQL statement from the application), has to wait for I/O or cannot obtain a lock for a database object or a region. If the required lock becomes free (or another of the reasons for the stoppage is removed), the task becomes executable again. It is then put in a queue and can be activated at the next opportunity.
The illustration:
1. figure:
Task T13 is not executable because it is waiting for a lock held by T12. Task T12 is executable. Task T11 is active and is currently requesting a lock held by T12.
2. figure:
Task T11 had to stop and is not executable. T12 became active and now releases the lock for which T13 is waiting.
3. figure:
Task T13 is now executable and can become active if T12 stops and no other task is ahead of it.

Multitasking (II)

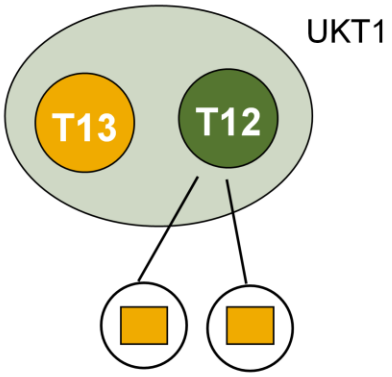


The threads UKT1 and UKT2 are running because they contain active tasks.
If task T11 stopped, T12 could become active, so UKT1 would remain active (as long as the operating system allows).
If task T21 stopped, UKT2 would no longer have an executable task and the thread would go to sleep.
UKT3 is asleep. It is not awakened by the operating system because it is not executable as long as it has no executable task.
UKT4, if it were granted CPU time by the operating system, would take off and activate task T41.
Tasks T31, T32, T33 and T42 are in the "Command wait“ state.

Stop Running Tasks

ExclusiveLockRescheduleThreshold (MAXRGN_REQUEST)

Maximum number of times a task can try to obtain any region without being interrupted by another task in the same UKT.



If, by chance, a task gets all the regions it requests and otherwise encounters no obstacles (SQL locks or I/O) over a lengthy period of time, it could be blocking other tasks. The other tasks cannot directly stop the running task. So when the number of successful region requests reaches the value set for ExclusiveLockRescheduleThreshold, the current task interrupts its work and triggers the search for another executable task within the UKT.

Small values result in more task changes. If not many users are working, this can result in unnecessary task changes. The overall cost of task switching rises.

Larger values mean longer task runtimes and fewer task changes. But blockage by "successful" tasks can occur.

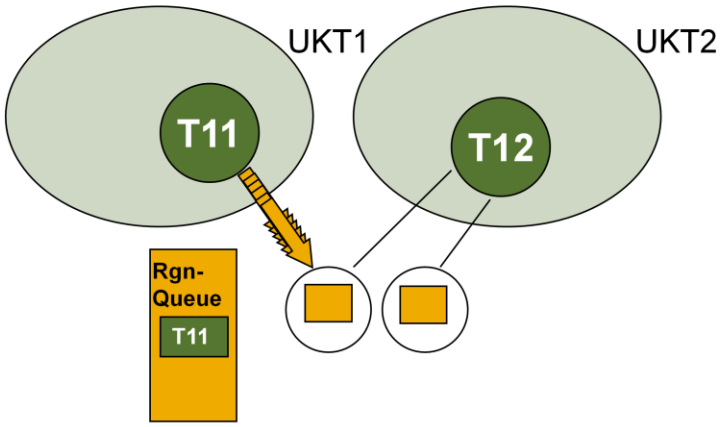
This parameter is especially important for single-processor systems.

Values	Default: MaxCPUs = 1: 300; MaxCPUs > 1: 3000 Min:100, Max: 100000 Online change: YES

Busy Looping

MaxExclusiveLockCollisionLoops (MP_RGN_LOOP)

Maximum number of attempts made to obtain a region used by another task (collision). If all attempts fail, the task is added to a specific queue.



Following a collision at a region, the task, being "executable", gets into the corresponding dispatcher queue in order to let another task become active. After it has been unsuccessful MaxExclusiveLockCollisionLoops times, the task enters into a special queue for that region in order to receive preferential access to it.

Values: Default: -1
With the default value -1, at the start the database kernel calculates:
0 if MaxCPUs = 1
100 if MaxCPUs = 2-7
10000 if MaxCPUs > 7 (starting in Version 7.6.02)
MaxExclusiveLockCollisionLoops should not be > 0 , if MaxCPUs = 1 .
Online change: YES

The optimal setting depends on the number of processors and their speed. The optimal value also depends on the speed of the operating system for IPC actions (semaphore, mutex) and thread changes.

The parameters _MP_RGN_QUEUE, _MP_RGN_DIRTY_READ and _MP_RGN_BUSY_WAIT have been removed with version 7.7. They haven‘t been changed in the systems for a long time. The removal saves some efforts for dispatching.

Dispatcher Loops

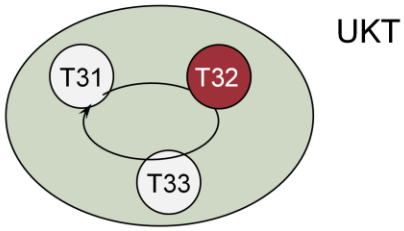
TaskSchedulerRetryLoops (_MP_DISP_LOOPS)

Maximum number of dispatcher loops to search within a UKT for a runnable task.

When this number is reached, the UKT voluntarily interrupts itself using the semop systemcall.

Values:

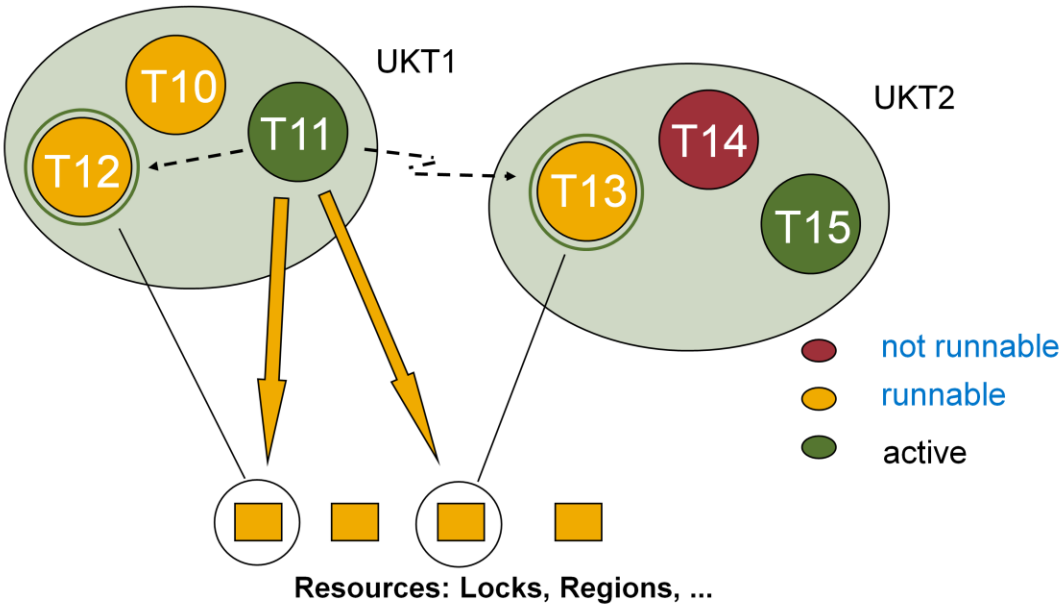
- Initial: 2
- 2 – 10,000



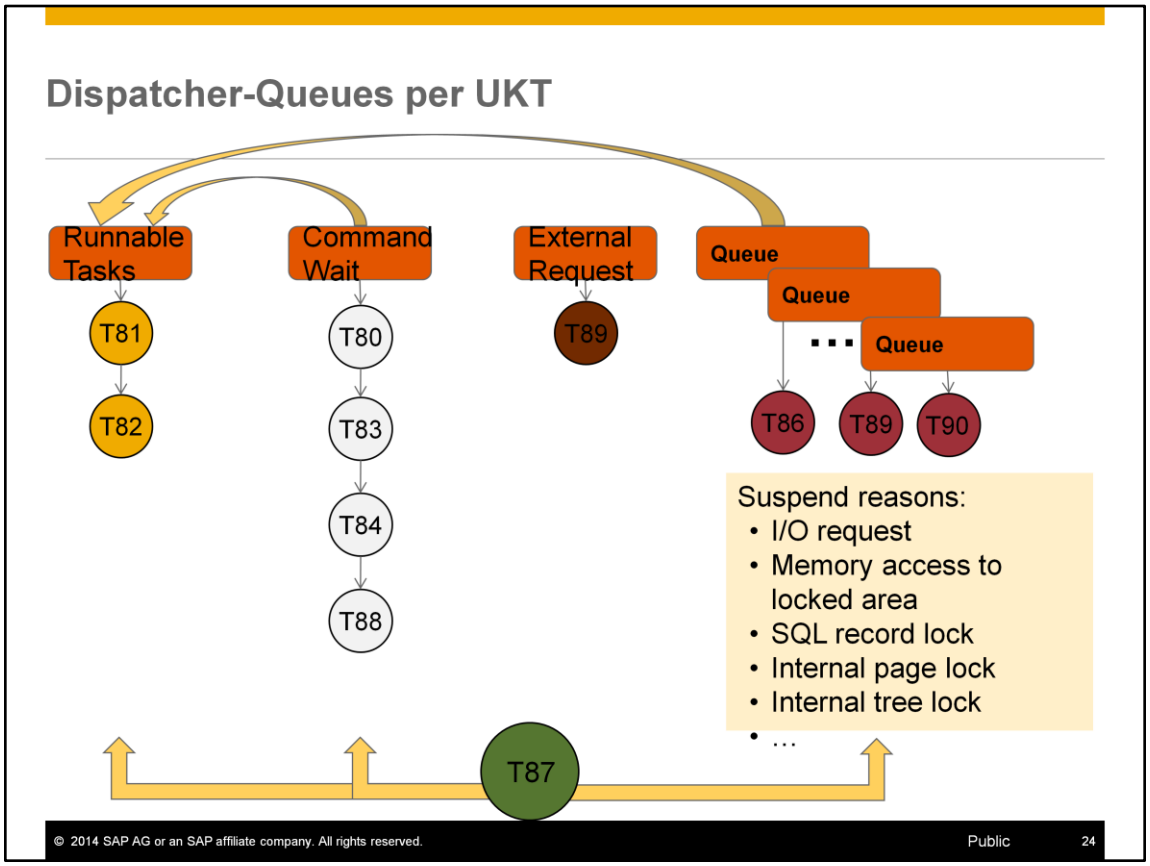
The UKT only becomes active again when a task becomes executable.
Online change: YES

Prioritization of Tasks

Which tasks are prioritized?



To ensure smooth operation of the database, you can prioritize tasks that contain resources needed by other tasks.
Exactly which tasks are prioritized and the manner of that prioritization can be set using parameters.



Tasks are assigned to certain queues existing per UKT.
Runnable tasks are waiting in a specific queue. They wait because one other task is running in this UKT. Each runnable tasks has a priority. The running task increases the priority of the runnable tasks every time it passes the dispatcher code. The runnable task with the highest priority will become the next running task. Accesses to the lists and queues inside the UKT are locking free because only one task can run at one time.
Tasks waiting for a request from the database client are assigned to the command wait queue.
Tasks which are suspended by any reason are waiting in a respective queue.
A task from another UKT or another thread can put the task id of a waiting task into the external request queue notifying that the waiting task can continue. With this method synchronized access are only needed on the external request queue. The accesses to all other queues are locking free. The dispatcher of a the UKT first tries a locking free dirty read to the external request queue and requests a lock if the queue is filled.
A running task releases control if

- it has finished it's work and sends back the result to the client; i.e. it puts itself into the command wait queue
- it has to wait for another task or thread. It puts itself into a queue according to the suspend reason
- another task is runnable and the running task has exceeded the maximum number of runs through critical sections. It put's itself to the queue of runnable tasks.

Base Priority Depending on the State

Base Priority	Previous State	Default
TaskCommunicationRequestPriority	Command Wait	Low
TaskYieldRequestPriority	Busy Loop	AboveLow
TaskResumeRequestPriority	Selbst zurückgetreten	BelowHigh
TaskLockGrantRequestPriority	Wartezustand (Suspend)	AboveHigh

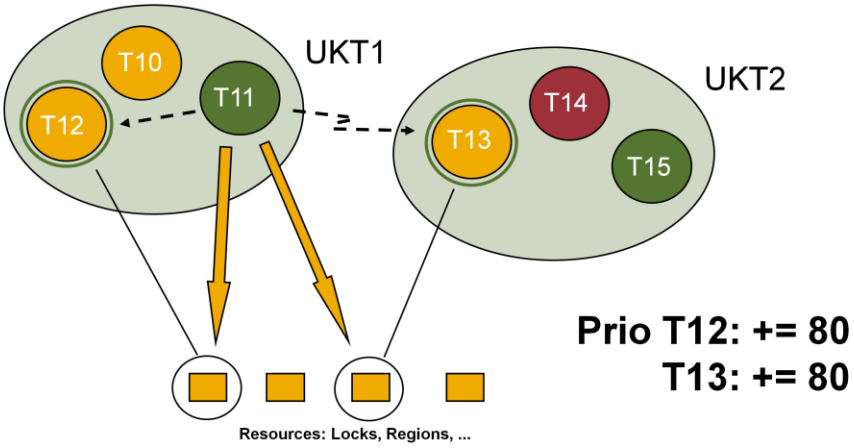
Priority Level	Default
TaskRequestPriorityLow	10
TaskRequestPriorityAboveLow	30
TaskRequestPriorityBelowNormal	50
TaskRequestPriorityNormal	80
TaskRequestPriorityAboveNormal	90
TaskRequestPriorityBelowHigh	100
TaskRequestPriorityHigh	240
TaskRequestPriorityAboveHigh	400

The dispatcher assigns base priorities to the runnable tasks according to their previous state.
Values: Possible values or base priorities are:
Low, AboveLow, BelowNormal, Normal, AboveNormal, BelowHigh, High, AboveHigh
Online Change: YES (Base priorities)

Additional Priorities for Lockholder

TaskNiceElevationValue (_PRIO_FACTOR)

Increase the base priority of tasks holding a lock by the value of TaskNiceElevationValue, when other tasks are waiting for the lock.



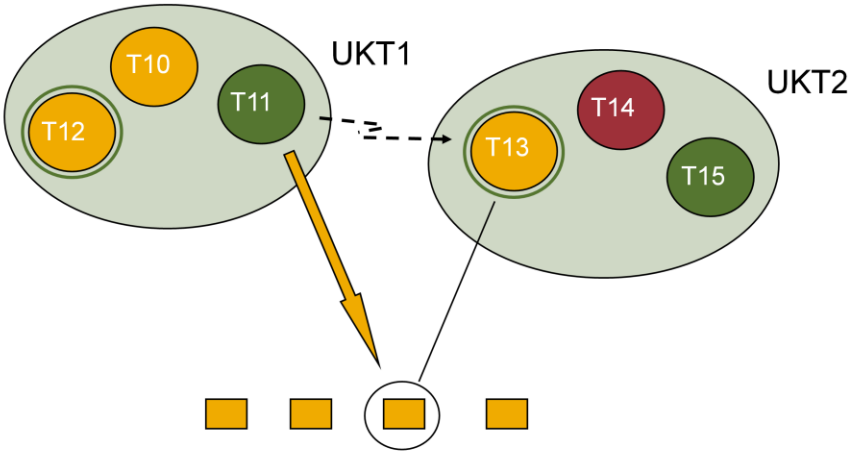
A task that holds a lock can block several other tasks. These, in turn, can block yet other tasks. This can cause undesired waits.
To reduce wait times, tasks that hold locks for which other tasks are waiting are given higher priority.
If this parameter is set too high, a long-running job can run at too high a priority. Other tasks that are not working with the locked object would then receive insufficient CPU resources; i.e. the runtimes for short queries (such as single record access) would rise.
Values: Default: 80
0 – 32,000
Online change: NO

Interruption of Tasks

ForceSchedulePrioritizedTask (_MP_DISP_PRIO)

Value 'YES':

If an inactive task gets a higher priority from a task of another UKT, the UKT of the inactive task stops its active task to activate the priority task.



In the illustration, task T11 runs into locks held by T13. That causes T13 to be given higher priority (if TaskPriorityFactor is set).
UKT2 would then have to interrupt T15 and activate T13, if ForceSchedulePrioritizedTask is set to YES. Otherwise T15 continues to run and T13 gets put into a prioritized queue.
Values: Default 'YES' if MaxCPUs > 1
'NO' if MaxCPUs = 1
Online change: YES

Questions

SAP® MaxDB™ Multi Tasking

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™ Architecture	SAP® MaxDB ™ Performance
Session 18: Introduction MaxDB Database Architecture	Session 27: SAP MaxDB Multi Tasking	Session 4: Performance Optimization with SAP MaxDB
Session 15: SAP MaxDB No-Reorganization Principle		Session 9: SAP MaxDB Optimized for SAP BW
Session 17: SAP MaxDB Shadow Page Algorithm		Session 16: SAP MaxDB SQL Query Optimization (Part 1)
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/

SAP® MaxDB™ – Expert Sessions Learning Map (3)

SAP® MaxDB ™ & Content Server

- Session 23: SAP MaxDB & Content Server Architecture
- Session 24: SAP MaxDB & Content Server Housekeeping
- Session 25: SAP MaxDB & Content Server ODBC Tracing

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: follows

Thank you

Contact information:

Christiane.Hienger@sap.com, Heike.Gursch@sap.com