

Top 10 Server Configuration Tips for a New ASE 15 Installation

BY JEFFREY GARBUS, SOARING EAGLE CONSULTING

EXECUTIVE OVERVIEW

Sybase Adaptive Server® Enterprise 15 offers a broad scope of tuning options at the server level. Many are simple decisions: configure them. Others require a small amount of research. This article focuses specifically on configuring ASE performance at the server level. I've created a list of things I look for, most of which are automatic and underutilized. A few are things you can change after a modicum of research into `sp_configure` output. So, to that end, I tip my hat to David Letterman and offer you... my Top 10 configuration changes to make on an ASE database.

INTRODUCTION

In almost 20 years of tuning Sybase Adaptive Server Enterprise (including the time it wasn't yet called ASE), I've found a variety of similarities in the types of missed opportunities in server configuration. This has gotten worse over time, as with the release of each new server, new configuration options become available.

This article focuses specifically on configuring performance at the server level. For information on identifying queries to tune, check out the article on finding poorly performing queries in ASE 15.

To this end, I've created a list of things I look for, most of which are automatic and underutilized. A few are things you can change after a modicum of research into `sp_configure` output. So, to that end, I tip my hat to David Letterman and offer you... my Top 10 configuration changes to make on an ASE database.

Most of these options are “no-brainers” — meaning that I make them automatically. Regardless, you should always track all of your changes, and measure the results of the change.

My only caveat: Tuning a database management system is what the word implies; a balancing act between resources. Tuning decisions are not made in isolation; they are made with all of the other system resources in mind. As a result, for many of these recommended configuration changes, I am going to advise you to look at specific external resources, and/or to look at the results of `sp_sysmon` (ASE's performance monitoring stored procedure) to determine “how much.”

So without further ado, and in no particular order:

MULTIPLE TEMPDBS

The tempdb database is used for intermediate results by users, applications, and occasionally by the server for complex queries (though with ASE 15 this has been reduced). As a result, it can be a bottleneck both from a physical io standpoint or (in the case of lots of user-created temp tables) potentially a system table bottleneck.

I teach my “Performance and Tuning” students to always treat tempdb as a bottleneck. Historically, customers have even gone to the extreme end of replacing standard disk drives with Solid State Disk (i.e. memory with a battery & disk backup) in order to alleviate tempdb bottlenecks. Others are going with expensive SAN alternatives.

Solid State Disk, while not cheap, usually solved the problem.

Today, there are other alternatives. If tempdb is creating a bottleneck, or especially *if you anticipate that it may become a bottleneck*, you can create multiple tempdbs.

This is a very flexible process. You can create multiple tempdbs for all users to share in a round robin fashion, or separate tempdbs set aside for specific users, applications, or business processes.

My own, default preference is to create multiple tempdbs for users, set up to be used on a round-robin basis, and a separate tempdb for the sa's use. This gives you, the sa, the ability to run system stored procedures (like `sp_who` & `sp_lock`) to find out why all of the other tempdbs have filled up.

To check on your current tempdb configuration, use the `sp_tempdb` procedure. (And for more information, look up `sp_tempdb` in the system administration guide.

Usage:

– To display current tempdb setup, use this:

```
exec sp_tempdb 'show'
```

I've seen recommendations, in print, that the number of user tempdbs should correspond with the number of processors on the box for optimal performance. This may be overkill. If you have a 20-processor box, going from 1 tempdb to 5 or 6 tempdbs may be just the relief you need; *not everybody* uses tempdb. The flip side is, if everybody *is* using tempdb, the number of database engines is probably a top-end limitation on the effectiveness of adding additional tempdbs.

Tip: Make sure that each of the tempdbs is still large enough to handle a significantly sized temporary work set. If you have a batch process that needs a huge tempdb, you don't need all the tempdbs to be huge, you just need to make bind that process to a specific tempdb that is big enough.

PROPERLY CONFIGURE SUFFICIENT DATA CACHE

How much data cache is enough? On a Sybase ASE server, when you're looking at your cache hit rate, you want to see numbers in the high 90s, preferably numbers in the high 99s (i.e. 99.7, 99.8). This means that most of your page requests are being managed out of cache. If you are getting numbers below this, you can use more memory.

That said, there are a few things you can do with data cache to use memory more efficiently.

2-page buffer pool

Add a 2-page (4k on 2k-page-sized servers) buffer pool to any data cache in which data is modified.

This is especially important in default data cache, as this will help the server come up (run recovery) faster. This pool doesn't have to be big, it simply needs to be big enough to contain current transactions. 10 or 20 meg may be big enough on even busy servers, 50 meg or 100 meg may be overkill; check your cache usage in `sp_sysmon` output, see if dirty data pages are getting past the wash marker in the 2-page pool (if it is, you can allocate a bigger buffer pool).

8-page pool

Add an 8-page (16k on 2k-page-sized servers) buffer pool to any cache that has potential io which is not random-index-read. This includes almost any cache, especially the default data cache.

This will have a major positive impact on activities like table scans and index scans. This size of this pool in proportion to the 2-page pool is more scientific method than math — start with 20-25% of available cache, and then watch the "large io % denied" in `sp_sysmon`, increasing the percentage until that number is pretty low, without starving the 2-page cache.

cache_partitions

In a multiprocessor environment (almost everybody these days), the server keeps the different processors from stepping on one another (simultaneously modifying or obtaining locks on objects) by placing a very fast lock called a latch, also called a spinlock, on the object. This spinlock is an extremely fast lock. Sometimes, though, something that is protected by spinlocks gets so busy that the spinlock access becomes a bottleneck.

In the past, when we had high spinlock ratios in our `sp_sysmon` output, we would add another cache (a log cache, a tempdb cache, etc.). Frequently this would lead to a significant over-segmentation of data cache (i.e. there would be so many small caches that the default data cache, or another large cache, when needed, had a poor hit rate).

Today, when we have a high spinlock ratio (I tend to treat 5% as a problem; 1 in 20 queries doesn't get into cache when it needs to), we can instead increase the number of partitions in cache.

This is easy to do:

```
sp_cacheconfig "[cachename [,cache_size [P|K|M|G]"
[ ,logonly | mixed ] [,strict | relaxed ] ]
                [, "cache_partition = [1|2|4|8|16|32|64]" ]
[ , "instance instance_name"]
```

We're looking specifically at the cache_partition option. I titrate this up when I have high spinlock ratios, it keeps me from having to segment the default data cache into multiple caches.

Note that we can set this to default at the server level for any additional caches using the "Global cache partition number" configuration option.

OAM trips/Index trips

Data cache is normal a FIFO queue: first in, first out. As pages work their way through cache, if they are touched, they move to the front of the queue, if they are not, they may fall off of the end of the page chain.

If you have queries that will occasionally fill cache with a single table, or if you have a relatively small cache in relation to your database size(s), you can use the configuration options "Number of index trips" and "number of OAM trips" to increase the number of passes an index or Object Allocation Map page will take, (i.e. not the default of 1) through cache before aging out. This can save a lot of physical io when other processes are accessing data via the indexes, to find that the index pages haven't been inadvertently aged out. You can bump this up in increments of 10.

Relaxed LRU

When you have a small enough amount of data in proportion to a cache size that everything fits into cache, sp_sysmon will recommend that you assign the "relaxed lru" property to the cache.

You have to make your own decision on this. If a named cache contains all of the objects that have been bound to the cache, you can reduce pointer movement in the cache by specifying relaxed LRU in the cache.

This reduces the amount of work that is done each time a page is accessed.

DON'T OVER CONFIGURE PROCEDURE CACHE

... but do configure sufficient procedure cache.

In general, procedure cache should be big enough to contain the largest stored procedure on the server for each process that is running, plus a margin for error, plus statement cache size.

The P&T guide for ASE 15 offers this formula:

$$(\text{Max\# of concurrent users}) * (4 + \text{Size of the largest plan}) * 1.25$$

If your biggest stored procedure is a 1 meg stored procedure and you have 1000 users, this means about a gig of procedure cache, plus a bit extra for overhead. If your biggest stored procedure is a 60K stored procedure and you have 500 users, this is much smaller; 30 meg plus overhead.

Once you're there, you also need to add in space for histograms, sort buffers (aux buffers), subqueries, and other things. Sybase recommends, for ASE 15, that you make your procedure cache at least double the size of your 12.5 procedure cache.

In the early days of SQL Server memory management, procedure cache defaulted to 20% of available memory, and to this day, many shops with 10 gig of memory will set aside 8 gig for data cache and 2 gig for procedure cache.

Even when what they need is 50 meg.

Give the extra procedure cache back to data cache.

Except...

STATEMENT CACHE

A new ASE feature enables the server to cache individual SQL statement plans, in addition to stored procedures. If you want to use this feature, and you should want to use this feature, you should add space for statement cache.

Note that the statement cache uses spaces in procedure cache, so your procedure cache needs to be big enough to accommodate both your stored procedures and the statement cache requirements.

I generally start with about 200 meg of statement cache, and check the effects in `sp_sysmon` output.

Turn on Literal Autoparameterization

On a lateral topic to statement cache, “literal autoparameterization” allows the server to treat similar queries as the same from a statement cache perspective.

So, if you take the following 2 queries:

```
Select * from authors where au_id = '111-11-1111'
```

And

```
Select * from authors where au_id = '222-22-2222'
```

Should the plan be the same?

With literal autoparam on, they are the same from a plan perspective, otherwise they are different. This option can take a huge amount of strain off of the optimizer, especially at shops where a lot of queries are ad-hoc (as opposed to through stored procedures).

HISTOGRAMS

Have you ever had a query that works great with some data, horribly for others? In other words,

```
Select * from MYTABLE where The_date between '1/1/91' and '1/1/92'
```

Is very fast, and

```
Select * from MYTABLE where The_date between '1/1/08' and '1/2/08'
```

Is very slow?

Chances are, the histograms that the server uses are either out of date or insufficient to the optimizer’s task of estimating rows returned.

It’s always important to make sure statistics are up to date, but sometimes that’s not enough. Sometimes, you need to increase the number of steps involved in the histograms. In the past, we’ve had to change the way we run the “update index statistics” statement, increasing the number of steps (values clause) and/or the amount of sampling. I’ve done this at literally dozens of shops.

Now, we can make these changes at a global level. How many histogram steps you add will depend upon what is working for you, but a good starting point is:

```
sp_configure 'number of histogram steps',50
```

```
sp_configure 'histogram tuning factor', 20 -- (the default in 15.0.2)
```

You may want to override these numbers for specific tables with large volumes of data and an erratic skew, but these are a good start. Note that 50 & 20 in combination give you 1000 steps, which are likely enough for most purposes. Along those lines, you may also want to adjust the sampling percentage. Sybase experimentation has shown that a sampling percentage of 10% is typically as good as a sampling percentage of 100%. I tend to act on the side of paranoia, and double that for good measure:

```
sp_configure 'sampling percent', 20
```

SEMANTIC PARTITIONING

Evaluate ASE’s newest trick for managing large tables: Semantic partitioning.

Semantic partitioning allows you to place subsets of data on different devices and or different sections of disk. If you are on a SAN, this may not seem like a huge advantage, but think of this also: You are able to perform maintenance on specific partitions.

For example, if you have a partition for data related to 2009, you can update statistics, run `dbcc checkstorage`, or run a `reorg` on *only the data that has changed* (presumably in 2009), leaving all the rest of it alone.

This may give you enough of a benefit, just in terms of reducing your maintenance window, to take the plunge. But, as you realize that local indexes are smaller than global indexes, you will find that there's a great performance benefit in using those indexes as well. (We're not going into a huge discussion here; that's a subject fit for a whole article all by itself!)

DSYNC VS. DIRECTIO

io is almost always a potential bottleneck.

In a server environment, the server operating system will try to buffer writes in order to speed up performance. The catch is, if the server suddenly powers down, the database thinks that io is consistent, but because of the buffering, and the issue of many different simultaneous page writes, inconsistencies may slip in.

In the past, Sybase always recommended (in a Unix environment) using raw partitions for data, and usually recommends a file system for tempdb (because tempdb is always recreated when the system comes back up anyway, we let the operating system improve performance). Recently, Sybase added the dsync option to the disk init syntax, which enables the use of file systems (specifically for SANs!), without having to have a raw partition. The dsync option forces the OS to sidestep the buffering.

For ASE 15, there is now a directio option, which is similar to but different from (and antithetical to!) dsync; in fact, you can only have one of the two enabled.

Experimentation has shown directio to vary from a little better to as much as 400% faster... and, Sybase recommends using it rather than dsync on any platform whose file system supports it

Options:

DSYNC DIRECTIO Recommendation

Off	Off	Only (and always) for tempdb
Off	On	Experiment to see which works best in your environment
On	Off	Experiment to see which works best in your environment
On	On	Not permitted

LIMITED PARALLEL PROCESSING

Enable parallel processing, but in a small way (to start).

Parallel processing can dramatically speed up query performance, index creation, sorts, scans, preventive maintenance, etc.

Don't go overboard. The more processes you enable for parallel processing, the higher the impact on your server. Turning parallel processing on does not improve server workload, instead it uses additional server resources to improve end-user experience. Enable parallel processing *in isolation* of other changes, so that you can measure the effect. Chances are, CPU usage will go up, and end users will call to ask you if things are going faster or if they're going crazy.

To start:

```
sp_configure "number of worker processes", 10
```

```
sp_configure "max parallel degree", {<= number of processors on your box}
```

```
sp_configure "max scan parallel degree", {<= number of processors on your box}
```

Make sure you can afford the additional CPU, and you can titrate the resources up or down depending on the resources on your box. If you are not currently partitioned, parallel processing will probably not work unless you also have "max scan parallel degree" set.

SCHEDULE A FULL PREVENTIVE MAINTENANCE REGIMEN

One of the more common tuning errors I encounter includes not setting up a consistent, periodic maintenance regimen. ANY of the following, not performed on a regular basis, can have a profound negative effect on performance. Make sure you include:

- Update statistics
 - Without this, the optimizer is making decisions based on potentially out-of-date data
- Reorg
 - If any of your data is locked using the newer locking schemes (i.e. datarows or datapages), you'll need to make sure rows are actually deleted, forwarded rows are put back on the correct pages, and pages are rebalanced. Otherwise, io performance to these tables will snowball in a bad way.

- Index rebuilds
 - ASE indexes are self-balancing and self-maintaining. Mostly. If you haven't done it already, you should set up a periodic job to drop & rebuild indexes which aren't set up for a reorg rebuild.
- DBCC checkstorage
 - It doesn't happen often (I've seen it only a handful of times in 20 years), but sometimes you get inconsistencies in your data. Running checkstorage identifies these inconsistencies, and on a good day allows you to correct this before it snowballs into a full-blown corruption.

EXPERIMENT WITH NEW OPTIMIZATION GOALS

Depending upon your environment – Online transaction Processing (OLTP), Decisions Support(DSS), or Mixed – the server can take a little more time to optimize to bring results back a lot faster, or use fewer possible join techniques and reduce time spent with the optimizer.

This is a new feature with ASE 15.

Syntax:

```
sp_configure "optimization goal", o, "allows_oltp"
```

This can also be changed at the session level (or query level!), and in fact this is the best place for experimentation. Some shops have very simple queries, and well tuned databases, and as a result don't see a big difference in sets of plans. Others have a great deal of ad hoc query which benefit from a few extra milliseconds at optimization time.

Note that if you are going to be testing `allows_dss`, you might also want to experiment with optimizer timeout limits, so that it doesn't get too carried away with the large number of optimization opportunities available for the monster queries:

Syntax:

```
set plan opttimeoutlimit
```

Experiment; decide what works best for you.

SUMMARY

Sybase Adaptive Server Enterprise 15 offers a broad scope of tuning options at the server level.

Many are simple decisions: configure them. Others require a small amount of research.

All are worth implementing.

Tuning is an ongoing exercise; it is striving for perfection, not necessarily attaining it. Make sure that you monitor / measure the effects of each of the above recommendations, and continue tuning accordingly.

Jeff Garbus has 20 years of expertise in architecture, tuning and administration of Sybase ASE, Oracle, and Microsoft SQL Server databases with an emphasis on assisting clients in migrating from existing systems to pilot and enterprise projects. He has co-authored 15 books and has published dozens of articles on the subject. Mr. Garbus is the CEO of Soaring Eagle Consulting, an organization that specializes in assisting businesses maximize database performance. www.soaringeagle.biz.