SQL Server Performance Tuning Using Waits and Queues

Microsoft Corporation

July 2003

Writer: Tom Davidson
Updated: Jan 14, 2005

SQL Server Performance Tuning
Using Waits and Queues

SQL Server Performance Tuning
Using Waits and Queues

SQL Server Performance Tuning
Using Waits and Queues

Performance Overview

SQL Server Performance tuning using waits and queues is an effective mechanism for identifying and resolving application performance problems. The performance of SQL Server 2000 database applications should be evaluated from several different perspectives. Each tells a different portion of the performance story. Together they paint a detailed performance picture of the whole.

Waits are represented by SQL Server wait statistics. SQL Server 2000 tracks wait information anytime a user connection is waiting. The application requests resources and can wait for its completion. This wait information is summarized and categorized across all connections so that a performance profile can be obtained for a given work load. Thus, SQL wait types identify and categorize user (or thread) waits from an application workload or user perspective.

The **queues** part of performance is represented by PERFMON counters. The counters show performance from a resource point of view. SQL SERVER object counters are exposed to PERFMON using the system table master..sysperfinfo. Finally, associations or correlations of wait types to performance counters, as well as interesting performance counter ratios round out the picture.

Each PERFMON object has counters that are used to measure various aspects of performance, such as transfer rates for disks or the amount of processor time consumed for processors. Perfmon counters (including system, physical disk, etc.) provide a view of performance from a resource standpoint while SQL waits provide a view of performance from a user connection (or application) perspective.

Correlations of wait types to perf counters, and specific ratios of perfmon counters form the basis for an application performance methodology called waits and queues.

Waits & Queues: A Performance Methodology

Application performance can be simply explained by looking at waits and queues. *Dbcc sqlperf(waitstats)* provides a valuable source of wait information from a thread (or application) point of view. PERFMON on the other hand, provides a breakdown of system resource usage in terms of resource queues.

SQL Server Performance Tuning
Using Waits and Queues

Requests for system resources such as IO, are made by user connections or threads. If those requests cannot be immediately satisfied, a queue of requests will wait until resources are available.

Wait Types

If a thread goes into a sleep status, a wait state is set. The wait state is contained in master..sysprocesses in the columns waittype, and lastwaittype. Lastwaittype is a character description of the last wait state for this thread. It is not reset until another wait state occurs. Waittype is a varbinary wait state that is the current wait state. A wait time of 0 means the thread is currently running.

Sysprocesses

Each user has an associated row in the system table master..sysprocesses. The stored procedure sp_who provides a list of these user connections or threads as well as other connection information such as command, resource, wait types, wait time and status. When a thread waits, the columns ***waittype (binary(2)), waittime (int)*** and ***lastwaittype (nchar(32)) and waitresource.***. The values for waittype and lastwaittype columns are set by memory structures in SQL Server.

Lastwaittype is a character description of the last wait type for this thread. It is not reset until another wait state occurs. Thus, a non-blank **lastwaittype** means the thread had at least one wait state.

The current wait status is recorded in the **waittype** column. If the waittype is non-zero, the lastwaittype and waittype will be equivalent and indicate the current waitstate for the SPID. If **waittype** is 0x00, this means the thread is currently running.

Track_waitstats stored procedure

Track_waitstats is a stored procedure that will capture waitstats from DBCC SQLPERF, and provide a ranking of descending order based on percentage. This is useful in identifying the greatest opportunites for performance improvements. See the sample output below:

```
CREATE proc track_waitstats (@num_samples int=10,@delaynum int=1,@delaytype nvarchar(10)='minutes')
as
--
-- This stored procedure is provided "AS IS" with no warranties, and confers no rights.
```

```
-- Use of included script samples are subject to the terms specified at http://www.microsoft.com/info/cpyright.htm
--
-- T. Davidson
-- @num_samples is the number of times to capture waitstats, default is 10 times
-- default delay interval is 1 minute
-- delaynum is the delay interval - can be minutes or seconds
-- delaytype specifies whether the delay interval is minutes or seconds
-- create waitstats table if it doesn't exist, otherwise truncate
--
set nocount on
if not exists (select 1 from sysobjects where name = 'waitstats')
        create table waitstats ([wait type] varchar(80),
                requests numeric(20,1),
                [wait time] numeric (20,1),
                [signal wait time] numeric(20,1),
                now datetime default getdate())
else    truncate table waitstats
dbcc sqlperf (waitstats,clear)  -- clear out waitstats
declare @i int,@delay varchar(8),@dt varchar(3), @now datetime, @totalwait numeric(20,1)
        ,@endtime datetime,@begintime datetime
        ,@hr int, @min int, @sec int
select @i = 1
select @dt = case lower(@delaytype)
        when 'minutes' then 'm'
        when 'minute' then 'm'
        when 'min' then 'm'
        when 'mm' then 'm'
        when 'mi' then 'm'
        when 'm' then 'm'
        when 'seconds' then 's'
```

```
        when 'second' then 's'
        when 'sec' then 's'
        when 'ss' then 's'
        when 's' then 's'
        else @delaytype
end
if @dt not in ('s','m')
begin
        print 'please supply delay type e.g. seconds or minutes'
        return
end
if @dt = 's'
begin
        select @sec = @delaynum % 60
        select @min = cast((@delaynum / 60) as int)
        select @hr = cast((@min / 60) as int)
        select @min = @min % 60
end
if @dt = 'm'
begin
        select @sec = 0
        select @min = @delaynum % 60
        select @hr = cast((@delaynum / 60) as int)
end
select @delay= right('0'+ convert(varchar(2),@hr),2) + ':' +
        + right('0'+convert(varchar(2),@min),2) + ':' +
        + right('0'+convert(varchar(2),@sec),2)
if @hr > 23 or @min > 59 or @sec > 59
begin
        select 'hh:mm:ss delay time cannot > 23:59:59'
```

```
        select 'delay interval and type: ' + convert (varchar(10),@delaynum) + ',' + @delaytype + ' converts to ' + @delay
        return
end
while (@i <= @num_samples)
begin
         insert into waitstats ([wait type], requests, [wait time],[signal wait time])
        exec ('dbcc sqlperf(waitstats)')
        select @i = @i + 1
        waitfor delay @delay
end
--- create waitstats report
execute get_waitstats
go
exec track_waitstats 20,15,'seconds'            ----- take 20 samples (run 5 minutes), gather waitstats every 15 seconds
```

Get_waitstats stored procedure

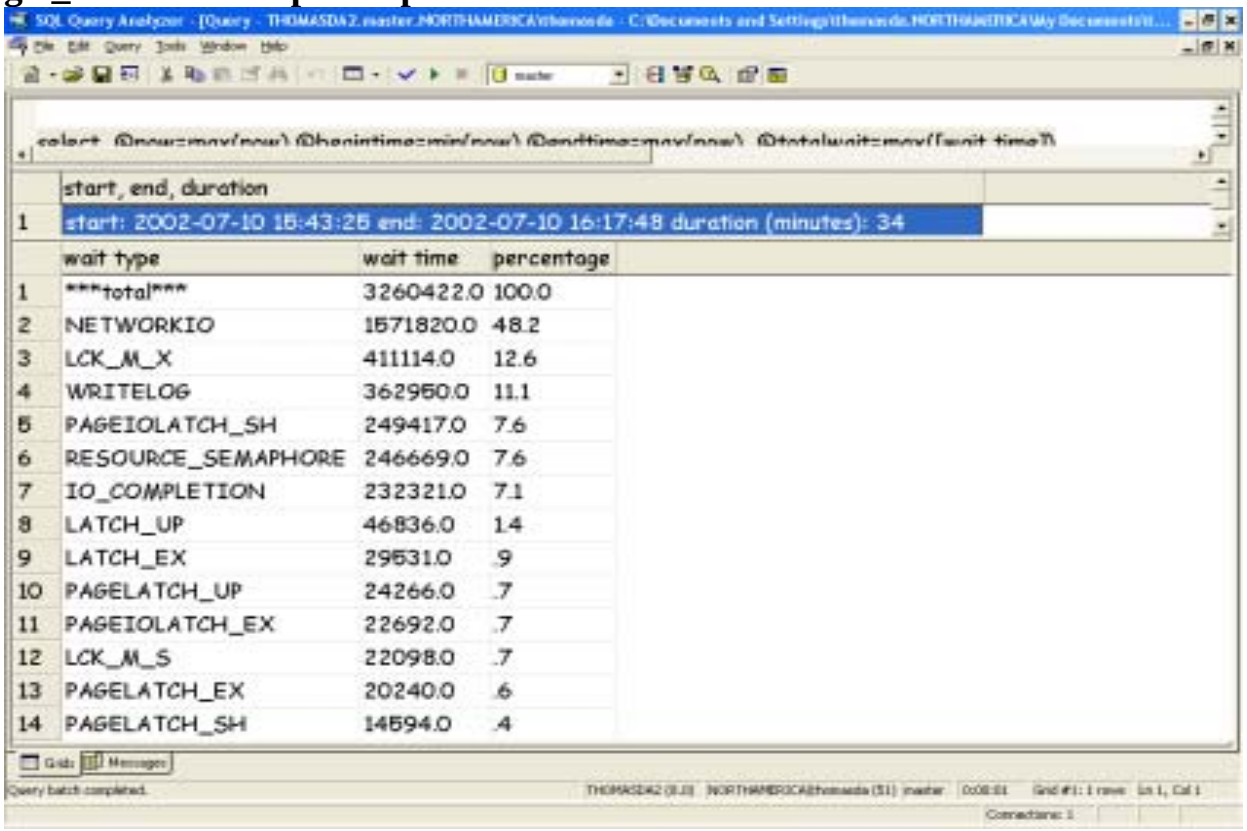
The stored procedure get_waitstats can be run during the execution of track_waitstats or after track_waitstats completes, to provide a report of wait stats.

```
CREATE proc get_waitstats (@report_format varchar(20)='all')
as
-- This stored procedure is provided "AS IS" with no warranties, and confers no rights.
-- Use of included script samples are subject to the terms specified at http://www.microsoft.com/info/cpyright.htm
--
-- this proc will create waitstats report listing wait types by percentage.
--      (1) total wait time is the sum of resource & signal waits, @report_format='all' reports resource & signal
--      (2) Basics of execution model (simplified)
--          a. spid is running then needs unavailable resource, moves to resource wait list at time T0
```

```
--          b. a signal indicates resource available, spid moves to runnable queue at time T1
--          c. spid awaits running status until T2 as cpu works its way through runnable queue in order of arrival
--      (3) resource wait time is the actual time waiting for the resource to be available, T1-T0
--      (4) signal wait time is the time it takes from the point the resource is available (T1)
--          to the point in which the process is running again at T2.  Thus, signal waits are T2-T1
--      (5) Key questions: Are Resource and Signal time significant?
--          a. Highest waits indicate the bottleneck you need to solve for scalability
--          b. Generally if you have LOW% SIGNAL WAITS, the CPU is handling the workload e.g. spids spend move through runnable queue quickly
--          c. HIGH % SIGNAL WAITS indicates CPU can't keep up, significant time for spids to move up the runnable queue to reach running status
--      (6) This proc can be run when track_waitstats is executing
set nocount on

declare @now datetime, @totalwait numeric(20,1), @totalsignalwait numeric(20,1), @totalresourcewait numeric(20,1)
        ,@endtime datetime,@begintime datetime
        ,@hr int, @min int, @sec int

select  @now=max(now),@begintime=min(now),@endtime=max(now)
from waitstats where [wait type] = 'Total'

--- subtract waitfor, sleep, and resource_queue from Total
select @totalwait = sum([wait time]) + 1, @totalsignalwait = sum([signal wait time]) + 1 from waitstats
where [wait type] not in ('WAITFOR','SLEEP','RESOURCE_QUEUE', 'Total', '***total***') and now = @now

select @totalresourcewait = @totalwait - @totalsignalwait
-- insert adjusted totals, rank by percentage descending
delete waitstats where [wait type] = '***total***' and now = @now
insert into waitstats select '***total***',0,@totalwait,@totalsignalwait,@now
select 'start time'=@begintime,'end time'=@endtime,'duration (hh:mm:ss:ms)'=convert(varchar(50),@endtime-@begintime,14)
```

```
if @report_format = 'all'
	select [wait type],[requests]
		,'Total wt (T2-T0)'=[wait time], 'wt_%'=cast (100*[wait time]/@totalwait as numeric(20,1))
		,'Resource wt (T1-T0)'=[wait time]-[signal wait time]
		,'res_wt_%'=cast (100*([wait time] - [signal wait time]) /@totalwait as numeric(20,1))
		,'Signal wt (T2-T1)'=[signal wait time]
		,'sig_wt_%'=cast (100*[signal wait time]/@totalwait as numeric(20,1))
	from waitstats
	where [wait type] not in ('WAITFOR','SLEEP','RESOURCE_QUEUE','Total')
	and now = @now
	order by 'wt_%' desc
else
	select [wait type],[wait time],percentage=cast (100*[wait time]/@totalwait as numeric(20,1))
	from waitstats
	where [wait type] not in ('WAITFOR','SLEEP','RESOURCE_QUEUE','Total')
	and now = @now
	order by percentage desc
GO
exec get_waitstats 'all'
```

get_waitstats Sample output



start, end, duration

1	start: 2002-07-10 15:43:26 end: 2002-07-10 16:17:48 duration (minutes): 34

	wait type	wait time	percentage
1	***total***	3260422.0	100.0
2	NETWORKIO	1571820.0	48.2
3	LCK_M_X	411114.0	12.6
4	WRITELOG	362960.0	11.1
5	PAGEIOLATCH_SH	249417.0	7.6
6	RESOURCE_SEMAPHORE	246669.0	7.6
7	IO_COMPLETION	232321.0	7.1
8	LATCH_UP	46836.0	1.4
9	LATCH_EX	29531.0	.9
10	PAGELATCH_UP	24266.0	.7
11	PAGEIOLATCH_EX	22692.0	.7
12	LCK_M_S	22098.0	.7
13	PAGELATCH_EX	20240.0	.6
14	PAGELATCH_SH	14594.0	.4

The above sample shows the lion's share of wait time, 48%, being due to network IO waits. Improving network IO is the single largest opportunity for improving application performance.

Other lesser opportunities in the above example include LCK_M_X (exlusive locks) and WRITELOG (transaction log). Exclusive lock waits account for almost 13% of total wait time. An examination of transaction management may offer clues as to whether improvements can be made here.

WRITELOG means threads are waiting for physical writes to complete to the transaction log. Given the 11% writelog waits, a further analysis of PERFMON disk queues for the transaction log will confirm whether the IO capacity of the transaction log drives have trouble keeping up with write requests as shown by steady and high disk queues.

The following table contains wait types, descriptions and correlation to other performance information. It is not exhaustive but will point you in the right direction. The important idea is that if track_waitstats shows a significant amount (by percentage) of a given wait type, you should corroborate this clue with the correlated information. The PERFMON counter table follows with descriptions, additional correlations, and possible conclusions and actions.

Wait Types and correlation to other Performance info

Wait Type	Category	Description	Correlation to Other info
ASYNC_DISKPOOL_LOCK	IO (Restore DB)	RARE During Backup and Restore (e.g. including zeroing out pages) threads written in parallel.	Possible disk bottleneck. See disk perf counters for confirmation.
ASYNC_IO_COMPLETION	IO	Waiting for asynchronous IO requests to complete. Identify disk bottlenecks, using PERF Counters, Profiler, ::fn_virtualfilestats and SHOWPLAN Any of the following will reduce these waits: 1. Adding additional IO bandwidth, 2. Balancing IO across other drives	See PERFMON Disk perf counters: 1. Disk sec/read 2. Disk sec/write 3. Disk queues See PERFMON SQL Buffer Cache perf counters for memory pressure: 1. Page Life Expectancy 2. Checkpoint pages/sec 3. Lazywrites/sec

		3. Reducing IO with proper indexing 4. Check for bad query plans 5. Check for memory pressure	See PERFMON SQL Access Methods for correct indexing: 1. Full Scans/sec 2. Index seeks/sec Check IoStallMS – IoStallMS is the number of cumulative milliseconds of IO waits for a particular file. If IoStallMS is inordinately high for one or more files, you have a disk bottleneck. 1. select * from ::fn_virtualfilestats (dbid,file#) 2. select * from ::fn_virtualfilestats (dbid,-1) to list all files for a database. SQL Profiler can be used to identify which TSQL statements do scans. Select the **scans** event class & events **scan:started** and **scan:completed.** Include the **object Id** data column. Save the profiler trace to a trace table, and then search for the **scans** event. The **scan:completed** event will provide associated IO so you can also search for high reads, writes, and duration. Check SHOWPLAN for bad query

			plans
CMEMTHREAD		Waiting for thread safe memory objects	
CURSOR		Asynch Cursor thread	
CXPACKET		Parallel process waits. Possible skew of data possible lock of a range for this cpu meaning one parallel process is behind, etc. In an OLTP environment, excessive CXPACKET waits can impact the throughput of other OLTP traffic. In a DW environment, CXPACKET waits are expected for multiple proc environments.	Check for parallelism – sp_Configure "max degree of parallelism". If max degree of parallelism = 0, you may want to do one of the following: 1. turn off parallelism entirely: set max degree of parallelism to 1 2. limit parallelism by setting max degree of parallelism to some number less than the total number of CPUs. For example if you have 8 procs, set max degree of parallelism to <=4.
DBTABLE		New Checkpoint request that is waiting for outstanding checkpoint request to complete	See SQL Buffer Cache perf counters: 1. Page Life Expectancy 2. Checkpoint pages/sec 3. Lazywrites/sec
DTC		Waiting for Distributed Transaction Coordinator	Check transaction isolation level
EC		Non-parallel synchronization between parent and child thread	
EXCHANGE		Waiting on a parallel process to complete, shutdown or startup.	Check for parallelism – sp_Configure "max degree of parallelism".

			If max degree of parallelism = 0, you may want to do one of the following: 1. turn off parallelism entirely: set max degree of parallelism to 1 2. limit parallelism by setting max degree of parallelism to some number less than the total number of CPUs. For example if you have 8 procs, set max degree of parallelism to <=4.
EXECSYNC		Query memory and spooling to disk	
IO_COMPLETION	IO	Waiting for IO requests to complete. Identify disk bottlenecks, using PERF Counters, Profiler, ::fn_virtualfilestats and SHOWPLAN Any of the following will reduce these waits: 1. Adding additional IO bandwidth, 2. Balancing IO across other drives 3. Reducing IO with proper indexing 4. Check for bad query plans	See Disk perf counters: 1. Disk sec/read 2. Disk sec/write 3. Disk queues See SQL Buffer Cache perf counters: 1. Page Life Expectancy 2. Checkpoint pages/sec 3. Lazywrites/sec See SQL Access Methods for correct indexing: 1. Full Scans/sec 2. Index seeks/sec See memory perf counter 1. Page faults/sec

			Check IoStallMS 1. select * from ::fn_virtualfilestats(dbid,file#) SQL Profiler can be used to identify which TSQL statements do scan. Select the **scans** event class & events **scan:started** and **scan:completed.** Include the **object Id** data column. Save the profiler trace to a trace table, and then search for the **scans** event. The **scan:completed** event will provide associated IO so you can also search for high reads, writes, and duration. Check SHOWPLAN for bad query plans

LATCH_x	Latch	Latches are short term light weight synchronization objects. Latches are not held for the duration of a transaction. "Plain" latches are generally not related to IO. These latches can be used for a variety of things, but they are not used to synchronize access to buffer pages (PAGELATCH_x is used for that). Possibly the most common case is contention on internal caches (not the buffer pool pages), especially when using heaps and/or text.	If high, check PERFMON for 1. memory pressure 2. SQL Latch waits (ms) Look for LOG and Pagelatch_UP wait types. Latch_x waits can often be alleviated by solving LOG and PAGELATCH_UP contention. In the absence of LOG and/or PAGELATCH_UP contention, the only other option is to partition the table/index in question in order to create multiple caches (the caches are per-index).
LATCH_DT	Latch	Destroy Latch	See LATCH_x
LATCH_EX	Latch	Exclusive Latch	See LATCH_x
LATCH_KP	Latch	Keep Latch	See LATCH_x
LATCH_NL	Latch	Null Latch	See LATCH_x
LATCH_SH	Latch	Shared Latch	See LATCH_x
LATCH_UP	Latch	Update Latch	See LATCH_x
LCK_x	Lock	Possible transaction management issue. 1. For shared locks, check Isolation level for transaction. 2. Keep transaction as short as possible	See SQL Locks perf counters 1.Lock wait time (ms) Hint: check for memory pressure, which causes more physical IO, thus prolonging the duration of transactions and locks.
LCK_M_BU	Lock	Bulk update lock	See Lck_x
LCK_M_IS	Lock	Intent Share lock	See Lck_x

LCK_M_IU	Lock	Intent Update lock	See Lck_x
LCK_M_IX	Lock	Intent Exclusive lock	See Lck_x
LCK_M_RIn_NL	Lock	Range Intent Null Lock	See Lck_x
LCK_M_RIn_S	Lock	Range Intent Shared lock	See Lck_x
LCK_M_RIn_U	Lock	Range Intent Update lock	See Lck_x
LCK_M_RIn_X	Lock	Range Intent Exclusive lock	See Lck_x
LCK_M_RS_S	Lock	Range Shared Shared (Key-Range) lock	See Lck_x
LCK_M_RS_U	Lock	Range Shared Update (key-range) lock	See Lck_x
LCK_M_RX_S	Lock	Range Exclusive shared (key-range)	See Lck_x
LCK_M_RX_U	Lock	Range Exclusive update (key-range) lock	See Lck_x
LCK_M_RX_X	Lock	Range Exclusive Exclusive (key-range)	See Lck_x
LCK_M_S	Lock	Shared Lock:	See Lck_x
LCK_M_SCH_M	Lock	Modify schema lock:	See Lck_x
LCK_M_SCH_S	Lock	Shared Schema (Stability) lock	See Lck_x
LCK_M_SIU	Lock	Share Intent Update lock	See Lck_x
LCK_M_SIX	Lock	Share Intent Exclusive lock	See Lck_x
LCK_M_U	Lock	Update lock.	See Lck_x
LCK_M_UIX	Lock	Update intent exclusive lock	See Lck_x
LCK_M_X	Lock	Exclusive lock	See Lck_x
LOGMGR	Transaction Log	Waiting for write requests to the transaction log to complete. Identify disk bottlenecks, using PERF Counters, Profiler, ::fn_virtualfilestats and SHOWPLAN Any of the following will reduce these waits: 1. Adding additional IO bandwidth, 2. Balancing IO across other drives 3. Moving / Isolating the transaction log on its own drive	See Disk perf counters: 1. Disk sec/read 2. Disk sec/write 3. Disk queues See SQL Buffer Cache perf counters: 1. Page Life Expectancy 2. Checkpoint pages/sec 3. Lazywrites/sec Check IoStallMS for tranlog 1. select *

			from ::fn_virtualfilestats(dbid,file#)
MISCELLANEOUS		Catch all wait type	
NETWORKIO		Waiting on Network IO Completion. Waiting to read or write to a client on the network This can occur if a client is in the middle of sending packets to SQL Server, or when SQL writes data to a client and is waiting for an ACK.	Check NIC bandwidth. 100mbits is preferable to 10mbs.
OLEDB		OLEDB waits. Common causes are: • SQL Server is waiting for client application to send data. Some examples include: • 1. BULK INSERT • 2. CONVERT (6.5 to 2000) • 3. Full text • 4. Linked server calls incl. four part name calls, remote procedure calls, openquery, openrowset etc. • 5. Queries that access virtual tables, since these are implemented as OLEDB rowset providers. • 6. Heavy use of Profiler	1. Check placement of client app including any file input read by the client and SQL Server data and log files. See PERFMON disk secs/read & disk secs/write. If disk secs/read are high, you may add additional IO bandwidth, balance IO across other drives, or move / isolate the database and transaction log on its own drives 2. Inspect TSQL code for RPC, Distributed (Linked Server) & Full Text Search. While SQL server supports these type queries, they are sometimes performance bottlenecks. 3. To get the SQL Statement involved in OLEDB waits, Select virtual table master..sysprocesses as follows: a. SQL2000 Service Pack 3 Only

			DECLARE @Handle binary(20) SELECT @Handle = sql_handle FROM sysprocesses WHERE waittype = 0x0042 SELECT * FROM ::fn_get_sql(@Handle) b. SQL2000 RTM, SP1, SP2 – limited to 255 characters dbcc inputbuffer (spid)
PAGEIOLATCH_x		Latches are short term synchronization objects. used to synchronize access to buffer pages. PageIOLatch is used for disk to memory transfers.	If this is significant in percentage, it normally suggests disk IO subsystem issues. Check disk counters.
PAGEIOLATCH_DT		IO Page destroy latch	See PAGEIOLATCH_x
PAGEIOLATCH_EX		IO Page latch exclusive	See PAGEIOLATCH_x
PAGEIOLATCH_KP		IO Page latch keep	See PAGEIOLATCH_x
PAGEIOLATCH_NL		IO Page latch null	See PAGEIOLATCH_x
PAGEIOLATCH_SH		IO Page latch shared	See PAGEIOLATCH_x
PAGEIOLATCH_UP		IO Page latch update	See PAGEIOLATCH_x
PAGELATCH_x		Latches are short term light weight synchronization objects. Latches are not held for the duration of a transaction. Typical latching operations during row transfers to memory, controlling modifications to row offset table, etc. Consequently, the duration of latches is normally sensitive to available memory.	If this is significant in percentage, it normally indicates cache contention.
PAGELATCH_DT		Page latch	See PAGELATCH_x
PAGELATCH_EX		Page latch exclusive Contention can be caused by issues other	See PAGELATCH_x

		than IO or memory performance, for example, heavy concurrent inserts into the same index range can cause this type of contention. If a lot of inserts need to be placed on the same page they are serialized using the latch. A lot of inserts into the same range can also cause page splits in the index which will hold onto the latch while allocating a new page (this can take a while). Any read accesses to the same range as the inserts would also conflict on the latches. The solution in these cases is to distribute the inserts using a more appropriate	
PAGELATCH_KP		Page latch keep	See PAGELATCH_x
PAGELATCH_NL		Page latch null	See PAGELATCH_x
PAGELATCH_SH		Page latch shared Contention can be caused by issues other than IO or memory performance, for example, heavy concurrent inserts into the same index range can cause this type of contention. If a lot of inserts need to be placed on the same page they are serialized using the latch. A lot of inserts into the same range can also cause page splits in the index which will hold onto the latch while allocating a new page (this can take a while). Any read accesses to the same range as the inserts would also conflict on the latches. The solution in these cases is to distribute the inserts using a more appropriate	See PAGELATCH_x
PAGELATCH_UP		Page latch Update is used only for allocation related pages, and contention on it is often a sign that more files are needed. With multiple files, allocations can be distributed across multiple files thus reducing demand on the per-file data structures stored on these pages. The contention is not IO performance, but	See PAGELATCH_x

		rather internal allocation contention to access the pages – adding more spindles to a file or moving the file to a faster disk will not help, nor will adding more memory.	
PAGESUPP		Waits for parallel page supplier. Possible disk bottleneck Any of the following will reduce these waits: 1. Adding additional IO bandwidth, 2. Balancing IO across other drives 3. Reducing IO with proper indexing 4. Check for bad query plans	See Disk perf counters: 1. Disk sec/read 2. Disk sec/write 3. Disk queues See SQL Buffer Cache perf counters: 1. Page Life Expectancy 2. Checkpoint pages/sec 3. Lazywrites/sec Check IoStallMS for database 1. select * from ::fn_virtualfilestats(dbid,file#)
PIPELINE_INDEX_STAT		PIPELINE waittypes added to allow one user to perform multiple operations such as writes to log cache on behalf of himself as well as other users who are waiting for same operation. It does all log writes in single operation.	See Disk perf counters: 1. Disk sec/read 2. Disk sec/write 3. Disk queues See SQL Buffer Cache perf counters: 1. Page Life Expectancy 2. Checkpoint pages/sec 3. Lazywrites/sec Check IoStallMS for database 1. select * from ::fn_virtualfilestats(dbid,file#)
PIPELINE_LOG		PIPELINE waittypes added to allow one	See Disk perf counters:

		user to perform multiple operations such as writes to log cache on behalf of himself as well as other users who are waiting for same operation. Does in single operation.	1. Disk sec/read 2. Disk sec/write 3. Disk queues See SQL Buffer Cache perf counters: 1. Page Life Expectancy 2. Checkpoint pages/sec 3. Lazywrites/sec Check IoStallMS for database 1. select * from ::fn_virtualfilestats(dbid,file#)
PIPELINE_VLM		PIPELINE waittypes added to allow one user to perform multiple operations such as writes to log cache on behalf of himself as well as other users who are waiting for same operation. Does in single operation.	See Disk perf counters: 1. Disk sec/read 2. Disk sec/write 3. Disk queues See SQL Buffer Cache perf counters: 1. Page Life Expectancy 2. Checkpoint pages/sec 3. Lazywrites/sec Check IoStallMS for database 1. select * from ::fn_virtualfilestats(dbid,file#)
PSS_CHILD		Waiting on Asynch thread	
RESOURCE_QUEUE		Internal Use only	
RESOURCE_SEMAPHORE		COMMON for DSS like workload & large queries such as hash joins; must wait for memory quota (grant) prior to execution.	See SQL Memory Mgr perf counters 1. Memory Grants Pending

			2. Memory Grants Outstanding
SHUTDOWN		Shutdown without specifying NOWAIT, waits for other users to logout before shutdown completes	Monitory SQL Statistics: User Connections To expedite shutdown you can: 1. SHUTDOWN WITH NOWAIT 2. Use SQL Kill command to terminate user connections.
SLEEP		Internal Use only	
TEMPOBJ		Dropping a global temp object that is being used by others.	
TRAN_MARK_DT		Transaction latch - destroy	
TRAN_MARK_EX		Transaction latch - Exclusive	
TRAN_MARK_KP		Transaction latch - Keep page	
TRAN_MARK_NL		Transaction latch - Null	
TRAN_MARK_SH		Transaction latch - Shared	
TRAN_MARK_UP		Transaction latch - Update	
UMS_THREAD		Batch waiting on a worker thread to free up (or batch waiting to get a worker thread to run it)	If this is a high percentage, you can increase the number of worker threads from the default of 255. The maximum is 1024.
WAITFOR	Waitfor	Check for waitfor delay in TSQL code	Inspect TSQL code for “waitfor delay” statement
WRITELOG	Transaction Log	Waiting for write requests to the transaction log to complete. Identify disk bottlenecks, using PERF Counters, Profiler, ::fn_virtualfilestats and SHOWPLAN Any of the following will reduce these waits: 1. Adding additional IO bandwidth,	See Disk perf counters: 1. Disk sec/read 2. Disk sec/write 3. Disk queues See SQL Buffer Cache counters: 1. Page Life Expectancy 2. Checkpoint pages/sec 3. Lazywrites/sec

		2. Balancing IO across other drives 3. Moving / Isolating the transaction log on its own drive	Check IoStallMS for tranlog 1. select * from ::fn_virtualfilestats(dbid,file#)
XACTLOCKINFO		Transaction escalation, rollback	

QUEUES (Perfmon Counters)

The Queues aspect of the Waits and Queues approach to performance analysis refers to PERFMON counters. PERFMON counters provide a view of system performance from a resource standpoint.

PERFMON Counters, correlation, possible conclusions & actions

Resource Component	Perfmon Object	Counters to Monitor	Description	Possible conclusions / actions
Disk	Physical Disk	Current Queue Length	Sustained high queues mean your IO subsystem is not keeping up.	Confirm IO issues with disk sec/read & disk sec/write. Waitstats correlation: 1. IO_COMPLETION 2. ASYNC_IO_COMPLETION 3. WRITELOG 4. LOGMGR
		Avg. Disk Queue Length	Average of disk queues over time. If this number is	Confirm IO issues with disk sec/read and disk sec/write.

			consistently high, disk sec/read and disk sec/write will be high as well indicating IO bandwidth issues.	Waitstats correlation: 1. IO_COMPLETION 2. ASYNC_IO_COMPLETION 3. WRITELOG 4. LOGMGR
		Disk Sec/Read	Under normal circumstances, reads should take 4-8ms – confirm with hardware vendor for exact read time. Sustained queues will skew this number higher because disk sec/read factors in the effects of disk queues. High numbers mean your IO subsystem is not keeping up with requests Check individual drive performance if there are multiple drives. If it is a broad problem affecting all drives, the IO subsystem is not keeping up. More drives could be beneficial. If there is ONE very hot drive, then look at disk activity such as location of paging file, database, transaction log, and other	If disk sec/read > normal read time (ask vendor for normal read time) you can consider the following options: 1. **Resolve IO bottleneck by adding more drives; spreading IO across new drives** if possible e.g. move files such as database, transaction log, other application files that are being written to or read from. 2. **Check for memory pressure** – see memory component. 3. **Check for proper indexing** of SQL tables. Proper indexing can save IO. Check SQL query plans looking for scans and sorts, etc. Showplan identifies sorting steps. 4. **Run SQL Profiler** to identify TSQL statements doing scans. In Profiler, select the **scans** event class & **scan stopped** event. Go to the data column tab and add **object Id**. Run the trace. Save the profiler trace to a trace table, and then search for the **scans** event. Alternately, you can search for high duration, reads, and writes. Waitstats correlation: 1. IO_COMPLETION 2. ASYNC_IO_COMPLETION 3. WRITELOG 4. LOGMGR

			read/write activity.	
		Disk Sec/Write	Under normal circumstances, reads should take 4-8ms - confirm with hardware vendor. Sustained queues will skew this disk sec/write higher because this counter factors in the effects of disk queues. High numbers mean your IO subsystem is not keeping up with requests. In some SAN environments, writes can be as low as 1-2ms.	See disk sec/read. High performance (significant insert, update, and delete activity) requires the transaction log to be on a separate drive from the database. Waitstats correlation: 1. IO_COMPLETION 2. ASYNC_IO_COMPLETION 3. WRITELOG 4. LOGMGR
Memory / Cache	Memory	Page Faults/sec	This counter includes both hard faults (those that require disk access) and soft faults (where the faulted page is found elsewhere in physical memory.) Most processors can handle large numbers of soft faults without significant consequence. However, hard faults, which require disk access, can cause significant delays. See the **disk** component for more information.	Check for memory pressure (see SQL Server buffer manager), lo data page hit rates, & memory grants pending.

		Pages/sec	Number of pages read from or written to disk to resolve hard page faults. These are hard faults that require physical IO to fetch the page.	Compare with Page Faults/sec. Check for memory pressure (see SQL Server buffer manager), low data page hit rates, & memory grants pending.
CPU	Processor	%User Time	SQL Server runs in User mode. Privileged mode, is designed for operating system components and allows direct access to hardware and all memory.	Make sure **% user time** > 70%. Check task manager (taskmgr.exe) to see how much CPU sqlserver.exe is getting. If user time < 70%, check on **%Processor Time** & **% Privileged** activity.
		% Privileged Time	The operating system switches application threads to privileged mode to access operating system services	Should be < 20%. Check task manager (taskmgr.exe) to see how much CPU sqlserver.exe is getting. If **%privileged time** > 20%, check on **%Processor Time** & **% User Time**.
		%Processor Time	% of time CPU is executing over sample interval.	Common uses of CPU resources: 1. **Compilation and re-compilation** use CPU resources. **Plan re-use and parameterization** minimizes CPU consumption due to compilation. For more details on compilation, recompilation, parameterization and plan re-use, see http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsql2k/html/sql_queryrecompilation.asp --Plan re-use is where usecounts are > 1

				select dbid, objid, cacheobjtype, objtype, usecounts, sql *from master..syscacheobjects* *order by dbid,objid, cacheobjtype,objtype,usecounts,sql* Correlate to PERFMON counters: 1. System: Processor Queue length 2. SQL Statistics: Compilations/sec 3. SQL Statistics: re-Compilations/sec 4. SQL Statistics: Requests/sec If both of the following are true, you are cpu bound: 1. Proc time > 85% on average 2. Context switches (see **system** object) > 20K / sec light weight pooling can provide a 15% boost. Lightweight pooling (also known as fiber mode) divides a thread into 10 fibers. Overhead per fiber is less than that of individual threads.
		%Idle Time	% of time CPU is idle over sample interval	
		Interrupts/sec	Interrupts/sec is the average rate, in incidents per second, at which the processor received and serviced hardware interrupts.	Correlate with other perfmon counters such as IO, Network.
Thread	Process	Page Faults	This counter includes both hard faults (those that require disk access) and soft faults (where the faulted page is found	Check for memory pressure (see SQL Server buffer manager), low data page hit rates, & memory grants pending, page life expectancy.

			elsewhere in physical memory.) Most processors can handle large numbers of soft faults without significant consequence. However, hard faults, which require disk access, can cause significant delays. See the **disk** component for more information.	
System	Usage	Processor Queue Length		Number of threads waiting to be scheduled for CPU time. Some common uses of CPU resources that may be avoidable: 1. Unnecessary compilation and recompilation. Parameterization and plan re-use would reduce CPU consumption. See http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsql2k/html/sql_queryrecompilation.asp 2. memory pressure 3. lack of proper indexing
		Context Switches/sec		
SQL Server	Access Method	Forwarded Records/sec	Number of records fetched through forwarded record pointers. Tables with NO clustered index. If you start out with a	Look at code to determine where the short row is inserted followed by an update. Can be avoided by: 1. Using Default values (so that an update will not result in a longer row that is the root cause of forwarded records).

			short row, and update the row creating a wider row, the row may no longer fit on the data page. A pointer will be put in its place and the row will be forwarded to another page.	2. Using Char instead of varchar (fixes length so that an update will not result in a longer row
		Full Scan/sec	Entire table or index is scanned. Scans can cause excessive IO if an index would be beneficial.	SQL Profiler can be used to identify which TSQL statements do scan. Select the **scans** event class & events **scan:started** and **scan:completed.** Include the **object Id** data column. Save the profiler trace to a trace table, and then search for the **scans** event. The **scan:completed** event will provide associated IO so you can also search for high reads, writes, and duration.
		Index Searches/sec	Number of index searches. Index searches are used to start range scans, single index record fetches, and to reposition within an index.	Compare to Full Scan/sec. You want to see high values for index searches.
		Page Splits/sec	Number of page splits occurring as the result of index pages overflowing. Normally associated with leaf pages of clustered indexes and non-clustered indexes.	Page splits are extra IO overhead that results from random inserts. When there is no room on a data page, and the row must be inserted on the page (due to index order), SQL will split the page moving half the rows to a new page, and then insert the new row. Correlate to Disk: page sec/write. If this is very high, you may

				reorg the index(es) on the table(s) causing the page splits, to reduce page splits temporarily. Fillfactor will leave a certain amount of space available for inserts.
	Memory Manager	Granted Workspace Memory	Total amount of memory granted to executing processes. This memory is used for hash, sort and create index operations.	You can confirm memory pressure by checking the *Granted Workspace Memory (KB)* counter that tells you how much memory has currently been granted to running queries. If there is memory pressure due to workspace memory, this value should be at least 25% of the virtual memory available to SQL Server. If the memory pressure is severe, the server might even return errors such as 701 or 8645. If this is the case, this might be a good reason to consider using SS64. 1. No Memory pressure: When Granted Workspace memory <25% of virtual memory (up to 2GB or 3GB with /3GB) 2. Memory pressure: When Granted Workspace memory >25% of virtual memory (2GB or 3GB with /3GB)
		Memory Grants Pending	Current number of processes waiting for a workspace memory grant. Memory resources are required for each user request. If sufficient memory is not available, the user will wait until there is enough memory for the query to run.	If you see a long queue of Memory Grants Pending as compared to Outstanding grants, there is likely memory pressure due to query workspace memory. You can confirm this by checking the *Granted Workspace Memory counter.* Compare with Memory grants outstanding. If grants pending increases, you can do the following: 1. add more memory to SQL Server 2. add more physical memory to the box. 3. check for memory pressure – see & correct indexing if you experience “out of memory” conditions. Correlate to Waittype

				1. RESOURCE_SEMAPHORE
		Memory Grants Outstanding	Current number of processes that have successfully acquired a workspace memory grant	If you see a long queue of Memory Grants Pending grants as compared to Outstanding grants, there is likely memory pressure due to query workspace memory. You can confirm this by checking the *Granted Workspace Memory (KB)* counter that tells you how much memory has currently been granted to running queries. If there is memory pressure due to workspace memory, this value should be at least 25% of the virtual memory available to SQL Server. If the memory pressure is severe, the server might even return errors such as 701 or 8645. If this is the case, this might be a good reason to consider using SQL2000 64-bit.
		Target Server Memory	Total amount of **dynamic** memory SQL Server is willing to consume	If *Total Server Memory* is well below *Target Server Memory* at steady state, then it tells you that the server is not experiencing memory pressure.
		Total Server Memory	Total amount of **dynamic** memory SQL Server is currently consuming.	If *Total Server Memory* is well below *Target Server Memory* at steady state, then it tells you that the server is not experiencing memory pressure.
	Buffer Manager	Buffer cache hit ratio	Percentage of time the pages requested are already in cache	Check for memory pressure. See Checkpoint pages/sec, Lazywrites/sec and Page life expectancy.
		Checkpoint pages/sec	Pages written to disk during the checkpoint process, freeing up SQL cache	Memory pressure is indicated if this counter is high along with *high lazy writes/sec* and *low page life expectancy* (<300 seconds)
		Lazy writes/sec	Pages written to disk by the lazywriter, freeing up SQL cache	Memory pressure is indicated if this counter is high along with *high lazy writes/sec* and *low page life expectancy* (<300 seconds)

		Page life expectancy	Time in seconds the data pages, on average, stay in SQL cache. Low page life < 300 may indicate (1) SQL cache is cold, (2) memory problems or (3) missing indexes. Correlate to Lazywrites/sec and Checkpoint pages/sec	Memory pressure is indicated if this counter is low (<300) along with *high* *lazy writes/sec* and *checkpoint pages/sec*. Check for missing indexes and bad query plans (scans in profiler) Check for high page faults/sec.
		Readahead pages/sec	If memory shortages, cold cache, or low hit rates, SQL may use worker threads to readahead (bring in pages ahead of time) to raise hit rates. By itself readahead is not a problem unless users are flushing each other's pages consistently.	Correlate to counters for SQL buffer mgr: buffer cache hit ratio, page life expectancy, lazywrites, and checkpoint pages for memory pressure. Check for proper indexing and bad query plans (scans in profiler)
		Procedure Cache Pages	Number of pages used to store compiled queries.	*SQL Server:Buffer Manager:Procedure Cache Pages* captures the total number of pages in the plan cache. If this number is a significant fraction (typically, greater than 25 percent) of the total number of pages in the buffer pool, the application is plan cache intensive. However, this by itself is not sufficient to consider a move to SQL2000 64-bit. If the plan cache is large because it is full of plans that are seldom re-used, then moving to SS64 will not yield any benefits (and in fact might make matters worse due to the larger size of plan cache as described previously). To determine what kinds of plan are in the plan cache look at the contents of the virtual table *master..syscacheobjects*; to see whether these plans are being re-used, look at the *usecounts* column of this virtual table. If the plan cache is full of plans that are being re-used and yet there is memory

				pressure, this indicates the application would benefit from more virtual memory and hence SS64 might be a good option to consider.
	Cache Manager	Cache Hit Ratio	Percentage of time the procedure plan pages are already in cache e.g. procedure cache hits. I.e. how often a compiled procedure is found in the procedure cache (thus avoiding the need to recompile).	Check for memory pressure. See Checkpoint pages/sec, Lazywrites/sec and Page life expectancy. See also Memory Manager object. See SQL Profiler: Stored Procedure: CacheHit, CacheMiss, and CacheInsert to see what stored procedure query plans are already in cache (Hit), vs. those not in cache (Miss,Insert) Check for appropriate plan re-use in the usecounts column of master..syscacheobjects. It is often desirable for query plans to be re-used for similar SQL although not always. *Select cacheobjtype, objtype, dbid, sql, usecounts* *From master..syscacheobjects* See SQL Statistics: Compilations/sec for discussion of plan reuse. If there is memory pressure, plans will be discarded to make room for other data and/or procedure plans.
	Databases	Log Flush Wait Time	Waiting for transaction log writes (ms)	See disk perf counters Check transaction log file ::fn_virtualfilestats(dbid, file#) for IOStallMS (waits in ms)
		Log Flush Waits/sec	Tranlog writes per second	See disk perf counters, ::fn_virtualfilestats for IOStallMS.

		Log Growths	Windows will automatically grow transaction log to accommodate insert, update, and delete activity.	In general, growths of the transaction log will temporarily freeze writes to the transaction log while Windows grows the transaction log file. Check to see that the growth increment is large enough. If not, performance will suffer as log growths will occur more often.
		Transactions /sec	SQL Server transactions per second	
	General Statistics	Logins/sec	Number of logins per second	User connections
		Logout/sec	Number of logouts per second	
		User connections	Number of user connections	
	Latches	Average Latch Wait Time(ms)	Latches are short term light weight synchronization object. Latches are not held for the duration of a transaction. Typical latching operations during row transfers to memory, controlling modifications to row offset table, etc.	If high, check PERFMON DISK and MEMORY objects for 1. IO bottlenecks 2. memory pressure Normally reduced with more memory and/or IO capacity
		Latch Waits/sec	See above	
		Total Latch Wait Time(ms)	Short term light weight synchronization object. Latches are not held for the duration of a transaction.	If high, check PERFMON DISK and MEMORY objects for 1. IO bottlenecks 2. memory pressure

			Typical latching operations during row transfers to memory, controlling modifications to row offset table, etc.	Normally reduced with more memory and/or IO capacity
	Locks	Average Wait Time(ms)	Transactions should be as short as possible to limit the blocking of other users.	Hint: check for memory pressure, which causes more physical IO, thus prolonging the duration of transactions and locks.
		Lock Wait Time(ms)	Transactions should be as short as possible to limit the blocking of other users.	Hint: check for memory pressure, which causes more physical IO, thus prolonging the duration of transactions and locks
		Lock Waits/sec	Transactions should be as short as possible to limit the blocking of other users.	Hint: check for memory pressure, which causes more physical IO, thus prolonging the duration of transactions and locks

	SQL Statistics	Compilations/sec	Includes initial compile and subsequent re-compiles. Compilation and re-compilation are CPU intensive operations. Unnecessary compilation can sometimes be avoided with query plan re-use. Query plan re-use can be seen in the *usecounts* column of *syscacheobjects* as follows *Select cacheobjtype, objtype, dbid, sql, usecounts* *From master..syscacheobjects* Parameterization is important for plan re-use. In addition, some types of re-compilation can be avoided. See the SQL Server 2000 recompilation paper for more info: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsql2k/html/sql_queryrecompilation.asp To get initial compilations ONLY, you must subtract recompilations/sec from compilations/sec. Compare to batch requests/sec to see extent of compilation.
		Recompilations/sec	Only contains re-compiles. SQL Profiler can provide information on what procs are recompiling, what statement, and the reason for recompilation. In Profiler, select the **stored procedure** event class & **SP:recompilation** event, and include the data column **eventsubclass**. Review the trace searching for eventsubclass values 1 through 6. The statements preceding

			caused the recompilation. For more details on recompilation, see http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsql2k/html/sql_queryrecompilation.asp
		Batch Requests/sec	Total batch requests should be compared with compilations/sec
		Auto-Param Attempts/sec	Auto-param attempts should be compared to failed auto-params/sec. Proper parameterization is important for plan re-use. In some cases, Sp_executeSQL could be used with adhoc SQL. For more details on recompilation, see http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsql2k/html/sql_queryrecompilation.asp
		Failed Auto-Params/sec	Auto-param attempts should be compared to failed auto-params/sec. Proper parameterization is important for plan re-use. In some cases, Sp_executeSQL could be used with adhoc SQL. For more details on recompilation, see http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsql2k/html/sql_queryrecompilation.asp

Interesting PERFMON Ratios & comparisons

Some counters in PERFMON have to be compared to other counters for proper perspective. While the following ratios and comparisons are not exhaustive, they nonetheless will point you in the right direction.

1) **Batch requests/sec vs. SQL Compilations/sec.** The worst case is when compilations are very high compared with batch requests. Possible memory pressure in which query plans are discarded quickly to make room for other activity. Another possibility is lack of parameterization which is important for plan re-use. Parameterization is where variables are used instead of literal values. In some cases sp_executeSQL can be beneficial. *Perfmon counters are SQLServer:SQL Statistcs:Batch Requests/sec and SQLServer:SQL Statistics:SQL Compilations/sec.*

2) **SQL Compilations/sec vs. SQL Re-compilations/sec.** SQL Compilations/sec include the initial compile of the stored procedure while SQL Re-compilations/sec only includes re-compiles (excludes initial compile). If initial compiles are low (SQL Compilations – SQL Recompilations) compared to SQL recompilations, then there is a probable recompilation problem. *Perfmon counters are SQLServer:SQL Statistcs: SQL Compilations/sec and SQLServer:SQL Statistics:SQL Re-Compilations/sec.*

3) **Kernel CPU vs User CPU.** If (Kernel CPU/ User CPU) > .25, may indicate a network, disk driver, or hardware issue. Network and Disk IO is serviced in kernel mode. SQL is serviced in user mode. Look at Task Manager. *Perfmon counters are Processor:%Processor Time, Processor:%User time, Processor:%Interrupt time.*

4) **Disk Queue Length vs Disk sec/Transfer.** As disk queue length increases, so does disk sec/transfer. Perfmon counters are PhysicalDisk:Avg Disk Queue Length and PhysicalDisk:Avg Disk sec/Transfer.

5) **Page life expectancy, checkpoint pages/sec, lazywrites/sec comparison.** Memory pressure is indicated with low page life expectancy, and high checkpoint pages and lazywrites/sec. Memory pressure, which adversely affects performance, can be lessened one or more of the following
   a) Adding more memory to the box
   b) Increasing SQL memory
   c) Avoiding table and index scans with proper indexing

6) **Signal waits, UMSStats runnable queue comparison.** Basics of execution model (simplified)
   a) If a spid is running then needs unavailable resource, it moves to resource wait list at time T0
   b) a signal indicates resource available, spid moves to runnable queue at time T1
   c) spid awaits running status until T2 as cpu works its way through runnable queue in order of arrival

d) resource wait time is the actual time waiting for the resource to be available, T1-T0
e) signal wait time is the time it takes from the point the resource is available (T1) to the point in which the process is running again at T2. Thus, signal waits are T2-T1
f) Key questions: Are Resource and Signal time significant?
   a. Highest waits indicate the bottleneck you need to solve for scalability
   b. Generally if you have LOW% SIGNAL WAITS, the CPU is handling the workload e.g. spids spend move through runnable queue quickly
   c. HIGH % SIGNAL WAITS indicates CPU can't keep up, significant time for spids to move up the runnable queue to reach running status

7) **Network: Current bandwidth, bytes total/sec, packets/sec.** Network bandwidth issues should be corroborated with bytes total/sec. [Network interface: bytes total/sec] / [Network interface: Current Bandwidth] > .6, possible network bottleneck.

8) **Page Faults/sec vs Pages/sec.** Page faults include both hard faults (those that require disk access) and soft faults (where the faulted page is found elsewhere in physical memory.) Most processors can handle large numbers of soft faults without significant consequence. However, hard faults, which require disk access, can cause significant delays. Pages/sec represents the number of hard page faults that require physical IO to bring the pages into memory.

Memory Issues

This is extracted from SQL 32-bit vs. SQL 64-bit comparison by Prakash Sundaresan. The SQL Server relational database system uses memory for a number of different purposes internally. For a more complete discussion of the memory uses in SQL Server, see http://www.winnetmag.com/Article/ArticleID/43419/43419.html. To summarize briefly here, the main uses are:

1. Database page cache – used to cache database (table / index) pages
2. Query Workspace memory – used by memory intensive query operations such as Hash and Sort.
3. Plan cache – used to cache query plans so they can be re-used
4. Other – locks, connection memory, thread stacks, memory for utilities such as backup/restore etc.
5. Memory used by other components linked into the SQL Server process such as XPs, OLE-DB providers etc. The memory is commonly referred to as the MemToLeave memory area because SQL Server refrains from allocating this memory so that these other components linked into the process can do so.

Comparison of 32-bit memory architecture vs. 64-bit flat memory

Of these, only the first use – database page cache – is able to make use of AWE memory on 32-bit systems. The rest of the uses require virtual memory, hence they are limited to 2GB (or 3GB with the /3GB switch in boot.ini) on 32-bit systems. If an application stresses one or more of these other uses of memory in SQL Server to a point beyond what can be handled by the 32-bit virtual memory limits, you might consider the SS64 option. To determine if this is the case, see below for some steps you might want to consider:

1. Overall Server Memory: Look at counters under *SQL Server:Memory Manager*. If *Total Server Memory* is well below *Target Server Memory* at steady state, then it tells you that the server is not experiencing memory pressure. In this case, you likely have no performance-related reason to consider SS64. Otherwise, you need to look further into the cause of memory pressure by following the steps below. Of course, if you do have memory pressure, you might already be using additional memory in SQL Server by enabling AWE. In this case, SQL Server allocates Max Server Memory at startup and therefore *Total Server Memory* does not change dynamically. In this case as well, follow the steps below to tell if there continues to be memory pressure.
2. Query Workspace Memory: Look at counters under *SQL Server:Memory Manager*. Look at *Memory Grants Outstanding* and *Memory Grants Pending*. If you see a long queue of Pending grants as compared to Outstanding grants, there is likely memory pressure due to query workspace memory. You can confirm this by checking the *Granted Workspace Memory (KB)* counter that tells you how much memory has currently been granted to running queries. If there is memory pressure due to workspace memory, this value should be at least 25% of the virtual memory available to SQL Server. If the memory pressure is severe, the server might even return errors such as 701 or 8645. If this is the case, this might be a good reason to consider using SS64.
3. Plan Cache: The counter *SQL Server:Buffer Manager:Procedure Cache Pages* captures the total number of pages in the plan cache. If this number is a significant fraction (typically, greater than 25 percent) of the total number of pages in the buffer pool, the application is plan cache intensive. However, this by itself is not sufficient to consider a move to SS64. If the plan cache is large because it is full of plans that are seldom re-used, then moving to SS64 will not yield any benefits (and in fact might make matters worse due to the larger size of plan cache as described previously). To determine what kinds of plan are in the plan cache look at the contents of the virtual table *master..syscacheobjects*; to see whether these plans are being re-used, look at the *usecounts* column of this virtual table. If the plan cache is full of plans that are being re-used and yet there is memory pressure, this indicates the application would benefit from more virtual memory and hence SS64 might be a good option to consider.
4. Memory pressure on MemToLeave – e.g. due to XPs or OLE-DB providers Typically, if you have pressure in the MemToLeave area, you might see errors such as 7399, 17802, or 17803. In such cases, you might have already considered altering the –g startup parameter for SQL Server to increase the MemToLeave value. This in turn might translate to some of the other kinds of memory pressure described here.
5. High CPU cost of AWE memory: In some cases, even if your memory use consists primarily of database page cache, the CPU cost of mapping and un-mapping database pages using AWE might become too expensive, as evidenced by high kernel CPU time. This is especially true when the number of CPUs in your system is 8 or more and/or when the size of physical memory exceeds 32GB on your 32-bit system. This is another point at which you might consider use of a 64-bit system.

64-bit flat memory vs. higher 32-bit clock speeds

As seen above, there are cases where memory pressure is genuine and SS64 might be an attractive option in those cases. However, even in these cases the choice of a 64-bit system over 32-bit systems is not straight-forward. Clock speeds on Itanium-based 64-bit systems are much lower than on Xeon-based 32-bit systems. The Itanium's ability to execute multiple instructions concurrently does compensate for this to some extent. However, if your application is CPU-heavy, you might find you need as many or more processors on a 64-bit system as on the comparable 32-bit system to handle the same workload. It is always recommended that the relative performance of the two choices be verified through a prototype or proof of concept to verify that the 64-bit platform would be a good investment.

Application Design issues

There are application design considerations resulting from the Waits and Queues methodology. The following table describes some of the application design implications.

Observation	Application issue	Possible remedies
High IO waits	1. Database design 2. Memory pressure	1. Bad query plans resulting from improper indexing. 2. Add correct indexes to minimize IO. 3. Add more memory
High CPU utilization	1. Memory pressure 2. Plan re-use 3. Parameterization	1. Check for correct plan re-use, parameterization, re-compilation , see http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsql2k/html/sql_queryrecompilation.asp
High blocking / locking	1. Transaction management	2. Redo transaction management 3. use correct transaction isolation levels

Conclusion: Waits & Queues Analysis

There are two complimentary sources of performance information for SQL Server. Wait types are an invaluable clue in analyzing overall system performance from an application point of view. Wait types provide a view of system performance from a SQL thread standpoint while PERFMON provides a view of system performance from a resource standpoint.

Wait statistics should be corroborated or associated with resource counters in PERFMON. For example, a high SQL Server wait types signal the need for further PERFMON investigation of underlying resources such as processor, IO subsystem, network and so forth. Together, these associations or correlations of wait types to perf counters, and other related counter ratios provide a broad picture of application performance.

In come cases, the experienced performance expert must look beyond the symptom to find the root problem. While not exhaustive, the correlated performance info, possible conclusions and actions, and interesting ratios and comparisons sections will shed light on actual root problems, given the symptoms. The waits and queues methodology presented here, will identify system bottlenecks and propose further corroboration and conclusions, where appropriate.

In sum, the performance methodology of waits and queues draws on the available performance information comprised of Waitstats, PERFMON counters, and correlated information, to provide a broad profile of application performance. It is an invaluable tool in ferreting out and fixing performance problems.

Appendix A: References

1. Microsoft SQL Server 2000 Recompilation http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsql2k/html/sql_queryrecompilation.asp
2. Microsoft SQL Server 2000 Performance Tuning by Whalen, Garcia, DeLuca & Thompson
3. Inside SQL Server 2000 by Kalen Delaney
4. Understanding and resolving SQL 2000 Blocking http://support.microsoft.com/default.aspx?scid=kb;en-us;224453
5. How to monitor SQL Server 2000 blocking http://support.microsoft.com/default.aspx?scid=kb;en-us;271509

Appendix B: IO

Quick overview of IO subsystems

IO bandwidth depends on a number of factors such as drive types, configuration (e.g. RAID type) and workload characteristics. Besides space capacity, each drive has vendor published random and sequential IO rates per drive. RAID provides fault tolerance and additional IO capacity by combining multiple physical drives into a single logical drive. The type of RAID implemented has an effect on the extent of fault tolerance (mirror is an identical copy, vs RAID5 allows the loss of 1 drive), read and write performance, and space capacity (a mirror takes twice as much space).

Workload characteristics dictate the mix of sequential vs. random IO. Sequential IO is common with large queries such as reports while random IO is normally associated with OLTP workloads.

File & Table level IO

SQL Server provides file level IO using the function ::fn_virtualfilestats. In addition, SQL Server can provide IOs per table or index **if you are set up properly**.

Here are the details. IO is reported at the file level. One or more files belong to a file group. Objects are placed on a File Group. If your file group has multiple files, IO for the File Group is spread across the files. If you place a *single* table on a file group, you can easily get IO for the table using:

```
Select * from ::fn_virtualfilestats(dbid,file#)
```

Do not drop tables, just drop the indexes. Then re-create the index with an ON FILEGROUP clause. If the index is a clustered index, the table and clustered index are treated as a single entity, and will move to the new FILEGROUP. If the index is non-clustered, only the nonclustered index is created on the new FILEGROUP. To get all IO for a table, the table and all indexes should be moved to the filegroup.