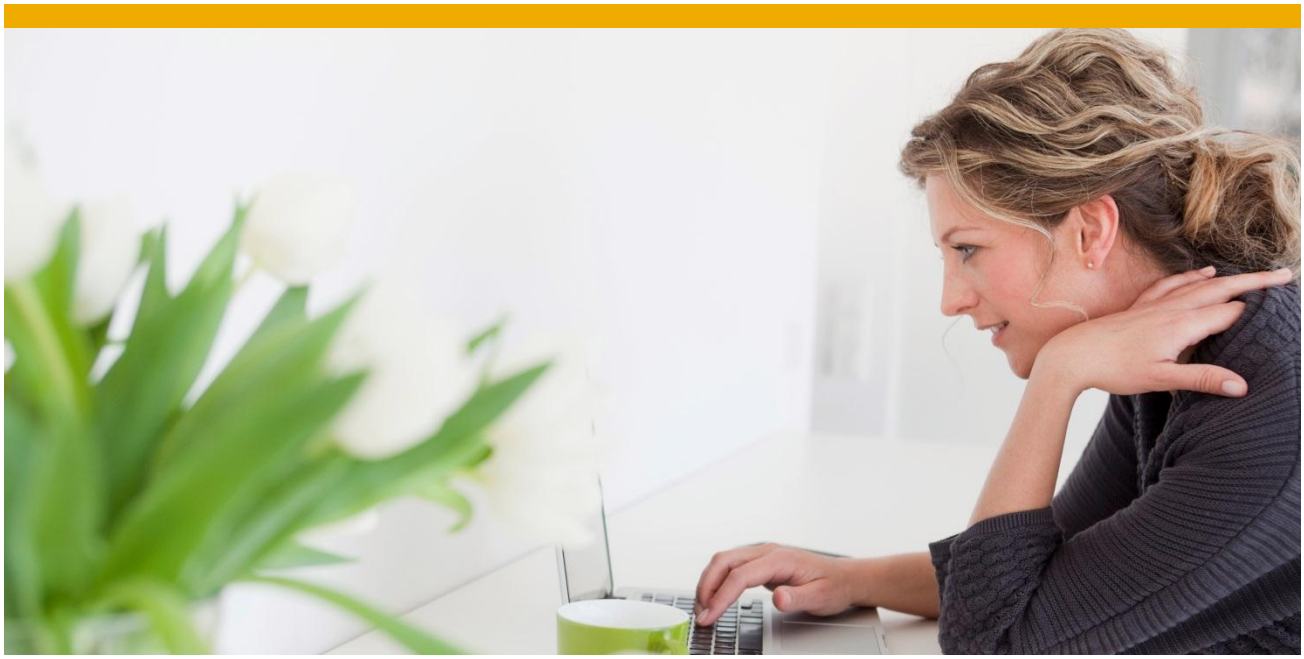


Optimizing SAP Sybase Adaptive Server Enterprise (ASE) for IBM AIX

An IBM and SAP Sybase white paper on optimizing SAP Sybase ASE 15 for IBM AIX 5L , AIX 6.1, and AIX 7.1 on IBM POWER5, POWER6 and POWER7 processor-based servers

Peter H. Barnett (IBM), Stefan Karlsson (SAP Sybase), Mark Kusma (SAP Sybase), and Dave Putz (SAP Sybase)

August 2013

Table of contents

Abstract

This is a paper from IBM and SAP Sybase to assist database administrators (DBAs) and UNIX system administrators working together to achieve the best possible performance from SAP Sybase Adaptive Server Enterprise (ASE) on IBM Power Systems servers running in the IBM AIX operating environment.

The focus of this white paper is on SAP Sybase ASE 15.0.2 through SAP Sybase ASE 15.7, running under AIX 5.3, AIX 6.1, and AIX 7.1 on the IBM POWER6 and IBM POWER7 and IBM POWER7+ processor-based servers. The recommendations of the original version of this white paper, published in March 2006, applied to IBM AIX 5L logical partitions (LPARs) with dedicated processors, memory, and devices. The 2009 revision of the paper also addressed all relevant aspects of IBM PowerVM including shared-processor LPARs (referred to as IBM Micro-Partitioning) and virtual I/O, features which are only available in the POWER 5, POWER6, and POWER7 processor-based servers with AIX 5.3 or later releases. The 2012 revision of the white paper extended coverage to POWER7, enhancements to virtual I/O, and AIX 7.1 were relevant to ASE. The current revision extends coverage to ASE 15.7 Electronic Software Distribution (ESD) 4.2. Recent field and lab experience with POWER7and POWER7+ processor-based servers is applied to new or revised best practice recommendations. The recommendations apply to ASE Cluster Edition for AIX except where noted.

Introduction

This paper is particularly relevant to those businesses that are implementing SAP Sybase ASE instances in one of the following environments:

- Implementing new SAP Sybase ASE instances on IBM® Power Systems™ with IBM AIX® 5.3, AIX 6.1, and AIX 7.1
- Upgrading from earlier IBM Power Systems and AIX installations
- Migrating to an IBM Power System server running AIX 5.3, AIX 6.1, or AIX 7.1 from another version of the UNIX® operating system

Note: In this paper, the operating system is referred to as AIX unless a feature or issue is specific to a particular version of AIX 5.3, AIX 6.1, or AIX 7.1. The intermediate release levels, called technology levels are abbreviated as TL where referenced. References to the IBM POWER7® architecture apply to the IBM POWER7+™ processor-based server line unless otherwise noted. ASE 15.0.1 to ASE 15.7 are referred to as ASE unless a feature or issue is specific to a certain level. References to ASE 15 “process mode” apply to ASE 15.7 process mode as well as to ASE 15.0.3 and ASE 15.5. References to ASE 12.5 and AIX 5.2 and AIX 5.3, which are out of support, are retained for the sake of clients who might have extended maintenance arrangements for special circumstances.

In new implementations, upgrades, or migrations, DBAs and system administrators can encounter features of the AIX operating system and the IBM Power Systems architecture including its management of memory and processor utilization, network I/O, and disk I/O operations that are unfamiliar. Administrators might also have questions about how best to implement SAP Sybase ASE in their environment. The high throughput and unique processor-dispatching mechanism of the IBM Power Systems architecture can also require revisions of older configurations.

The scope of this white paper is the interface between SAP Sybase ASE and the AIX operating system. SAP Sybase ASE manuals address general ASE tuning, such as systems tuning, physical-database

design, and query tuning. AIX and IBM Power Systems manuals and white papers address details about the server, the operating system, storage, and virtualization that are generic to database servers and are beyond the scope of this paper. To the extent that ASE system tuning is pertinent to the optimization of performance in an AIX or IBM Power® environment, it will be addressed. Likewise, to the extent that virtual memory and I/O management, processor options, and virtualization features of Power Systems are relevant to optimizing ASE on AIX/Power, they will be addressed as well.

The focus of this white paper is also on the aspect of database administration in which the DBA and UNIX system administrators interact most closely. Ideally, DBAs who read this paper will be competent in SAP Sybase ASE; UNIX system administrators will be competent in IBM Power Systems and AIX, and both will be reasonably acquainted with the other's specialty.

This white paper discusses some commands that require UNIX **root** privileges. Other commands require SAP Sybase ASE **sa_role** authority. Close communication and cooperation between the system administrator and the DBA is essential in every aspect of tuning, including the following tasks:

- Sizing and allocation of shared memory in the ASE engine
- Configuration of virtual memory in the operating system
- The relation of processor resources to online ASE engines
- Storage considerations, such as raw logical volumes for ASE devices and asynchronous I/O configuration

This paper contains sections on virtual memory, processing, and storage. Following these three major sections, there is a short section on networking, a reminder on *ulimits*, and a summary. All sections are updated to include changes pertinent to ASE 15.7.

The "Appendix A: Tuning and performance testing of SAP Sybase ASE 15 on AIX 6.1 and POWER7" section, which describes a laboratory exercise in tuning and testing SAP Sybase ASE 15 on POWER7 processor-based servers, is retained as pertaining to ASE 15.7 process mode and earlier releases of ASE. The "Appendix B: Comparing ASE 15.7 threaded mode with ASE 15.5 and ASE 15.7 process mode on AIX 7.1 and POWER7+" section reports comparative performance of ASE 15.5, ASE 15.7 threaded mode and ASE 15.7 process mode on the same workloads as were used in Appendix A. The "Appendix C: AIX 15.7 prerequisite – I/O completion ports" section contains details on activating and patching the input/output completion port (IOCP) subsystem which is used by ASE 15.7. All references (in the "Appendix D: Resources" section) have been updated.

The recommendations that are included in this white paper have been gathered and compiled from a number of resources, including:

- Experience in the field (pertaining to development, testing, and production systems) with ASE in the AIX environment
- Laboratory experiments conducted by specialists at SAP Sybase and IBM
- Close cooperation and discussions between AIX and ASE performance and development teams

System and database administration is close to being an exact science, but performance tuning is an art. The recommendations in this white paper are a starting point. Administrators can apply these techniques in consultation with users, business units, and application developers. Every enterprise has its unique

characteristics, its mixture of batch and online transaction processing (OLTP), its daily ebb and flow of demand, and its evolution and growth.

Likewise, both SAP Sybase and AIX specialists have changed their performance recommendations over time. This might continue to be the case as SAP Sybase ASE, the AIX operating environment, and the IBM Power Systems platform evolve in parallel. A formal partnership exists between IBM and SAP Sybase. IBM and SAP Sybase have committed to continuing a dialogue to keep their recommendations current and relevant.

The authors want to close the introduction with an observation and a strong recommendation. In many organizations, a significant disconnect exists between database and system administrators. This disconnect adversely affects system performance and availability. Therefore, it is important to establish close cooperation and direct communication between these units and roles to build stronger business benefits from the effective use of the system.

LPARs and dynamic logical partitioning (DLPAR)

Several IBM POWER4 processor-based UNIX servers and all IBM POWER5™, IBM POWER6®, and POWER7 and POWER7+ processor-based UNIX servers are LPAR-capable. That is, they support LPARs, which are multiple OS images of varying sizes and capacities, that exist on a single physical server frame. Even if there is only one image on an applicable hardware platform, it is still an LPAR. All of the tuning recommendations contained in this white paper apply to the AIX operating system images in the form of LPARs.

LPAR-capable POWER4, POWER5, POWER6, and POWER7 processor-based systems support the dynamic reallocation of memory and processors, referred to as DLPAR. The system administrator can add or remove processors and memory in an LPAR without rebooting, within the minimum and maximum memory and processor values specified in the LPAR profile. The same applies to virtual resources that are discussed in the following sections.

A note on optimization level

While the focus of this white paper is on ASE and AIX system tuning, the authors wish to call attention to the ASE configuration parameter, **optimization level**. This parameter is perhaps not as familiar as **optimization goal** or **compatibility mode** but needs to be taken into account when migrating from an earlier to a current version of ASE. Whereas, **optimization goal** optimizes for an OLTP, mixed or batch type workload, and **compatibility mode** allows the use of an ASE 12.5 level of optimizer, **optimization level** allows the database administrator and application administrator to control the level of optimizer changes within the ASE 15 family of servers. The default is the ASE 15.0.3 ESD1 optimizer; other choices are ASE 15.0.3 ESD2 and 3, and ase_current, which is the optimizer supported by the current installation, for example, ASE 15.7 ESD4. Thus, in migrating from ASE 15.0.3 to ASE 15.7, the administrator might choose to implement the default value until the optimizer enhancements of ASE 15.7 could be thoroughly tested with the applications. Full documentation of optimization level is available at: http://www.sybase.com/detail?id=1080354

Considering virtual-memory issues

Incorrect sizing or OS configuration can severely degrade performance. This section explains how ASE uses and AIX manages virtual memory and provides detailed configuration advice.

All host applications consume memory because of the code, static variables, and more. Certain applications have greater memory requirements than others. Typically, an administrator configures a relational database management system (RDBMS) to use the majority of the host's memory for data caches, but other applications can use significant amounts, as well. For example, memory must be available for:

- Operating system
- File-system buffer cache
- A GUI-based environment
- Batch operations, such as loading or unloading data (bulk copy)
- Maintenance operations, such as database backup and restore
- Other SAP Sybase applications, such as SAP Sybase Open Client and SAP Sybase Open Server
- Third-party applications and utilities that run on the same LPAR against the data server

If virtual memory approaches over-allocation (being overcommitted), then the operating system looks for memory pages to write to disk, free memory, and make it available for other purposes. This process, typically referred to as paging, is a concern only when some memory area approaches its limit or the search for victim pages requires significant resources. The main performance hit stems from the fact that the application expects to find some piece of data in memory, yet the data has been paged to disk. Retrieving paged data can result in throughput that is an order of magnitude slower. Although AIX and its applications continue to run in a degraded fashion while some paging space remains, paged-out code paths might result in application timeouts with unpredictable results.

ASE memory usage

SAP Sybase ASE uses memory for many purposes, including:

- Executable binaries
- Configuration (static and dynamic configurable parameters)
- Resources such as **number of open objects**
- User tasks (number of user connections)
- Data caches
- Procedure cache
- In-memory databases (ASE 15.5 and higher versions)
- Kernel resource memory (ASE 15.7)

During startup, the ASE engine allocates shared memory to meet all these needs, up to the limit defined by the **max memory** value. If **max memory** is set to less than the total of configured resources, then the engine does not start. If it is set higher than what the specified items need, the ASE instance requests only the required amount of memory.

The classic ASE scheme for allocating shared memory is to request the total **max memory** amount at startup. Users can revert to the classical scheme by setting the **allocate max shared memory** parameter to **1**. This does not limit the administrator's ability to reconfigure the ASE engine dynamically, unless the **dynamic allocation on demand** value has changed from its default setting. You can increase the amount of memory that the ASE engine uses by increasing the **max memory** value during run time. Optionally, the ASE engine also allows administrators to lock its memory into physical memory (refer to the "Locking (pinning) ASE memory in physical memory" section).

but if ASE memory is pinned, memory added by increasing **max memory** will not be pinned until the next restart.

SAP Sybase ASE maintains two other measures of memory besides **max memory**, namely **total logical memory** and **total physical memory**. You can observe them in the **sp_configure 'memory'** procedure (refer to Listing 1:).

total logical memory read-only	**63488**	**3189754**	**1594877**	**1594803 memory pages(2 k)**
total physical memory read-only	**0**	**3189756**	**0**	**1594878 memory pages(2 k)**

Listing 1: SAP Sybase ASE maintains two additional measures of memory (besides max memory)

The **total logical memory** value represents the amount of memory required by all objects that are allocated in the configuration file or through the **sp_configure** procedure; for example, users, caches, open databases, and open objects.

The **total physical memory** value represents the amount of memory that is actually in use at a given time. This fluctuates according to the number of users online and the number of open objects, among other factors. This value corresponds closely to the value of active virtual memory in the AIX **vmstat** report that is attributable to ASE (that is, minus the kernel and other running processes that use memory). The difference between the memory that is set aside for ASE by the OS and the amount actually allocated can create misunderstandings regarding the sizing of the system and the amount of free memory the system actually has. Moreover, AIX uses a lazy allocation policy such that even if **allocate max shared memory** is enabled, AIX does not attach the memory until ASE needs to access it. Refer also to the "Locking (pinning) ASE memory in physical memory" section.

ASE 15.7 update: kernel resource memory

As described in the ASE 15.7 *System Administration Guide,* kernel resource memory is required to bring online a given number of engines (both in process and thread mode) and is determined mainly by two factors, **max online engines** and **number of user connections**. For configurations of eight engines or less, it is recommended to add one 2 KB page of kernel resource memory for every two user connections above 100, and for 9 engines or more, increase kernel resource memory by one page for each user connection. The default value is 4096. The symptom of insufficient kernel resource memory is not being able to online the number of engines specified by **number of engines at startup** (process mode) or in the **syb_default _pool** (threaded mode). Increasing kernel resource memory can allow the additional engines

to be brought on line. ASE 15.7 threaded mode will be described in detail in the “Processor considerations” section.

AIX virtual memory

For a dedicated ASE host, the two largest consumers of memory are likely the ASE engine and the file-system buffer cache. When memory is close to being over-allocated (overcommitted), the AIX operating system pages memory to disk. This is also true for any other operating system in which ASE runs. File-buffer pages and pages that belong to running processes, such as ASE instances, can be paged to disk. AIX support refers to file-buffer paging as **page replacement** or **page stealing** as distinct from the paging of an application’s working set. Paging in the latter sense causes far more serious performance issues.

The size of the file-system buffer cache is set as a percentage of physical memory (RAM) using the **minperm%** and **maxperm%** parameters. The journaled file system (JFS) and the client file systems are managed within this cache. Common examples of client file systems are the enhanced journaled file system (JFS2) and the Network File System (NFS). The administrator can set the client file system buffer cache maximum size (through the **maxclient%** parameter) to a percentage of RAM. In practice, **maxclient%** is usually set to the same value as **maxperm%**.

An intuitive approach to tuning the file-system buffer cache size (familiar from earlier versions of IBM AIX 5L™) is to limit the cache size by setting the **maxperm%** parameter to a low figure, such as the memory left over after allocating for ASE, applications and the kernel, and then enforcing that figure as a hard limit by setting the **strict_maxperm** parameter to **1**. The **maxperm%** parameter defaults to a soft limit. Enforcing a hard limit has two drawbacks:

- When workloads change, the administrator must change the configuration.
- When memory is overcommitted, the system does not behave gracefully.

The following sizing and configuration guidelines represent the current best practices for ensuring performance and reliability, without demanding regular reconfiguration. These guidelines reflect the design of AIX virtual memory, which is to fill almost all memory left over from applications (up to all but **minfree)** with file cache, and to replace this file cache on demand. They are implemented as defaults in AIX 6.1 and AIX 7.1 and must not be altered without the advice of AIX support. Clients still using AIX 5.3 are advised to adhere to the same values as the defaults for AIX 6.1 and AIX 7.1.

Memory-sizing guidelines

An administrator usually sizes a dedicated ASE host so that it is large enough to accommodate one or more ASE instances. The general guideline is to dedicate as much memory as possible to the ASE engine and also to save space for other applications. The latter must be emphasized, as proper sizing is imperative in the AIX environment. The recommendations in this section strive to provide a solution, as robust as possible. Here is a simplified example of the sizing process:

The size of the AIX kernel is proportional to the total memory of the LPAR. The kernel grows between reboot operations—owing to JFS2 buffer allocation and inode caching. This is also discussed in the ”Locking (pinning) ASE memory in physical memory” section.

- If a host has 16 GB of RAM, the kernel will be around 1.5 GB in size. Then 14.5 GB or less should be considered available to other uses, including file-system buffer cache and ASE.
- Assuming that there are no other major applications and no windowing environment, the next item is the file-system buffer cache. In this example, it is possible to assume a total of 3 GB for the JFS and JFS2 file-system buffer caches, leaving 11.5 GB or less of memory that is available to applications, including ASE.

  Note: As explained in the following section, this does not imply that you need to limit file-system buffer-cache size to a maximum of 3 GB. Instead, you can determine this maximum by examining empirical data from monitoring. Current AIX best practices, detailed next, allow almost all memory that is not occupied by program and kernel memory to be occupied by file cache and to replace file cache on demand.
- The next consideration involves batch jobs and maintenance tasks. In this case, batch jobs do not run concurrently with maintenance tasks. The greater of these might consume 1 GB of memory for the duration of the operation, which leaves 10.5 GB available for ASE.
- Reserving 1 GB of memory (for headroom and future increases in batch jobs) suggests configuring ASE for 9.5 GB of memory.

  Note: In this example, if the 9.5 GB to the ASE engine is insufficient, you can add more physical memory to the host.
- As a rule of thumb, start sizing ASE on an LPAR dedicated to ASE with no other significant applications by allocating at most 70% of physical memory on the LPAR to ASE **max memory**: if there is more than one instance on an LPAR, then the sum of all **max memory**. Local applications consuming significant memory should be factored into the 70%. Over-allocating memory to ASE (or any large application) can result both in bad paging of computational pages and also in *file-cache thrashing* where normal page replacement is so rapid, or so many pages have to be scanned to find page-eligible ones, that it consumes a substantial amount of the processor and slows down overall processing.

AIX memory-configuration guidelines

A versatile operating system such as AIX, which runs applications of many different characteristics, requires that the administrator configures it specifically for its intended use. In the case of virtual memory, there are some areas of greater interest, where you need to ensure system performance and provide a robust configuration. Administrators can view and set parameters that are related to the AIX Virtual Memory Manager (VMM) through the AIX **`vmo`** command, which requires root privileges. However, you can see several VMM parameters by using the `vmstat -v` command, which by default, does not require root privileges.

- To view all parameters and their respective settings, use the **`vmo -a`** command (vmo –Fa in AIX 6.1 and AIX 7.1 to see restricted parameters).
- To set the **`maxperm%`** parameter to 90%, use the **`vmo -p -o maxperm%=90`** command.
- AIX versions 6.1 and 7.1 come with virtual memory defaults that usually make these changes unnecessary.
- Do not change AIX 6.1 and AIX 7.1 restricted tunables without the advice of AIX support.

The following recommendations largely correspond to AIX 6.1 and AIX 7.1 defaults and are included to provide the reader with a basic understanding of AIX virtual memory management.

maxperm% = **maxclient%** = <a high percentage for each>

[If possible, set these values so that **maxclient%** is always greater than **numclient%**. The current recommendation for AIX 5L and the default for AIX 6.1 and AIX 7.1 is 90%. To determine the current **numclient%** value, use the AIX **vmstat -v** command.]

- **minperm%** = <low percentage>

  [Set this value so that **minperm%** is always less than **numperm%**. The current recommendation for AIX 5L and the default for AIX 6.1 and AIX 7.1 is 3%, but you can set it as low as 1%. To find the current **numperm%** value, use the **vmstat -v** command.]
- **strict_maxperm = 0**
- **strict_maxclient = 1**
- **lru_file_repage = 0**
- **lru_poll_interval = 10**
- **page_steal_method = 1**

Notes:

- In contrast to many other UNIX operating systems, the AIX environment does not require you to configure limits for semaphores or shared-memory size (for example, compare this to the **kernel.shmmax** tunable [configuration parameter] on the Linux® operating system and /etc/system in Solaris).
- To find the amount of physical memory, use the AIX **lparstat –i** command (/usr/bin/lparstat, illustrated in Listing 5 under the "Processor considerations" section). The **vmstat -v** command also displays physical memory, but in 4 KB pages, and not in megabytes (MB).
- File cache size is reported by **vmstat –v**, which can be run by any user. The key values, **numperm%** and **numclient%** represent, respectively, total file cache and file cache consisting of **client** pages, such as JFS2 (64-bit native AIX file system), NFS, Common Internet File System (CIFS) and Veritas File System. These values are usually the same in contemporary systems owing to the obsolescence of the legacy 32-bit JFS.
- The recommended values apply to all AIX operating environments and to all applications running on AIX. They are set the same whether ASE is using buffered file system, direct I/O, or raw devices.

Setting the file-system buffer-cache configuration

The **minperm%** and **maxperm%** parameters, which represent a percentage of physical memory, specify the size of the AIX file buffers. Within this cache, the AIX operating system also manages file pages for client file systems (for example, JFS2 and NFS). Administrators set the maximum size of this subset using **maxclient%**, again representing a percentage of physical memory.[1]

Numperm% should be monitored continually, especially on a new system, until it reaches a steady state based on routine usage. If the **numperm%** parameter drops to less than the **minperm**% value, then the AIX operating system will swap out computational pages (for example, pages belonging to the ASE processes) instead of file cache. Therefore, you must set **minperm%** low (3% or less) to prevent this. Note that if **numperm%** is as low as 3% on a regular basis, then memory might be overallocated and file cache might have been effectively squeezed out of the system. Even if ASE uses raw data devices, its bulk copies, backups, and other maintenance operations need file cache, and if they have difficulty obtaining it, they will slow down and consume extra processor[2].

By setting the **maxperm%** parameter to a high value, the file-system buffer cache can grow, but not at the expense of computational pages, because **lru_file_repage** is set to **0** and **page_steal_method** is set to **1**.

The AIX operating system can page file-system buffer pages and computational pages to disk. When memory requirements exceed the amount that is currently available, a certain amount of paging

---

[1] JFS2 is the 64-bit AIX journaled file system, and is almost universally used instead of JFS, the 32-bit implementation. However, AIX virtual memory management still classifies JFS2 file systems as *client* file systems.

[2] One can reduce the burden on file cache by mounting file systems used for dumps, backups and load with the **release behind** options: **release behind read** for file systems typically read sequentially, **release behind write** for file systems typically written to sequentially, and **release behind read write** for file systems both read and written to sequentially. An ASE dump area which is swept by a tape backup utility would be a good use case for **release behind read write.**

occurs. At this time, the least recently used (LRU) algorithm comes into play (you are probably already familiar with LRU because of its use in ASE data caches). A good example of using the LRU algorithm involves burst loads where a large amount of file buffers are currently in use. Then, the AIX page-stealing daemon (**lrud**) looks at the memory pages to determine if a particular page has been paged recently; if not, the **lrud** process sends the page to disk. This happens irrespective of whether the page belongs to a file or a process. The recommended **lru_file_repage = 0** and **page_steal_method = 1** settings prevents the **lrud** process from paging computational pages except as a last resort. Setting **lru_poll_interval** to 10 can improve the responsiveness of **lrud** while it is running. The value of 10 is the default in AIX 5.3, 6.1, and 7.1.

Note: The **lru_file_repage** parameter is tunable in AIX 5L 5.2.05 and later. For AIX 5L 5.2, this tunable parameter is available as a fix through the authorized program analysis report (APAR) IY62224. The value of 0 causes **lrud** to ignore the age flag on the page when considering it to be paged out. This helps preserve computational pages. In AIX 7.1, **lru_file_repage=0** is *hard wired* and no longer visible to be modified. The **page_steal_method** tunable, introduced in AIX 5.3, TL6 also helps avoid paging out computational pages by maintaining separate lists of computational and file cache pages and favoring the latter for paging,

Here is a summary of the recommended virtual-memory configuration values for AIX 5.3, AIX 6.1, and AIX 7.1:

Parameter	Recommended Value for AIX 5.3, AIX 6.1 and AIX 7.1	AIX 6.1 and AIX 7.1 default	AIX 6.1 and AIX 7.1 restricted	AIX 5.3 default
minperm%	3	3	No	20
maxperm%	90	90	Yes	80
maxclient%	90	90	Yes	80
strict_maxclient	1	1	Yes	1
strict_maxperm	0	0	Yes	0
lru_file_repage	0	0/na*	Yes	1
lru_poll interval	10	10	Yes	10
Minfree	960	960	No	960
Maxfree	1088	1088	No	1088
page_steal_ method	1	1	Yes	0
Memory affinity	1	1	Yes	1
v_pinshm	0	0	No	0
lgpg_regions	0	0	No	0
lgpg_size	0	0	No	0
maxpin%	80	80/90*	No	80
esid_allocator	1 (AIX 6.1 and 7.1)	0/1	No	n/a

Table 1: Virtual memory configuration for AIX

Large memory segment aliasing

Large segment aliasing allows each memory segment lookaside buffer to address up to 1 TB of memory, reducing segment lookaside buffer faults and improving memory access. This is enabled by default on AIX 7.1, and is enabled using vmo –p –o esid_allocator=1 in AIX 6.1.

Locking (pinning) ASE memory in physical memory

Administrators can configure the ASE engine to lock (pin) its memory into physical memory. This requires both: operating system configuration and ASE configuration. Pinning ASE shared memory

* In AIX 7.1, maxpin% (maximum allowable pinned memory) default is 90%. In AIX 7.1, the lru_file_repage parameter is implemented by default and is neither listed nor modifiable.

used to be a common practice to avoid paging, but AIX performance support generally recommends against pinning (if memory is overcommitted, pinning makes matters worse). If paging is caused by incorrect virtual memory configuration, then that can be corrected without pinning memory. SAP Sybase and IBM only recommend pinning in order to use 16 MB large pages with ASE 15.0.3 and higher versions. If you decide to employ pinning, you must configure the ASE instance to allocate all configured and shared memory during startup. The following steps demonstrate the configuration of AIX and ASE for locking ASE memory:

1. To enable pinning from the OS level, use **`vmo -p -o v_pinshm=1`** command.
2. To configure ASE to lock shared memory, use the ASE **`sp_configure 'lock shared memory', 1`** command and **`sp_configure 'allocate max shared memory', 1`** command, or modify the values in the ASE <servername>.cfg file. It is not necessary to restart AIX but ASE must be restarted.

Notes:

- By default, AIX allows for the pinning of 80% of physical memory in AIX 6.1 and 90% in AIX 7.1. To change this, use **vmo-p –o maxpin%=<value>**. In general, this value must not be increased, or else, it might result in starvation of processes using file-system cache.
- Locking shared memory can cause issues for other applications if you have not taken into account the memory requirements of the kernel, other applications, and the file cache.
- You can dynamically configure the ASE memory to be larger or smaller by using the **sp_configure 'max memory', <new size**> command, and then use this memory for data caches or other purposes. The newly allocated memory is not locked into physical memory. Thus, to add memory to ASE and have it pinned, ASE needs to be shut down and restarted.
- The IBM AIX performance-development team generally recommends against pinning shared memory because the operating system can do a better job by allocating memory dynamically. AIX attaches mapped memory to ASE when ASE needs it and not before. AIX will also not attach the difference between ASE **physical memory** and **max memory**. In both cases, this memory is available to other processes and to file cache, whereas, if it was pinned, it might be unused by ASE and yet unavailable to other processes. As already noted, the only case in which AIX support recommends pinning memory is when using the large (16 MB) memory page for performance enhancement, which must be pinned. Refer to the "Using large pages with ASE" section for details.
- In AIX, kernel-pinned memory grows between OS restarts principally because it caches inodes and file system lookahead buffers. Inode caching can occupy up to 12% of pinned physical memory in AIX 5.3 and higher. It is not unusual to see 18% of physical memory that is pinned by the kernel. This kernel behavior must be taken into account in the relative sizing of ASE memory and file cache, whether shared memory is pinned or not, but it is especially important in the case of pinning.
- If the limit of pinned memory is exhausted between ASE shared memory and the kernel, the system stops. You can restrict the growth of the AIX kernel through certain tunable variables, but there can be performance tradeoffs. If kernel growth is a problem, you should place an AIX support call.

The best way to understand ASE engine memory requirements is to check the size of its shared memory. Compare the values reported by the ASE **sp_configure 'memory'** command (maximum, logical, and physical memory) with the value reported by the **ipcs -b -m** UNIX command.

The ipcs command

Refer to the example use of the AIX **ipcs** investigator command (shown in Listing 2):

```
# ipcs -b -m:

IPC status from /dev/mem as of Wed Dec 28 12:32:27 est 2005

T         ID     KEY         MODE           OWNER    GROUP     SEGSZ

Shared Memory:

m         8      0xd8009059  --rw-------    sybase   sybase    2147483648

m         9      0xd800905a  --rw-------    sybase   sybase    877723648

#
```

Listing 2: Example use of the AIX ipcs investigator command

The combined byte counts in the SEGSZ column (right-most column in Listing 2) might be much less than the ASE configuration **max memory** parameter. This indicates that ASE is configured not to allocate all memory at startup, but to allocate memory on demand. This is the default setting.

The AIX VMM has a rather lazy allocation algorithm for shared memory. This means that VMM reserves all the memory that an application requests (as seen through **ipcs –b –m**); however, VMM does not actually allocate the memory until the application refers to it (active virtual memory as measured by vmstat, sar and svmon). Even if the **sp_configure** parameter **allocate max shared memory** is set to 1 (*true)*, AIX does not allocate a shared-memory segment until it is referenced. The memory that ASE uses, as perceived by the OS (refer to "The vmstat command" section) increases over time and usage until it equals the size of the ASE logical memory. The actual size of allocated ASE shared memory seen by the Systems Administrator may differ from what the DBA expects. This is one of the many reasons for close cooperation between DBAs and system administrators.

Applications that terminate abnormally, including ASE itself, can leave shared-memory segments stranded. It is important that you explicitly remove these stranded-memory segments by using the AIX **`ipcrm`** command.

The vmstat command

Administrators can monitor virtual memory usage with the **`vmstat -I`** (upper case i) command. The output of this command contains relevant information in the **`avm`**, **`pi`**, **`po`**, **`fr`** and **`sr`** columns (refer to Listing 3:).

- The **`avm`** column shows the amount of virtual memory in use (in 4 KB memory pages).
- The **`pi`** column shows the number of cache misses that are paged in.
- The **`po`** column shows the number of pages that are paged out.

- The **fr** column shows the number of pages freed.
- The **sr** (scan rate) column shows the number of pages that were scanned to find eligible pages to page out.

```
kthr              memory             page                  faults
cpu

----- ----------- ------------------------ ------------ -----------

 r  b   avm    fre       re  pi  po  fr  sr  cy  in    sy     cs   us sy id wa

 2  0 2466459 12842265   0   0   0   0    0   0 2561  683764 13014  8  3  89  0

 0  0 2466459 12842205   0   0   0   0    0   0 3896  875010 18645  11  4 84  0

 1  0 2466462 12842010   0   0   0   0    0   0 10548 1170276 37027 26  9 65  0

 9  0 2466472 12841649   0   0   0   0    0   0 22147 1120355 66841 50 17 33   0
```

Listing 3: Using the vmstat command to monitor virtual memory usage

Note: In contrast to the other UNIX implementations, the AIX file-system I/O operations are separate from the paging operations. You can see this by looking at the **fi** and **fo** columns from the **vmstat -I** command output (shown in Listing 3:). This helps in determining whether memory is actually running low or not. Also, in contrast to other UNIX implementations, AIX does not clear the available memory of idle file pages until the space is needed for another process (for example, a Java™ heap). This can create the impression (for administrators new to AIX) that AIX is out of memory all the time.

Using large pages with ASE

Large pages (16 MB) can make a noticeable improvement in performance for large-memory configurations by decreasing the number of translation-lookaside buffer (TLB) misses. Field and lab experience shows that many ASE workloads achieve up to 15% performance improvement with 16 MB large pages. SAP Sybase ASE supports 16 MB large pages starting with version 15.0.3. All supported versions of AIX implement
16 MB pages.

Perform the steps for using large pages.

1. Ensure that ASE memory is locked according to the procedures given earlier.
2. Use the **lgpg_regions** and **lgpg_size** tunable parameters (in the **vmo** command) to specify how many large pages to pin. Be sure to round up to the next 16 MB page greater than the amount of **max memory**. For example, if **max memory** is set to 699999, you need to pin 85 large pages. The two parameters must be specified in the same command line, in this example, **vmo -p -o lgpg_regions=85 -o lgpg_size=16777216**. The change is dynamic and a reboot is not required. However, the **vmo** command warns you to run **bosboot** to make sure that the values are correctly set the next time you reboot. Two cautions apply:

a. AIX allocates 16 MB large pages in 256 MB frames. If the amount allocated in **lgpg_regions** is not an even multiple of 256 MB, then the remainder is allocated in 4K ordinary pinned pages. If the remainder is large, this might diminish the effectiveness of the 16 MB pages. So, stay as close as possible to the 256 MB boundary. If ASE **max memory** exceeds the amount of 16 MB pinned pages, the same will result – the remainder will be pinned in 4K pages.
b. If the sum of the memory pinned for ASE plus the AIX kernel exceeds the **maxpin%** value (p.12) the remainder of ASE shared memory will not be pinned at all, and thus is at risk of being paged out to swap space. If **maxpin%** is exceeded, there is a high risk of paging to begin with, so this condition should be avoided.

3. As root user, configure the application user ID with permission to use large pages, as follows:

   `chuser capabilities=CAP_BYPASS_RAC_VMM,CAP_PROPAGATE <user id>`

   For example, **`chuser capabilities=CAP_BYPASS_RAC_VMM,CAP_PROPAGATE sybase`** Afterwards, verify the change with **`lsuser -a capabilities sybase.`**

   Note: The sybase user must log out and log back in for the capability to take effect. So, be sure you log in as ***sybase*** after the capability change to restart ASE.
4. Verify the use of large pages with the **`vmstat -l`** or **`svmon -G`** (global view) command (svmon requires root access while **`vmstat -l`** does not).

   Listing 4 illustrates an LPAR with 16 MB large pages enabled and used by SAP Sybase ASE.
5. ASE must be restarted after large page pinned memory is configured, but AIX does not have to be restarted.

```
# svmon -G
               size       inuse        free         pin       virtual    mmode
memory       983040      951504       31536      820477      936839     Ded
pg space     737280      110327

                work        pers        clnt       other
pin           736973           0           0       34352
in use        875453           0                   26899

PageSize   PoolSize       inuse        pgsp         pin     virtual
s     4 KB          -      149600       55831      103773      171463
m    64 KB          -       18375        3406       13050       19164
L    16 MB        124         112           0         124         112
```

Listing 4: svmon –G command output (last line shows 112 16 MB large pages in use out of a pool of 124 such pages)

Reclaiming swap space

AIX versions 5.3, 6.1, and 7.1 periodically reclaim swap space for reuse after the pages it contains have been *paged back* into physical memory. This is true even when long-running processes, such as ASE, are still running. In AIX 5.3, AIX 6.1, and AIX 7.1, if the initial paging space is adequately sized (usually 8 GB to 16 GB for large memory systems), you do not have to worry about running out of swap space unless memory is actually overcommitted. Several tunable parameters control the reclamation of swap space, but the defaults are usually satisfactory. It is a better practice to use a monitoring tool, commercial or home grown, to alert the System Administrator to the use of swap space, than to create a huge swap space.

In AIX 5L V5.2, when paging occurs, occupied swap space is not released until the owner process terminates, even if the pages are paged back. This creates a small risk that long-running processes might run out of swap space. For the sake of performance, the best defense against this is to avoid paging altogether by using the virtual-memory tuning recommendations that have just been offered. Another option is to recalculate the memory needed for the ASE instance (**max memory**) and for the operating system and for related batch and maintenance processing. You can then pin this recalculated ASE **max memory**. Consider this as a last resort for AIX 5.2, which does not change the general advice against pinning*.

Processor considerations

The observations and recommendations in this section apply to the ASE 15.7 threaded kernel and process mode except where noted. Additional topics have been added to this section to address the new features of the threaded kernel, configuration parameters, and monitoring specific to the threaded kernel.

* AIX 5.2 is out of support with the exception of the AIX 5.2 workload partition (WPAR) supported under AIX 7.1 These remarks about AIX 5.2 paging are retained for the assistance of administrators who might be running legacy environments not yet migrated to supported versions of AIX.

Configuring the number of processors and ASE engines should be straightforward, but this is not always the case—especially when migrating an existing ASE from an older hardware to POWER6, POWER7, or POWER7+ processor-based systems. In the initial sizing of a new or migrated ASE partition, it is best to allocate one engine per virtual processor or core and to exceed that only after live testing (refer to the "SMT and online engines" section). The best practice for ASE 15.0.3, ASE 15.5, and ASE 15.7 process mode is to bring no more ASE engines online than you can maintain at least 60% *CPU busy* on average as measured by sp_sysmon (in ASE 15.7 threaded mode there are additional factors to consider—refer to the "ASE 15.7 and the threaded kernel" section). However, the existence of varying workloads and intermittent-burst activity, as well as the anticipation of added users and application functions, can take precedence over the recommended configuration. In short, there might be a sacrifice of optimal performance configuration at average utilization in order to provide ample headroom. When there is also a need to allocate processor costs among departments, the allotment of processors becomes even more challenging.

Administrators can monitor the processor through various means. As already mentioned, one common method is through the **vmstat** command, which displays system, user, I/O wait, and idle times. The AIX command, **prtconf**, gives an easy-to-read summary of memory and processor. More complete configuration information is obtained from **lparstat –i** (/usr/bin/lparstat), which can be run by any user. An example of a dedicated processor LPAR is given first, followed by a shared processor example. The shared processor terminology is explained in the "Shared-processor LPARs" section.

```
Lparstat –i  partial listing illustrating a dedicated, POWER7 LPAR with four SMT
threads

Node Name                                  : f3dn23

Partition Name                             : 10-326FB

Partition Number                           : 1

Type                                       : Dedicated-SMT-4

Mode                                       : Capped

Entitled Capacity                          : 8.00

Partition Group-ID                         : 32769

Shared Pool ID                             : -

Online Virtual CPUs                        : 8

Maximum Virtual CPUs                       : 8

Minimum Virtual CPUs                       : 1

Online Memory                              : 61184 MB

Maximum Memory                             : 61184 MB

Minimum Memory                             : 256 MB
```

```
Variable Capacity Weight                    : -

Minimum Capacity                            : 1.00

Maximum Capacity                            : 8.00

Capacity Increment                          : 1.00

Maximum Physical CPUs in system             : 16

Active Physical CPUs in system              : 16

……..

Lparstat –i partial listing illustrating a shared processor POWER6 LPAR with two
SMT threads and three virtual processors and entitled capacity of 1



Node Name                                   : atssg54

Partition Name                              : vios2_atssg54_VM1_Sybase1

Partition Number                            : 5

Type                                        : Shared-SMT

Mode                                        : Uncapped

Entitled Capacity                           : 1.00

Partition Group-ID                          : 32773

Shared Pool ID                              : 0

Online Virtual CPUs                         : 3

Maximum Virtual CPUs                        : 8

Minimum Virtual CPUs                        : 1

.....................................................
```

Listing 5: Using the lparstat –i command to obtain processor information in dedicated and shared environments

General sizing guidelines

There is a temptation, in particular when consolidating systems or migrating from slower processors to faster (for example, POWER6 and POWER7) processor-based environments, to keep the previously assigned processor configuration and engine allocation. This often leads to idle ASE engines and wasted processors. In earlier generation reduced instruction set computer (RISC) processors, using more active ASE engines than the number of physical processors proved detrimental to performance, throughput, and response times. Since the introduction of POWER6 processor-based systems in 2008, allocating more engines than cores (or virtual processors) has become commonplace. One active ASE engine per physical processor, or core, stands as a sizing guideline for POWER5 processor-based systems and as a starting

point for sizing POWER6, POWER7 and POWER7+ processor-based systems. The corresponding starting point for sizing ASE in a shared processor environment, **engines = virtual processors**, is explained in the section on shared-processor LPARs. Refer to the “SMT and online engines” section for further discussion of the ratio of engines to processors.

Some environments, by contrast, benefit from the reverse: running fewer ASE engines than there are processors. Other tasks (applications and OS processes) require processor capacity. This includes native threads, for example, SAP Sybase Real-Time Data Service (RTDS) and the asynchronous I/O servers for asynchronous I/O processes on the file systems. The latter is often forgotten, which leads to poor performance in high-workload situations. For this reason, administrators must take asynchronous I/O servers and other processes into account when allocating processor resources. It is up to you to test and decide what is most beneficial to your organization.

An ASE that uses raw devices makes little use of the asynchronous I/O subsystem. So, if there are no other substantial applications running on the LPAR, *engines=cpus (engines=virtual processors)* is a safe setting. When migrating a relatively constant workload from slower processors to faster processors, the best practice is to reduce the number of online engines proportionally to the faster clock speed while taking account of the requirements of user connections and physical I/O. Then, add engines and processors while monitoring peak and sustained utilization to ensure that ample headroom is available.

Given the very high clock speeds of the POWER6, POWER7 and POWER7+ processors, ASE engine allocation presents two special challenges: avoiding high rates of idling and avoiding spinlock contention.

Idling engines (ASE 15.7 process mode, ASE 15.03, ASE 15.5)

ASE engines are peers; that is, each is responsible for its own network and disk I/O processes. Through ASE 15.7 in process mode, an engine that has started an I/O method must also manage it (from polling to I/O completion and delivering the I/O event to the sleeping task). As an ASE engine starts idling (for example, when there are no tasks on local or global run queues that the engine can run) it does not yield the processor immediately. Instead, the ASE engine tries to stay on the processor, as this reduces the additional processor load of frequently rescheduling the engine. Therefore, instead of yielding, the engine looks for work to do by continuously checking for network I/O, completed disk I/O processes, and other tasks to run from the run queues. On the AIX operating system, the completion of disk I/O methods does not require a system call, but polling the network does.

For a busy ASE engine, user time typically far outweighs system time, but an idling engine reverses this ratio; the data server process consumes a significant amount of system time and a marginal amount of user time. The AIX trace report **tprof** reports a very high rate of **select()** calls when engines idle. Use the ASE **sp_sysmon** system procedure together with **vmstat** and **TOPAS_NMON** to troubleshoot the throughput issue. In an extreme case, you might witness 100% processor utilization at the OS level (with a high percentage of system processor usage) and only 25% average processor busy at the ASE level. If the system is improperly sized, then the system processor time consumed in polling might actually degrade the overall performance, even on a dedicated ASE server. This is because the polling engines can prevent engines with work to do from being dispatched as soon as they are runnable. SAP Sybase ASE has a configuration parameter, **runnable process search count** (RPSC), which controls the number of times an idle engine polls before yielding the processor. Reducing **runnable process search count** is

intended to reduce the amount of kernel processor consumed by idling engines. The use of this parameter in conjunction with engine count is discussed in the following section.

After an ASE has started, all its configured engines, that is, the number specified by *number of engines at startup*, administrators can bring additional ASE engines online dynamically. The ASE configurable parameter *number of engines at startup* defines the number of engines that are booted during startup. If an administrator configures additional engines through the ASE `max online engines` parameter, these new ASE instances can come online by using the ASE `sp_engine 'online'` command.

Note 1: It is preferable to bring ASE engines online when the system is idle or nearly idle.

Note 2: ASE supports taking engines offline through the ASE procedure `sp_engine 'offline' [, <engine ID>]`. However, this does not always succeed because the designated engine might hold resources that cannot be released. You can use the `can_offline` argument to `sp_engine` to find out whether it is possible to take an engine offline.

ASE engine configuration for AIX and runnable process search count (ASE 15.7 process mode, ASE 15.03, ASE 15.5)

Earlier in this discussion of processor best practices for SAP Sybase ASE, idling engines has been discussed and **runnable process search count** has been referred to. To decrease the amount of processor time that idling ASE engines consume, it is sometimes suggested that the DBA set the **runnable process search count** configuration parameter to a lower number than the default of 2000. An engine that cannot find a task to run in local or global queues will only loop that lower number of times looking for work, instead of the default 2000 times. A commonly suggested value is 3, which was the default value in an earlier version of ASE. Notice that the value of **runnable process search count=0** has the reverse effect, preventing the engine from ever yielding.

There are several reasons for concern about decreasing the `runnable process search count` (RPSC) parameter:

- Reducing RPSC can delay the completion of pending network I/O processes. You can directly observe this through **vmstat** and other OS monitors as a significant increase in *I/O wait* time that is attendant to reduced RPSC. I/O wait is defined as the percentage of time the processor is idle while one or more I/O tasks are pending. Reducing RPSC causes the ASE engines to spend more time sleeping, rather than polling for I/O requests.
- No value other than the default works consistently. Some sites use 1000, others use 100, while others use 3. It is a matter of trial and error to find an RPSC that improves throughput.
- The net effect of reducing RPSC, at best, is less than that is obtained by reducing the number of engines online for the given volume of work. For example, an engine with at least one pending asynchronous I/O will repeatedly check for completion of the I/O, even after RPSC is exhausted. Field experience with ASE and AIX 5L and AIX 6.1 on the POWER6 processor-based architecture generally shows degradation of performance when RPSC is reduced from 2000.
  Note: For ASE 15 Cluster Edition, the default RPSC is 3. It is recommended to increase this configuration value. Cluster Edition does not use a native thread for cluster interprocess communication (IPC) on the AIX platform. If the engine is sleeping, it would delay cluster IPC messages and result in decreased performance. This is especially true when the second instance

is passive. If no work is going on, a lower RPSC will have the engine sleeping more often and messages from the primary instance will be delayed.

On the other hand, there are two convincing cases for testing the reduction of RPSC to reduce idle polling and benefit standalone ASE performance:

- The case of an I/O bound workload
- The case of a workload with extreme fluctuations of activity, as between 20% and 100% processor busy

Field and lab results suggest that reducing RPSC is more effective on the POWER7 processor-based architecture than it was on POWER6, but the same results confirm the best practice of achieving an optimal engine count with RPSC left at the default of 2000 before attempting to reduce RPSC. Put another way, reducing RPSC does not compensate for the degraded performance of an excessive number of online engines.

Spinlock contention

Developers and administrators who work with database management systems (DBMSs) are well aware of the need to manage concurrency. The DBMS itself manages concurrency by enforcing isolation levels and protecting data, typically through locking. The same risks and needs are present for hardware and operating systems. The potential issues are managed through constructs such as semaphores and spinlocks. The remedies and potential issues are collectively termed mutual exclusion.

The semaphore is used for long-term locks, and spinlocks are exclusively for short-term locks. Tasks waiting for a semaphore typically sleep, although the behavior for tasks that are waiting for a spinlock differs significantly. As the term spinlock indicates, a task that is waiting for a spinlock hardly sleeps, but continuously polls whether the spinlock is available to grab. Hence, spinlock contention also manifests itself through increased processor usage without increased throughput.

The high clock speed of the POWER6, POWER7, and POWER7+ processors might aggravate spinlock contention. This further underscores the need to diagnose and tune for spinlock contention. There are hundreds of spinlocks in ASE, and only *object manager* and *data cache* spinlocks are reported in default **sp_sysmon.** The **sp_sysmon –dumpcounters** output is notoriously hard to interpret, and SAP Sybase support maintains a **spinmon** utility, which organizes the dumpcounters output by ranking the busiest spinlocks by their rate of contention. In addition, ASE 15.7 ESD2 and above provide SAP Sybase ASE monitoring and diagnostic agent spinlock monitoring, The most effective first measure in reducing spinlock contention is, if possible, to reduce engine count. Additional measures include partitioning data caches and binding hot object descriptors in the metadata cache. You can find the general guidelines for ASE spinlock tuning in the ASE *Performance and Tuning Guides*.

ASE 15.7 and the threaded kernel

ASE 15.7 ESD4 (current release at the time of publishing this paper) has two requirements on AIX which do not apply to earlier releases. First, ASE 15.7 ESD4 requires AIX 6.1 Technology Level 7 or above.

Second, ASE 15.7 requires the activation of the **I/O completion port** subsystem. IOCP is explained in general in the present section, and the precise installation requirements are provided in Appendix C.

The ASE 15.7 threaded kernel is extensively documented by SAP Sybase* and for the purposes of this white paper we will focus on those features which are pertinent to implementation on AIX and Power Systems.

The threaded kernel is enabled by default in ASE 15.7, but the traditional process kernel in which each engine is a separate process and does its own disk and network I/O polling is still available. The kernel mode is a configuration parameter which can be changed through **sp_configure "kernel mode", 0, [threaded, process]** to enable one or the other mode after an ASE restart. The running kernel mode is displayed by **_configure "kernel mode"** without arguments, and by **select @@kernelmode**. Changing kernel mode requires a restart of ASE.

The threaded kernel implements the traditional ASE engine architecture as a single OS process which manages three or more thread pools. The threads are *native threads* as distinct from Portable Operating System Interface (POSIX) threads. The running data server presents itself at the OS level as a single process, rather than a separate process for each engine.

A particular advantage of the threaded kernel is in having less context to restore when a thread is dispatched, as compared with a process. Thus, there should be less latency with dispatching an engine on a different core or virtual processor.

The threaded kernel implements another level of asynchronous I/O: engines will initiate network sends, Component Integration Services (CIS) requests and disk I/O requests, then hand off the handling of polling for the completion of those requests to dedicated network, disk, and CIS threads. This releases the engine to handle other runnable tasks. The engine in the threaded kernel is no longer responsible for completing an I/O it has dispatched, nor is it responsible for completing a network request that was assigned to it. When an I/O completes, the waiting task can be run by the next available engine, not just its initiator.

The threaded kernel implements the I/O completion port API to optimize asynchronous I/O. An input/output completion port object is created and associated with a number of sockets or file handles. When I/O services are requested on the object, completion is indicated by a message queued to the I/O completion port. A process requesting I/O services is not notified of completion of the I/O services, but instead checks the I/O completion port's message queue to determine the status of its I/O requests. The I/O completion port manages multiple threads and their concurrency.* Details on setting up IOCP are provided in Appendix C.

The ASE 15.7 kernel supports four species of thread pools:

---

* Introduction to the ASE 15.7 thread kernel can be found in the *ASE 15.7 New Features Guide* at: http://adaptiveserver.de/wp-content/uploads/2012/01/asenewfeat.pdf and in the following two presentations by Peter Thawley:
  - ASE 15.7 technical overview-- Peter Thawley https://websmp104.sap-ag.de/~sapidp/011000358700001121852012E/assets/sybase_ase_techovervw.pdf
  - ASE 15.7 Technical look inside the "hybrid kernel" -- Peter Thawley http://www.sybase.com/files/Product_Overviews/ASE-15.7-New-Threaded-Kernel.pdf

* The definition of IOCP is courtesy of Wikipedia.

- **syb_default_pool**, which contains general purpose engines
- **syb_system_pool**, which contains the system, ctlib (CIS) disk, and network controller threads
- **syb_blocking_pool**, which is dedicated to running long blocking calls
- **User defined pools**, which contain engines normally used in conjunction with execution classes, and which are typically bound to an execution class

Execution classes are implemented through a legacy ASE feature, the **Logical Process Manager**, which allows users, groups, or named applications to be bound to one or more engines and assigned priorities. In earlier ASE releases, engines bound to execution classes were able to run tasks other than those assigned by the execution class. In the current implementation, these engine threads are entirely dedicated to the execution class to which they are bound. This new implementation promises to make execution classes more useful.

Engine pools are managed through three new commands each having several options – create, alter, and drop thread pool. The number of disk or network controller threads is controlled by **sp_configure** or configuration file parameters **number of disk tasks** (default 1) and **number of network tasks** (default 1)**.**

The number of engines that can be brought online in syb_default_pool plus any user-defined pools is limited, as in earlier ASE, by the configuration parameter **max online engines.** The **number of engines at startup** is ignored by the threaded kernel, which looks at the number of engines assigned to the thread pools instead. As noted earlier in the “ASE 15.7 update: kernel resource memory” section, underallocation of kernel resource memory might unintentionally limit the number of engines that can be brought online.

The ASE 15.7 kernel has five configurable features:

- The facility to create user thread pools and bind them to execution classes
- The engine thread count, in **syb_default_pool** or in a user defined pool
- The network and disk controller thread count
- The idle timeout – a microsecond counter that determines when an engine yields the processor, defaulting to 100 microseconds
- The allocation of processor at the OS level, dedicated or virtualized, with respect to the thread count

Owing to the complete redesign of I/O handling in the threaded kernel, familiar configuration parameters in earlier ASE releases (**runnable process search count** and **I/O polling process count**) do not apply to the threaded kernel. This is because the polling process that an idling engine went through before voluntarily yielding the processor (governed by **runnable process search count**) has been replaced by the default 100 microsecond idle loop before yielding. In ASE 15.7, the polling is performed by the disk and network controller threads, instead of the engine. Ctlib I/O, used by replication agents and CIS proxy and remote procedure calls, is handled by its own thread in system_pool.

The threaded kernel changes the way in which ASE dispatches I/O requests, the way it handles completed I/O requests, and the way it implements the engine’s searching for work and idle cycles.

In the process kernel, the engine itself submits network and disk read and write calls, and the engine polls for completed network and disk I/Os.

In the threaded kernel, ASE uses the IOCP **ReadFile()**and **WriteFile()** calls for network operations. It calls

CreateIoCompletionPort() for the disk I/O when running in threaded mode, but still uses **aio_read()** and **aio_write()** to issue the async I/O. Polling is done by calling the IOCP routine, **GetMultipleCompletionStatus()**.

In the process kernel, an engine with no runnable task checks the global queue for tasks it can take over, then checks other engine's queues for eligible tasks to take over, polls for completed network I/O and disk I/O. If no work is found in the number of these cycles specified by runnable process search count (default 2000), issues a blocking network I/O that causes it to be descheduled. Eligible tasks for a process engine are restricted due to the requirement that the initiating engine complete its own network and disk I/Os.

In the threaded kernel, this restriction no longer exists, so any engine can take over any runnable task. When an I/O completion is received from IOCP by the network or disk controller thread, the task sleeping on that I/O is set to runnable, and the next available engine dispatches it. The threaded engine without a runnable task still checks the global and engine queues for eligible tasks, but it does not loop looking for network and disk I/O completions. Instead, the threaded engine loops looking for runnable tasks for the number of microseconds specified by the **idle timeout** parameter before issuing **pthread_cond_timedwait()** which yields the processor.

The changes to thread implementation, I/O handling, and idling in the threaded kernel have significant implications for performance. Handing off network and disk I/O to dedicated threads has the potential of freeing the engine to run any task that is not waiting on I/O, a potential benefit to computationally-intensive workload. The elimination of the engine's polling loop implemented in AIX by the system call **select()** has the potential to reduce a significant source of additional processor load when engines are mostly idle. But, because of the brevity of idle timeout, the frequency of yields might be much higher in the threaded than in the process kernel. This frequency of yields and reschedules is compensated by the *lighter* context that has to be restored or migrated when an engine thread is rescheduled. Finally, it might be anticipated that an I/O bound workload realizes less benefit from handing off I/O if it lacks tasks to run that are not **also** waiting on I/O.

The main new tuning tasks presented by the threaded kernel are:

- To determine whether engine count and processor allocation play the same way as they did in earlier ASE: how do you tell whether you have too few or too many engines
- To determine when more than the default one of each of network and disk controller threads are needed
- To determine when idle timeout needs to be adjusted and by how much
- To understanding the new metrics in sp_sysmon that are used to make the above determinations

ASE 15.7 threaded kernel: New sp_sysmon metrics

Engine utilization

This report differs from process_mode **sp_sysmon** *Engine Busy Utilization* in two ways. *CPU busy* in the process version is broken out as *user busy* and *system busy* in the threaded version. And, in the threaded version, *I/O busy* instead of reporting for each engine its own idle time with an I/O pending,

reports an engine's idle time with an I/O pending for *any* engine. This is related to the fact that any engine can reschedule a task blocking on an I/O when that I/O completes.

The first example is a processor-intensive workload:

```
Engine Utilization (Tick %)          User Busy      System Busy     I/O Busy      Idle

  ThreadPool : syb_default_pool

   Engine 0                           90.3 %          4.8 %          4.7 %        0.2 %

   Engine 2                           90.8 %          3.9 %          5.2 %        0.1 %

   Engine 3                           89.4 %          5.6 %          4.9 %        0.1 %

   Engine 4                           91.2 %          4.6 %          4.2 %        0.0 %

   Engine 5                           88.4 %          6.7 %          4.9 %        0.0 %

  -------------------------  ------------  ------------  ----------  ----------

  Pool summary   Total                450.2 %         25.5 %         23.9 %       0.3 %

                 Average               90.0 %          5.1 %          4.8 %        0.1 %
```

The next example is during a **dbcc textalloc** maintenance job with very low engine utilization and a high value for idle time with some I/O pending This does not necessarily reflect an **I/O bottleneck**:

```
Engine Utilization (Tick %)                User Busy    System Busy   I/O Busy      Idle

  -------------------------                ------------  ------------  ----------  ----------

  ThreadPool : syb_default_pool

   Engine 0                                   3.5 %         0.1 %       93.9 %       2.5 %

   Engine 1                                   2.6 %         0.2 %       94.7 %       2.5 %

   Engine 2                                   2.7 %         0.1 %       94.7 %       2.4 %

   Engine 3                                   3.6 %         0.0 %       93.9 %       2.6 %

   Engine 4                                   2.2 %         0.1 %       95.2 %       2.5 %

   Engine 5                                   4.3 %         0.1 %       93.2 %       2.4 %

   Engine 6                                   3.2 %         0.1 %       94.2 %       2.5 %

   Engine 7                                   2.7 %         0.1 %       94.6 %       2.6 %

   Engine 8                                   4.0 %         0.1 %       93.6 %       2.3 %

  -------------------------  ------------  ------------  ----------  ----------

  Pool summary               Total          28.8 %         0.8 %      848.1 %      22.3 %

                             Average         3.2 %         0.1 %       94.2 %       2.5 %
```

Average runnable tasks

This new metric reports the number of runnable tasks per engine that are waiting for cycles. Runnables are reported as of 15, 5, and 1 minutes ago, allowing to trend the application in terms of relative load.

In the first example, the average number of runnable tasks is low, reflecting an adequate number of engines and indirectly the I/O bound character of the particular workload:

```
Average Runnable Tasks            1 min         5 min      15 min  % of total
  -------------------------  ------------  ------------  ----------  ----------
  ThreadPool : syb_default_pool
   Global Queue                       0.0           0.0         0.0       0.2 %
   Engine 0                           1.0           0.6         0.3      28.6 %
   Engine 2                           0.7           0.5         0.2      19.1 %
   Engine 3                           0.6           0.4         0.2      17.5 %
   Engine 4                           0.6           0.6         0.3      17.8 %
   Engine 5                           0.6           0.5         0.2      16.7 %
  -------------------------  ------------  ------------  ----------
  Pool Summary        Total           3.6           2.6         1.2
                    Average           0.6           0.4         0.2
```

The second example, with a higher rate of runnable tasks, suggests the need for additional engines, and interestingly reflects the fact that the workload had only started running 5 minutes before the data collection:

```
Average Runnable Tasks            1 min         5 min      15 min  % of total
  -------------------------  ------------  ------------  ----------  ----------
  ThreadPool : syb_default_pool
   Global Queue                       0.1           0.1         0.1       0.3 %
   Engine 0                           4.9           3.7         3.2      21.4 %
   Engine 1                           3.0           3.2         3.3      13.1 %
   Engine 2                           3.7           3.8         3.6      16.4 %
   Engine 3                           3.9           3.6         3.3      16.9 %
   Engine 4                           3.4           3.6         3.4      14.9 %
   Engine 5                           3.9           2.4         1.1      17.0 %
  -------------------------  ------------  ------------  ----------
  Pool Summary        Total          22.8          20.2        18.0
                    Average           3.3           2.9         2.6
```

Processor yields by engine

This report differs from the process mode **sp_sysmon**, which reported the number of disk and network polls by engine, in that it compares the rates at which idle, yielding engines slept their full time slice before being reawakened and the rates at which they were awakened by an interrupt before the full time slice elapsed. This is a potential measure of the adequacy of the idle timeout, and another potential measure of the adequacy of the engine count. A high rate of interrupted sleeps might indicate a need to increase the *idle timeout*, or to bring more engines online. In the first example, the full sleeps outnumber the interrupted sleeps almost 3:1, suggesting an adequate engine count, and reflecting an I/O intensive workload:

```
CPU Yields by Engine              per sec      per xact       count  % of total
  -------------------------  ------------  ------------  ----------  ----------
  ThreadPool : syb_default_pool
   Engine 0
      Full Sleeps                    29.4           0.0        3529      14.6 %
      Interrupted Sleeps             10.7           0.0        1281       5.3 %
```

```
   Engine 2
      Full Sleeps                    29.4           0.0        3524      14.6 %
      Interrupted Sleeps             10.1           0.0        1206       5.0 %
   Engine 3
      Full Sleeps                    30.7           0.0        3682      15.3 %
      Interrupted Sleeps             10.4           0.0        1249       5.2 %
   Engine 4
      Full Sleeps                    29.1           0.0        3494      14.5 %
      Interrupted Sleeps             10.8           0.0        1295       5.4 %
   Engine 5
      Full Sleeps                    30.2           0.0        3626      15.0 %
      Interrupted Sleeps             10.5           0.0        1255       5.2 %
  -------------------------  ------------  ------------  ----------
  Pool Summary                     201.2           0.0       24141
```

In the second, processor-intensive example, the engines never get through a full sleep, in this case indicating the need for more engines. In other cases, a high rate of interrupted sleeps might indicate a need to increase engine *idle timeout*.

```
CPU Yields by Engine             per sec      per xact       count  % of total
  -------------------------  ------------  ------------  ----------  ----------
  ThreadPool : syb_default_pool
   Engine 0
      Full Sleeps                    0.0           0.0           0       0.0 %
      Interrupted Sleeps            31.4           0.0        3765      17.8 %
   Engine 1
      Full Sleeps                    0.0           0.0           0       0.0 %
      Interrupted Sleeps            29.9           0.0        3591      17.0 %
   Engine 2
      Full Sleeps                    0.0           0.0           0       0.0 %
      Interrupted Sleeps            30.2           0.0        3629      17.1 %
   Engine 3
      Full Sleeps                    0.0           0.0           0       0.0 %
      Interrupted Sleeps            31.9           0.0        3825      18.1 %
   Engine 4
      Full Sleeps                    0.0           0.0           0       0.0 %
      Interrupted Sleeps            29.7           0.0        3564      16.8 %
   Engine 5
      Full Sleeps                    0.0           0.0           0       0.0 %
      Interrupted Sleeps            29.6           0.0        3554      16.8 %
  -------------------------  ------------  ------------  ----------
  Pool Summary                     176.5           0.0       21175
```

Thread utilization at the OS level

This new report gives the percentage of real time that a thread spends on a processor, and estimates the number of processor units consumed.

```
Thread Utilization (OS %)      User Busy   System Busy        Idle
  -------------------------  ------------  ------------  ----------
  ThreadPool : syb_blocking_pool : no activity during sample

  ThreadPool : syb_default_pool
   Thread 6    (Engine 0)          89.2 %         1.3 %       9.5 %
   Thread 7    (Engine 1)          90.3 %         1.1 %       8.6 %
   Thread 8    (Engine 2)          83.5 %         1.2 %      15.3 %
   Thread 9    (Engine 3)          72.0 %         1.1 %      26.9 %
   Thread 10   (Engine 4)          88.9 %         1.3 %       9.8 %
   Thread 11   (Engine 5)          90.3 %         1.4 %       8.4 %
   Thread 12   (Engine 6)          80.5 %         1.0 %      18.5 %
   Thread 13   (Engine 7)          79.4 %         1.2 %      19.4 %
   Thread 14   (Engine 8)          89.5 %         1.3 %       9.2 %
```

```
  Thread 15   (Engine 9)          62.7 %          0.8 %       36.5 %
  Thread 16   (Engine 10)         90.2 %          1.4 %        8.4 %
  Thread 17   (Engine 11)         89.3 %          1.2 %        9.5 %
    ….          ….                ….              …            …


  Thread 27   (Engine 21)         81.3 %          0.9 %       17.8 %
  Thread 28   (Engine 22)         89.2 %          1.4 %        9.4 %
  Thread 29   (Engine 23)         87.2 %          1.4 %       11.5 %
  Thread 30   (Engine 24)         82.8 %          1.2 %       16.0 %
  Thread 31   (Engine 25)         90.4 %          1.4 %        8.2 %
  Thread 32   (Engine 26)         79.8 %          1.2 %       19.0 %
  Thread 33   (Engine 27)         89.1 %          1.4 %        9.5 %
 ------------------------  ------------  ------------  ----------
 Pool Summary      Total        2323.3 %         33.7 %     443.0 %
                 Average          83.0 %          1.2 %      15.8 %

 ThreadPool : syb_system_pool
  Thread 1    (Signal Handler)     0.1 %          0.1 %       99.8 %
  Thread 34   (NetController)      1.2 %          1.5 %       97.3 %
  Thread 35   (DiskController)     7.9 %         15.2 %       76.9 %
  Thread 36   (Network Listener    0.0 %          0.1 %       99.9 %
 ------------------------  ------------  ------------  ----------
 Pool Summary      Total           9.2 %         16.8 %     374.0 %
                 Average           2.3 %          4.2 %      93.5 %
Adaptive Server threads are consuming 23.8 CPU units.
 Throughput (committed xacts per CPU unit) : 1179.4
```

In this example of a large system running a processor-intensive workload, all three thread pools: the engine pool **syb_default**_pool, **syb_system_pool,** and **blocking_pool** are reported. In this illustration, the engines are spending 84% of their time on the processor, while the disk controller thread is moderately active. The system threads contribute a little to the overall processor consumption. This ASE server is using about 84% of its 28 dedicated processors.

Context switches at the OS level

Context switches at the OS level are another new metric provided by ASE 15.7 sp_sysmon. These context switches are distinct from ASE task context switches, which reflect whether a task is runnable or blocked on I/O, a lock or some other ASE resource. Rather, this metric is another way of looking at processor yields, already reported in terms of full compared to interrupted sleeps. What is measured here is the rate at which the ASE engine yielded the processor voluntarily by issuing **pthread_cond_timedwait()** or whether it used up its OS time slice or was pre-empted by the OS scheduler.

```
Context Switches at OS            per sec      per xact       count  % of total
 ------------------------  ------------  ------------  ----------  ----------
 ThreadPool : syb_blocking_pool
  Voluntary                         0.0           0.0           0       0.0 %
  Non-Voluntary                     0.0           0.0           0       0.0 %
 ThreadPool : syb_default_pool
  Voluntary                        79.6           0.0        9547       1.1 %
  Non-Voluntary                    83.4           0.0       10010       1.2 %
 ThreadPool : syb_system_pool
  Voluntary                      6713.6           0.9      805636      93.2 %
  Non-Voluntary                   325.2           0.0       39025       4.5 %
 ------------------------  ------------  ------------  ----------  ----------
  Total Context Switches         7201.8           0.9      864218     100.0 %
```

Simultaneous multithreading (POWER5, POWER6, and POWER7 with AIX 5.3, AIX 6.1, and AIX 7.1)

Simultaneous multithreading (SMT) is a POWER5, POWER6, and POWER7 processor-based technology that goes beyond traditional symmetric multiprocessing (SMP) and concurrent multiprocessing as implemented in other hardware platforms, in allowing two or four threads to run the same type of operation (for example, integer or floating point) at the same time. Its implementation, through duplicate registers, added control logic and queues, presents two or four **logical processors** to the operating system for each physical or virtual processor that is allocated to the LPAR. Although the IBM SMT implementation does not double the performance per core, it might increase throughput by 30% (POWER5 and IBM POWER5+) to 50% (POWER6 and POWER7). Most RDBMS workloads enjoy the benefit of SMT, but certain highly uniform and repetitive processes do not. The AIX environment enables SMT by default: in POWER5 and POWER6, this is dual-threaded mode, and in POWER7, quad-threaded mode. However, the administrator can change the SMT type (on/off or, in POWER7 mode, from four to two threads[1]) without the need for a reboot by using the **smtctl** command or the AIX System Management Interface Tool (SMIT) **smitty smt** command.[2] User and laboratory test experience strongly support leaving SMT enabled the default value of 4 for all SAP Sybase ASE applications.

A non-root user can identify whether SMT is enabled or disabled by running **lparstat -i**[3]. The root user can use the **smtctl** command without arguments to see the state of SMT, and use the same command with arguments to change or disable SMT mode.

SMT and online engines

When SMT is enabled, AIX is presented with two or four *logical* processors for each *core* in dedicated mode, and with two or four *logical* processors for each *virtual* processor in shared mode (refer to the "Shared-processor LPARs" section). Can you enable an ASE engine for each logical processor that is presented to AIX? You could, because ASE counts all these logical processors[4]. However, should you?

In the POWER5 and POWER5+ generation where the SMT benefit is in the order of 25% to 30%, the best results are obtained in the field by sticking to the standard ASE on AIX practice of allocating one engine per core. The standard practice prescribes that engines should be more than 60% busy on average. The SMT benefit is realized in that it is no longer required to leave a core for the kernel, as was done with traditional SMP. The kernel processes run on unused secondary SMT threads.

With POWER6 and POWER7, where the SMT benefit is as great as 50%, it has become a common place to allocate more online engines than there are cores or virtual processors. The two main use cases for this

---

[1] The term *POWER7 mode* is used because POWER7 might be run in POWER6 mode (SMT2 only) and if the AIX version is below 6.1 TL4, it can only be run in the POWER6 mode.

[2] SMIT, the AIX System Management Interface Tool, has a graphical Motif-based version SMIT and a pseudo-full-screen curses-based version "smitty" which alludes to legacy "tty". In practice, most administrators use smitty.

[3] In POWER5 or POWER6 the fourth line of **lparstat –i** output, type will be only **dedicated** or **shared** if SMT is disabled, and will add **SMT** if SMT is enabled. In POWER7, **dedicated SMT** or **shared SMT** means that two threads per core or virtual processor, and **dedicated SMT4** or **shared SMT4** means four threads per core or virtual processor.

[4] ASE 15.7 process mode and its predecessors count logical processors to determine how many engines to allow on line. ASE 15.7 threaded mode does not have any processor restriction on how many engine threads it will allow to run.

practice are scale-out and server consolidation. When scaling out ASE, the ability to add engines in excess of the number of cores (or virtual processors) without increased idling and degraded performance is proportionate to an increasing workload. Consider adding one engine at a time as the workload increases while keeping all engines greater than or equal to 60% busy. Conversely, when consolidating an existing ASE workload onto faster processors, the processor needed to handle the workload might be fractional, yet multiple ASE engines are required to handle user connections and disk I/O. In this case, you might configure two virtual processors and three or four ASE engines on a processor entitlement of .5. Refer to the "Shared-processor LPARs" section for further details.

If each SAP Sybase ASE 15.03-15.5 engine is (for all practical purposes if not strictly true[1]) a single OS process, why would legacy ASE benefit from SMT? Users can understand how the secondary threads benefit the kernel and peripheral processes on a dedicated ASE LPAR, but how would ASE itself make use of the second, third, or fourth thread? One benefit is in allowing runnable ASE engines to be dispatched immediately on a secondary thread, even if all primary threads are occupied. Another benefit is in permitting ASE tasks which have switched context into kernel mode to run concurrently with ASE tasks running in user mode. This concurrency is illustrated by ASE asynchronous I/O which is discussed under the "Storage alternatives" section.

There is a misconception that ASE prior to the 15.7 threaded kernel cannot take advantage of AIX/Power SMT. ASE process mode has been demonstrated to take full advantage of SMT for reasons given in the previous paragraph. While the engine typically monopolizes the first thread of each virtual or physical processor, the secondary (and in POWER7, the tertiary) threads allow runnable engines to be dispatched faster, and facilitate the handoff to the kernel of asynchronous I/O. As a key difference between threaded and process mode is the hand-off of asynchronous I/O to disk and network controller threads that are running in user space, one can observe the secondary SMT threads spending more time in the user mode when running the ASE 15.7 threaded kernel than when running in the process mode.

POWER7 SMT differs from that of POWER5 and POWER6 in three ways. First and most obvious, it provides four hardware threads as compared to two in the previous versions.

Second, the default algorithm according to which application threads are dispatched on hardware threads has been modified in POWER7 to strongly favor full utilization of the first thread of each core or logical processor before activating any others. It is not that the first thread is inherently more powerful than the others; rather, the fewer threads SMT threads are dispatched, the the more economical it is to manage the dispatch of user threads on processor threads.[2]

Third, POWER7 or POWER7+ SMT offers a quad_threaded mode, a dual_threaded mode, and a single threaded (SMT disabled) mode. The algorithm of the dual-threaded mode is the same as that of the quad-threaded mode, favoring the first thread until all cores' or virtual processors' first threads are maximized.

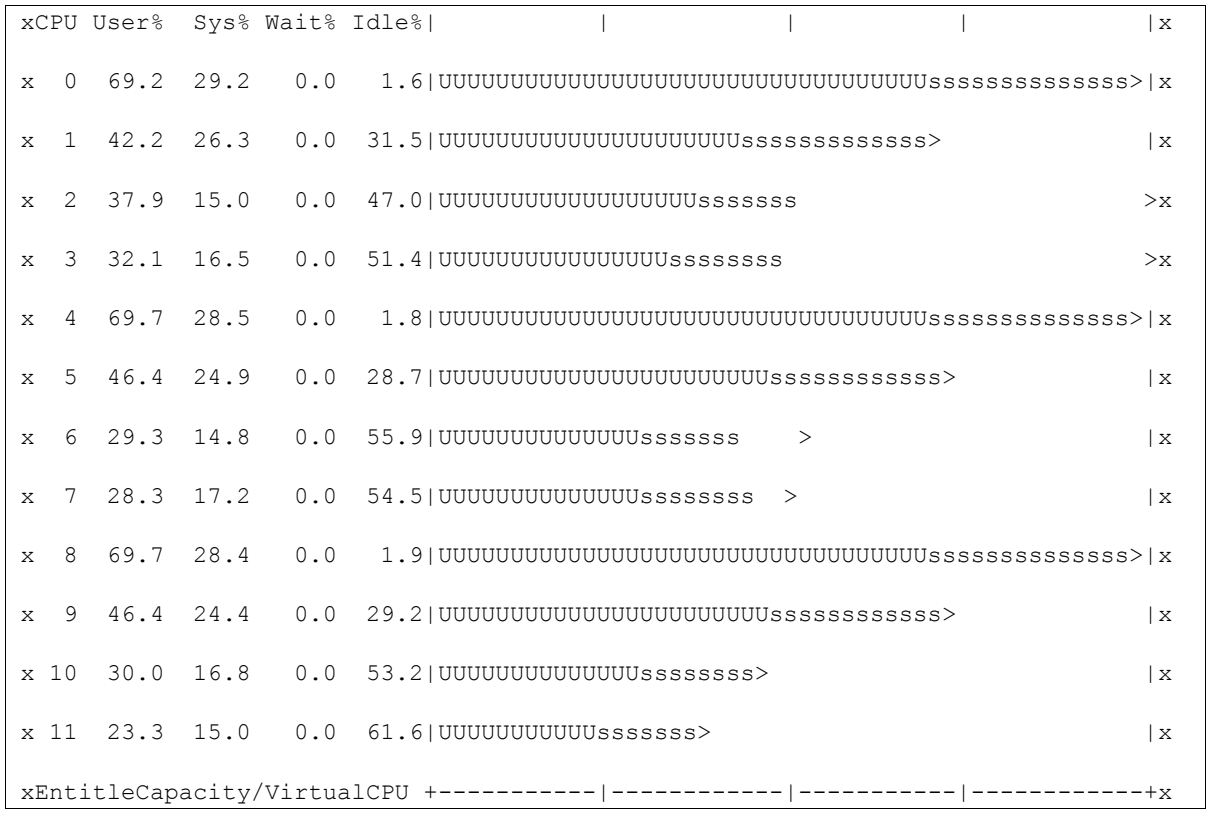
---

[1] The Sybase ASE 15.7 and earlier process mode engine actually has multiple threads: the engine thread, which uses the vast majority of processor cycles, a clock interrupt handler, and additional threads supporting client integration services, the job scheduler, and other configured functions.

[2] In AIX 6.1 TL 8 and AIX 7.1 TL2, an alternative algorithm for SMT processor dispatch has been introduced. Referred to as *scaled throughput*, this algorithm resembles the behavior of SMT in POWER6, in that all four threads of each processor are dispatched before the next processor is unfolded. Refer to Section 4.1.4 of *IBM Power Systems Performance Guide Implementing and Optimizing* at **ibm.com**/redbooks/redbooks/pdfs/sg248080.pdf.

The main purpose of the dual-threaded POWER7 SMT mode is to reduce the number of logical processors presented to applications, which count logical processors during operation, to determine parallel degree in query optimization or for application thread allocation. In those cases, allowing the application *see* four threads per core or virtual processor might be detrimental to performance. As ASE only counts logical processors at startup to determine the number of engines eligible to start (and only does so in process mode), POWER7 dual SMT mode does not benefit ASE. Field and lab results for general purpose transactional workloads show that the default quad-threaded SMT mode is preferable to other POWER7 SMT modes.[1] SMT4 is designed to adjust automatically to the distribution of workload. So if a workload is strongly single (or dual)-threaded, SMT operates in single- (or dual-) threaded mode.

Two interactive NMON illustrations of SAP Sybase ASE 15.7 threaded kernel workloads on SMT4 conclude the discussion of SMT. The first shows all four threads actively used, while the second, a more I/O-intensive workload, concentrates most of the work on the first two threads of each virtual processor. Even though the second, I/O-intensive workload appears to be using only the first two threads, it is shown empirically to degrade systematically when SMT mode is reduced from four to two and to single-threaded mode.



[1] It is sometimes reported that utilities (such as bulk copy) and maintenance jobs (such as update statistics) achieve better performance in single threaded mode. In POWER5 and POWER6, the administrative overhead of changing SMT mode for different workloads tends to outweigh any such benefit. In POWER7, the SMT algorithm heavily favors dispatching the first thread of each processor before using secondary threads, so batch and utility processes tend to exhibit single-threaded behavior in SMT4 mode.

```
xEC+   52.8   24.2    0.0   23.0|UUUUUUUUUUUUUUUUUUUUUUUUUUUUssssssssssssiiiiiiiiiii|x

x VP   47.6   21.9    0.0   20.7|UUUUUUUUUUUUUUUUUUUUUUUUUUssssssssssiiiiiiiiii------|x

xEC= 338.1%  VP=  90.2%      +--No Cap---|------------|-----------100% VP=3 CPU+x
```

Listing 6: NMON, a processor-intensive SAP Sybase ASE workload on POWER7+ utilizing all four threads of each processor

```
xCPU User%  Sys% Wait% Idle%|           |           |           |           |x

x  0  44.0  42.5   0.3  13.3|UUUUUUUUUUUUUUUUUUUUUUUUssssssssssssssssssss     >|x

x  1  54.2  40.8   0.2   4.8|UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUssssssssssssssssssss >|x

x  2   0.1   4.4   0.0  95.5|ss                 >                              |x

x  3   0.4   3.7   0.0  95.9|s     >                                           |x

x  4  51.4  37.9   0.3  10.4|UUUUUUUUUUUUUUUUUUUUUUUUUUUUssssssssssssssssss    > |x

x  5  67.1  31.5   0.4   1.0|UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUssssssssssssssss>|x

x  6   0.0   4.5   0.0  95.5|ss                        >                       |x

x  7   0.0   4.6   0.0  95.4|ss                       >                        |x

x  8  43.1  40.8   0.2  15.8|UUUUUUUUUUUUUUUUUUUUUUUUssssssssssssssssssss       >|x

x  9  61.5  35.8   0.4   2.3|UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUssssssssssssssss  >x

x 10   0.0   3.9   0.0  96.1|s                           >                     |x

x 11   5.3   6.2   0.5  88.1|UUsss                       >                      |x

xEntitleCapacity/VirtualCPU +-----------|------------|-----------|------------+x

xEC+   46.2  32.4   0.3  21.2|UUUUUUUUUUUUUUUUUUUUUUUUUUUUssssssssssssssssiiiiiiiiii|x

x VP   42.2  29.6   0.2  19.4|UUUUUUUUUUUUUUUUUUUUUUUUUUUUssssssssssssssiiiiiiiii-----|x

xEC= 342.9%  VP=  91.4%      +--No Cap---|------------|-----------100% VP=3 CPU+x
```

Listing 7: NMON, an I/O bound SAP Sybase ASE workload primarily using the first two threads of each processor

Shared-processor LPARs

Shared-processor LPAR, also known as IBM Micro-Partitioning® refers to the pooling of processor resources among a group of LPARs so that processing capacity can shift among LPARs, based on demand. This capability is part of IBM PowerVM® (formerly known as *Advanced Power Virtualization)* package which is available for purchase with POWER5, POWER6, and POWER7 processor-based architectures.

To design a shared-processor pool, the systems administrator needs to estimate the *minimum, desired, and maximum* processor requirements of each participating LPAR and its applications. The desired processor value is also referred to as *entitled capacity*. You can allocate these values (in hundredths of a

processor) in a range that is between one-tenth of a processor as a minimum(one-twentieth in the latest Power7+ models) and all the processors in the pool as a maximum. Next, you must determine whether to make the LPAR *capped* (limited to a desired, normal value) or *uncapped* (entitled to take over idle capacity of others), in terms of processing resources. Almost all installations of Power Systems Micro-Partitioning use uncapped mode to allow each LPAR to use capacity unused by its neighbors. The capped mode simulates a fractional dedicated processor, and is mainly used to manage charge-back for processor consumption.

Whether an LPAR is capped or uncapped, you also need to specify a minimum, desired, and maximum number of *virtual processors*. Virtual processors are integer values that represent time slices on the 10 millisecond dispatching cycle of each physical processor in the pool. Virtual processors appear to the LPAR OS as cores (as in the command lsdev –Cc processor), but are actually timeshares of pooled cores. You can alter the number of virtual processors, between the configured maximum and minimum, through DLPAR, but the number of virtual processors does not change (as does physical capacity) on the basis of utilization. The number of virtual processors that you configure can limit, in a manner similar to capped mode, the number of full processor's worth of resources an LPAR can ever acquire if all others are idle. If you configure two virtual processors, the LPAR can at most obtain two full processor's worth of capacity (even if four are idle); if you configure four virtual processors, an LPAR can obtain only four processors worth (even if six are idle), and so forth.

The final variable of *weight* is an integer that when compared to the weight of other LPARs, gives a relative priority in obtaining excess capacity. Weight only applies to *uncapped* mode. Assuming that there are four LPARs in a shared-processor pool and that each is given a weight of 128, then, when they all want excess capacity at the same time, they get the excess capacity in equal measure. However, if one is given a weight of 256, then, under the same circumstances, the favored LPAR gets 40% of the excess capacity and the others (each) get 20%. If there is no competition, each LPAR, regardless of weight, gets excess capacity up to its number of virtual processors.

The terms *minimum*, *desired* and *maximum* as applied to entitled capacity, virtual processor count, and memory, can be confusing. They represent a range within which the system administrator can increase or decrease these resources without a restart, not a range within which a running LPAR can acquire resources on demand[1]. These values thus play a role similar to **max online engines** and **max memory** in ASE. The running LPAR operates within a processor consumption range of zero, if idle, and the virtual processor count, if busy, with entitled capacity representing a guaranteed minimum.

Lparstat –i, illustrated at the beginning of the "Processor considerations" section, shows the minimum, desired, and maximum values for entitled capacity, virtual processor count, and memory, and also the **weight** variable.

The two greatest advantages of shared-processor LPARs are:

[1] *Minimum capacity* , which is usually set below *desired capacity*, also allows the LPAR to come on line in case processor resources are over-allocated across the server, or when a processor is damaged or missing.

- The ability to create LPARs with very small processor consumption without having to invest an entire physical processor to each of them. LPARs with fractional entitled capacity are commonly used to consolidate workloads from older servers.
- The ability to balance load between LPARs that are running applications with complementary load characteristics, for example, a daytime OLTP database that is replicated from a night-time batch database. You can get close to twice the utilization from each processor by pooling them in this fashion.

These advantages are inevitably paired with a minimal resource consumption that is associated with the scheduling logic (which is implemented at the microcode level, over and above the scheduling that takes place at the OS level). The ASE and AIX sizing and tuning recommendations for shared-processor LPARs, which follow, are intended to maximize the benefits of shared-processor LPARs while minimizing the extra resource consumption.

The advantages of the shared processor pool (most notably getting away from the practice of over-allocating resources on a dedicated server to accommodate peak workloads) also entail additional management responsibility. Multiple processor pools can be configured on POWER6 and POWER7 processor-based servers. To get the benefit of processor pooling, LPARs with complementary work cycles should be in the same pool, and LPARs with competing or conflicting workloads, which all spike at the same time, should be in different processor pools. Growth also needs to be managed more carefully. If several LPARs have growing workloads and are continually exceeding their entitled capacity and competing for excess processor capacity, performance might suffer.

The processor cycles that are allocated to an LPAR fluctuate based on load. If load drops to less than the desired processor level, excess capacity is returned to the pool and is made available to other LPARs. If load increases, capacity is first restored to the *desired* level for each shared-processor LPAR in the pool requiring more processor utilization. Capacity is then ramped up for LPARs requiring more than their entitlement to whatever the pool's excess capacity and the LPAR's weight permits. This allocation of excess capacity might even involve stealing excess from a neighbor LPAR of lesser weight.

All these transitions take time, measured in milliseconds, and are potentially noticeable to time-critical applications, such as stock-trading floors. Because the reallocation of processor resources necessarily trails demand, if an application's load shifts rapidly, then average allocation falls slightly, but measurably, short of entitlement. The same applies to excess capacity. A shared processor LPAR that is continuously hitting its limit of virtual processors will fall slightly short of the virtual processor count in terms of processor consumption (for example, 3.98 physical capacity on four virtual processors). For this reason, if your service level agreement (SLA) requires that you provide an ASE LPAR with the shared equivalent of a certain number of dedicated cores, you should allocate one more virtual processor than you would have dedicated cores (for example, five virtual processors to equate four dedicated cores).

Recommendations for ASE in a shared-processor LPAR environment

When setting minimum, desired, and maximum processor entitlement for a shared-processor LPAR, set the desired value to a normal utilization, not less than it. Therefore, if you use two dedicated processors at an average of 60% utilization and 90% peak utilization, consider setting the active virtual processor count to 3 and the desired capacity to 1.2 processors. This recommendation is made

because the design of shared-processor LPARs is to allocate all sharing LPARs with their desired capacity before allocating excess capacity to any LPAR. Thus, if an LPAR operates at lower than entitled capacity and then spikes up, it reaches its entitled capacity before another LPAR that operates at its desired capacity obtains any excess. Setting the desired capacity lower than true average demand is a false economy.

It is preferable to set maximum virtual processors = maximum online engines. In this way, the DBA and system administrator can coordinate when increasing capacity (by adding online engines and adding virtual processors together). You need to set active virtual processors to at least the next integer value that is above the desired (entitled) capacity. You should also set active virtual processors = number of engines at startup. In general, derive entitlement from your average load and derive your virtual processor count from your peak during a 24-hour or weekly life cycle, just as you would determine your ASE engine count.

It is best to keep the range between the desired capacity and the number of activated virtual processors to a minimum. Excess virtual processors have the effect of fragmenting the ASE engine's time slice at the shared-pool level. The ASE engine might get a 10 ms time slice at the OS level, but that might translate into four 2.5 ms slices at the shared-pool level if four virtual processors are queued for each core. IBM database specialists recommend a ratio of no more than 4:1 between active virtual processors and entitled capacity.

If ASE engines frequently exceed 80% busy, it is preferable from a performance standpoint to increase the entitled capacity, active virtual processors, and ASE online engines only at the time required, rather than over-allocating these resources before they are needed. As already observed, over-allocation of ASE engines might not only waste processor but actually degrade performance.

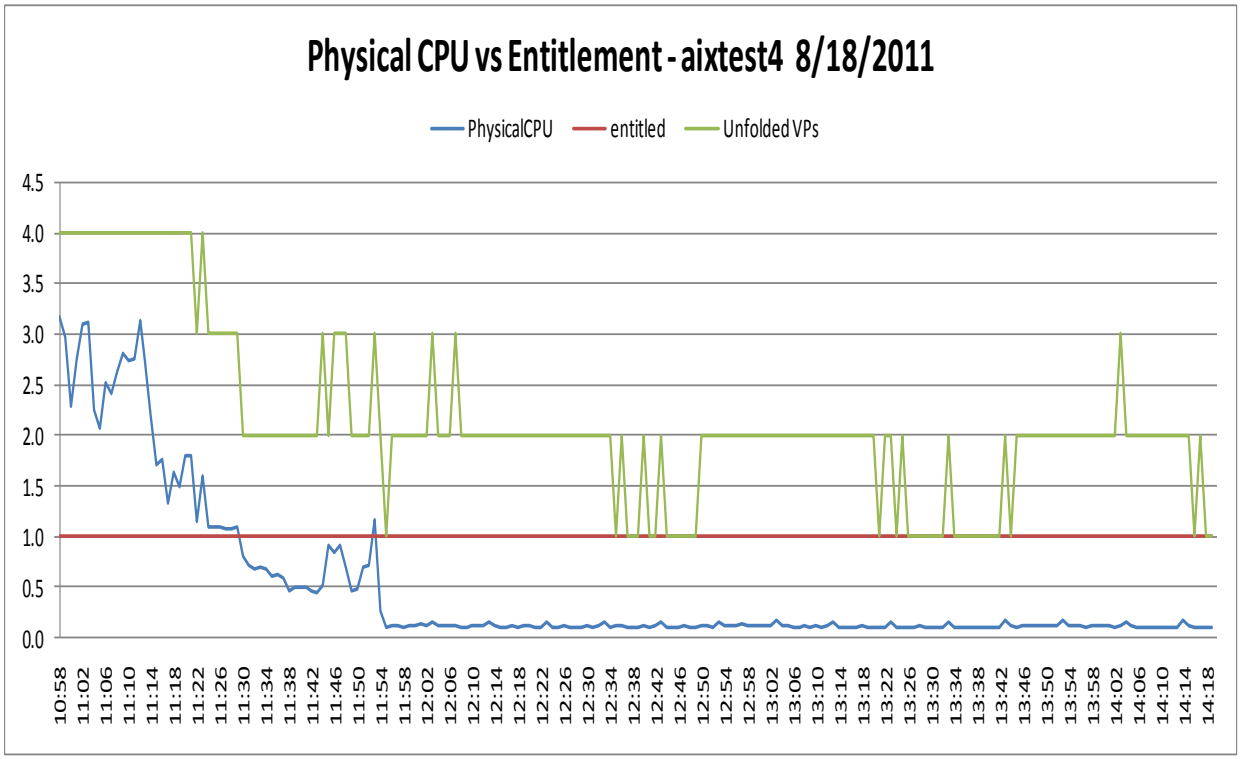


Figure 1: Illustration from NMON Analyzer of Virtual Processor Folding (blue range is processor utilization and green is unfolded virtual processors)

Virtual Processor Folding is a feature of shared process mode in POWER5, POWER6, and POWER7 processor-based systems in which virtual processors that are unused for a quantum of time are taken offline to reduce heat, energy consumption, and context switching while improving cache coherence. Folded virtual processors might increase the latency involved in obtaining processor resources when demand increases. Processor folding manifests itself through high utilization on a few logical processors and no activity on the rest. It is explicitly tracked in AIX utilities such as *Nmon* (refer to Figure 1). If processor folding results in uneven distribution of load across virtual processors, it is a best practice to reduce the number of active virtual processors. If you cannot do this because of widely fluctuating load, AIX support sometimes recommends tuning or disabling this feature through the AIX **schedo** option **vpm_fold_policy=0**, but to date there has been no indication that turning off virtual processor folding benefits typical SAP Sybase ASE transactional workloads.[*]

SMT applies to the shared-processor environment in an analogous manner to the dedicated-processor environment. The same performance benefits are enjoyed. Dual-threading (or quad-threading in POWER7 mode) is implemented on each virtual processor (instead of being implemented on a

[*] As with single threaded process mode, it has been reported that some ASE utility and maintenance jobs perform better with virtual processor folding turned off. If the number of virtual processors is equal to or greater than the number of work threads and Sybase engines running, turning off virtual processor folding amplifies the tendency of POWER7 to favor the first thread of each virtual processor, thus emulating single-threaded mode.

physical processor), and the OS sees two (or four) logical processors for each virtual processor. The standard sizing guideline of one engine per physical processor here becomes one engine per virtual processor, and the same guidelines (refer to the "SMT and online engines" section) apply to exceeding that starting point, for example, running six engines on four cores or six engines on four virtual processors.

It should be noted that the Virtual Processor Folding algorithm takes precedence over SMT. Especially in the POWER7 mode, the SMT round robin, wherein utilization of the first thread is maximized on each virtual processor before any second (third or fourth) thread is activated on any, operates within the confines of unfolded virtual processors. Thus, if only three virtual processors are unfolded, the SMT round-robin uses one or more threads on those three virtual processors until a fourth is unfolded and brought back online to meet processor demand.

Note: Binding SAP Sybase ASE engines to processors was once a fairly common practice. Currently, it is not recommended to bind an ASE process ID to a logical processor, even though it is technically possible to do so through the **bindprocessor** command. In the presence of SMT and virtual processor folding, the effect of ASE process binding is unpredictable.

Processor and memory affinity in POWER7 and POWER7+ processors

The POWER7 and POWER7+ hypervisor attempts to co-locate processor and memory resources assigned to an LPAR wherever possible. However, on large IBM Power® servers with many LPARs, particularly those with multiple enclosures, central processor complexes (CPCs) or processor *books*, there is the possibility of resource fragmentation. This principally occurs when memory and processor resources are added to or removed from LPARs after the LPARs on the frame have been created. Some latency might be experienced by very sensitive applications in accessing the processor and memory resources on distant sockets (analogous to nonuniform memory access (NUMA)). In rare instances, processor resource fragmentation can increase ASE spinlock contention owing to latency in synchronizing L3 cache information across a large system. AIX and the POWER7/POWER7+ processors provide a number of techniques to reduce fragmentation and improve performance. By default, AIX organizes the resources of an LPAR into Scheduler Resource Allocation Domains (SRADs). These SRADs are designated as *local* and *far* on small systems, *local*, *near* and *far* on larger systems. The **lssrad -av** command (root user) displays the distribution of resources. The **topas** command with the **M** option also displays SRAD information and is available to any user:

```
# lssrad -av
REF1  SRAD      MEM      CPU
0
        0       15662.56  0-15
1
        1       15857.19  16-31
```

This example of **lssrad**, from the *Power Systems Performance Guide* from IBM Redbook® sg248080 starts with a system-determined *logical reference point* within which there are two resource domains, or

SRADs, assigned to the LPAR. The first (local) domain has 15.6 GB of memory and logical processors 0 to15 (4 virtual processors) and the second (far) domain has almost identical resources.

This example from **topas** illustrates a large SAP Sybase ASE LPAR with significant resource fragmentation. There are six SRADs distributed over three different logical reference points corresponding to Power 795 processor books. There are approximately 30 other LPARs on this large system, with a high degree of interleaving of memory and processor resources among LPARs.

```
Topas Monitor for host:    mwcedb1p    Interval:   2    Tue Jul 16 16:33:47 2013
================================================================================
REF1     SRAD  TOTALMEM  INUSE    FREE    FILECACHE  HOMETHRDS  CPUS
--------------------------------------------------------------------------------
   0      0     38.7G    38.6G    160.2    6732.7       357      0-3 8-11 16-19 28-31 44-47
   1      1     33.3G    33.2G    134.0    5741.9       359      4-7 12-15 24-27 40-43
          5    5727.0    5694.9    32.1     877.5        186      52-55
   2      3     11.4G    11.4G     45.9    1875.5       289      36-39 60-63
          2     18.7G    18.6G     76.9    2322.0       289      20-23 32-35 56-59
          4    8964.0    8921.9    42.1    1401.6       271      48-51
```

This example from **topas** illustrates the User Acceptance Testing (UAT) instance of the production ASE server in the first example. In this case, the resources are concentrated in a single SRAD, and are in fact contained on two 8-core processor modules, an ideal layout not easily replicated on large shared frames.

```
Topas Monitor for host:    swcedb1u    Interval:   2    Tue Jul 16 16:34:00 2013
================================================================================
REF1     SRAD  TOTALMEM  INUSE    FREE    FILECACHE  HOMETHRDS  CPUS
--------------------------------------------------------------------------------
   0      0     83.5G    81.5G    1975.7   6927.5      1553      0-63
```

AIX has algorithms for dispatching workload on the nearest set of resources possible, before using *far* resources. There are also AIX techniques to force a workload to use a particular subset of the LPAR's processor and memory resources, such as its *home* socket and memory. This is called an **RSET**. There are both automated and manual techniques for defragmenting resources – the Dynamic System Optimizer implements a manual process by which the hypervisor *forgets* its LPAR resource allocations, and these allocations are recovered in order from the Hardware Management Console, defragmenting those allocations as much as possible. Systems Administrators are advised to be familiar with the SRAD concept and its monitoring.

The performance penalty in over-allocating engines (applies to all releases)

SAP Sybase ASE on Power Systems has been shown to be indefinitely scalable as workload increases. However, most lab benchmarks and client workloads of a fixed volume have a point of diminishing returns at which adding additional engines, with or without additional processor resources, results in degraded

throughput. This is the point at which the processor and time consumption involved in idling, spinlock contention, and inter-engine communication exceed the benefit of the added engines. An analogy could be drawn to the mix of air and fuel maintained by a carburetor or fuel injector: too many engines for the workload is similar to an air-fuel mixture that is too rich. Most clients do not have the luxury of *brute force* testing of a workload to establish the right engine count, as you would, for example, with the TPC-C benchmark in a lab. While it is relatively easy to tell when there are too few engines, it is harder to tell, particularly in a newly upgraded or migrated environment, whether the engine count is too high. In ASE 15.5 and earlier releases, you can take note of spikes of very high processor busy plus high spinlock, or low processor busy with high idle looping as symptoms of too many engines. ASE 15.7 sp_sysmon in threaded mode has a number of new measurements that contribute to the ability to evaluate engine count without already knowing what the *sweet spot* is. These were illustrated earlier in the “Processor considerations” section.

Over and above excessive idling and spinlock contention, there are two other scenarios in which engine over-allocation can degrade performance. One of them is processor saturation. Even though SMT4 provides 50% higher throughput than a single-threaded core, there might not be sufficient reserve processor capacity to run six engines, for example, on a 4-core LPAR, particularly where heavy physical I/O is also in play. This is illustrated in Appendix B where an optimal engine allocation on a 4-core system is typically no more than five. A second case is harder to decipher: this is where excess processor capacity appears available to ASE, but the application will not scale up to use as many engines as there is available processor resources including SMT. Neither engine idling nor spinlock contention explain an overall reduction of throughput measured by committed transactions, lock requests, and cache searches. One reason for such a scalability limit is the speed of disk I/O. Adding engines can improve I/O processing up to a point, but beyond that point, adding engines is not only useless but also degrades performance by increasing the time and resources consumed by idle yields and thread rescheduling. Network I/O may also reach a saturation point at which the client or application server cannot ingest returning data sets any faster. As with disk I/O latency, adding ASE engines in this scenario may actually degrade throughput.

The AIXTHREAD_SCOPE environment variable in AIX

To increase stability, you need to set the AIXTHREAD_SCOPE environment variable to system (S). This AIX environment variable enforces a 1:1 ratio between user and kernel threads when a process enters the kernel mode. This recommendation applies to ASE 15.7 and earlier ASE releases. Native threads are used by the ASE 15.7 threaded kernel. All ASE kernels can use native threads for RTDS and Lightweight Directory Access Protocol (LDAP). Native threads are also used when ASE reads the interfaces file to do a backup, perform a Remote Procedure Call (RPC), or start the SAP Sybase Extended Procedure (XP) server. Set **AIXTHREAD_SCOPE=S** in the ASE **RUN_server** file before the ASE **dataserver** command (shown in blue text in Listing 8). If you have other SAP Sybase components on the LPAR such as a replication server that can also benefit from AIXTHREAD_SCOPE=S, you can set it in SYBASE.sh or SYBASE.csh.

```
#!/bin/sh

#

# ASE page size (KB):   4k

# Master device path:   /sybase/devices/SYBASE.master

# Error log path:       /sybase/15.0/ASE-15_0/install/SYBASE.log

# Configuration file path:      /sybase/15.0/ASE-15_0/SYBASE.cfg

# Directory for shared memory files:    /sybase/15.0/ASE-15_0

# Adaptive Server name: SYBASE

#


AIXTHREAD_SCOPE=S

export AIXTHREAD_SCOPE


/sybase/15.0/ASE-15_0/bin/dataserver \

-d/sybase/devices/SYBASE.master \

-e/sybase/15.0/ASE-15_0/install/SYBASE.log \

-c/sybase/15.0/ASE-15_0/SYBASE.cfg \

-M/sybase/15.0/ASE-15_0 \

-sSYBASE \
```

Listing 8: Setting the ASE thread scope in the ASE run file

Storage alternatives

There are several alternatives for storing data when using SAP Sybase ASE on the AIX operating system. These choices include raw devices, ordinary buffered file-system devices, and file systems accessed through direct I/O, concurrent I/O or Veritas Quick I/O. Although ASE does not support concurrent I/O as a device attribute, concurrent I/O can be implemented through an AIX file system mount option.

General storage recommendations for AIX

The AIX storage hierarchy includes:

- The *hdisk* (or in an EMC Powerpath environment, the *hdiskpower* device). This is a logical representation of the logical unit number (LUN) that is presented to the server.
- The *volume group*, which is a collection of hdisks.
- The *logical volume*, which you create across one or more hdisks within a single volume group, and which you can use directly, either as a raw partition or under a file system.

The file system is journaled in AIX, meaning that it has a log of all transactions, similar to a database. JFS is the legacy AIX 32-bit journaled file system, little used at this time, and JFS2 is the enhanced 64-bit version.

> **Note:** AIX 5L, AIX 6.1, and AIX7.1 fully support Veritas Foundation Suite (including VxFS: refer to the "Quick I/O, direct I/O and concurrent I/O: File-system access without OS buffering" section) which can be used concurrently with AIX Logical Volume Manager (LVM).

AIX database performance specialists recommend the provision of relatively small, relatively numerous hdisks that are presented to the OS, regardless of how these LUNs are striped on the storage area network (SAN) side. The advantage of many small LUNs is that there are more I/O buffers at the physical level. It is further recommended that you use a technique called **maximum interpolicy** when creating logical volumes, which has the effect of round-robin placement of partitions across available disks in the volume group. This technique is also known as *poor man's striping* to distinguish it from true OS-level striping. The following **mklv** command illustrates **maximum interpolicy:**

```
mklv –ex –y mylv01 –t raw sybasevg 256 hdisk3 hdisk4 hdisk5 hdisk6 hdisk7
```

This command and parameter sequence creates a raw logical volume (mylv01) on the sybasevg volume group. The **-ex** parameter prompts to use a round-robin manner for spreading the 256 partitions across all five disks that are listed. This is a form of OS striping without the additional processor and memory usage of true OS striping and, unlike true striping, you can extend it to new disks as you add them to the volume group using the **reorgvg** command. When using **maximum interpolicy** with data servers, it is further recommended that for successive **mklv** commands to create multiple logical volumes, the order of hdisks be randomized. For example, you can extend the previous example to create additional logical volumes with different physical partition distributions.

```
mklv –ex –y mylv02 –t raw sybasevg 256 hdisk7 hdisk6 hdisk5 hdisk4 hdisk3

mklv –ex –y mylv03 –t raw sybasevg 256 hdisk4 hdisk3 hdisk7 hdisk6 hdisk5……..
```

Randomizing the order of hdisks in the **mklv** commands helps to avoid having last page writes of several ASE devices, such as log devices wind up hitting the same hdisk and defeating the purpose of **maximum interpolicy.**

Note: When creating a volume group to support **maximum interpolicy**, a small partition size must be used. Partition size larger than 64 MB tends to defeat the purpose of partition spreading because the partitions are too big to distribute widely.

Note: Most storage and database specialists currently recommend **striping across striping**, that is, striping at the OS level even if there is RAID5 or another striping scheme on the SAN. As already noted, the purpose is to avoid hotspots in the OS I/O stack. Two qualifications are in order. If the LUNs allocated to the host are pooled on the storage backend, there might be little benefit in maximum interpolicy striping at the OS level. And, if a tiered storage solution which promotes hot objects to solid state storage is providing the LUNs, then the *storage tiering size* needs to be taken account in relation to the size of each ASE logical volume. If the portion of the logical volume on each LUN is smaller than the storage tiering size, and busy logical volumes are intermixed with less busy ones by the striping scheme, this might impede the recognition and promotion of hot objects by the storage tiering algorithm[*]. Consult your SAN support team and understand your SAN configuration before implementing maximum interpolicy striping. A data server can only go as fast as its storage. Slow read and write service times not only degrade throughput, but also make it more difficult to tune ASE correctly, as slow I/O skews other indicators used to tune ASE performance. I/O latency as experienced by ASE, reported in the Monitoring and Diagnostic Access table monIOQueue, is the sum of service time, queue wait, and other bottlenecks such as data hotspots introduced at the ASE device and logical volume level.

AIX supports most storage manufacturers, most of which have proprietary drivers. One of the most-frequently encountered performance degradations in disk I/O occurs when the correct drivers for the disks are not installed or are improperly installed. The storage still works but in a degraded fashion. The most conspicuous sign of this problem is that AIX does not recognize the maker of the drive but refers to it in **lsdev –Cc disk** or in **lscfg –vl hdisk<>** as *other fiber SCSI disk*. Often this forces a queue depth on the hdisk of **1** or **3** and you cannot change it.

It is very important to ask your storage vendor for the recommended queue depth for each model of LUN presented to a system: queue depth is the number of parallel I/O operations that you can queue at the hdisk level. In general, storage vendors recommend values for queue depth based on the number of adapters and disks and other factors. The AIX **lsattr –El hdisk<>** command gives the current value of queue depth. If that value is very low, 1, 2 or 3, this might mean that:

- As just noted, the vendor-supplied hdisk drivers are not installed or not properly configured
- The hdisk is an internal serial-attached SCSI (SAS) drive, in which case. the queue depth is fixed at 3

[*] According to EMC support, VMAX storage promotes hot devices on a per-track, not per-LUN basis and so maximum interpolicy striping is recommended as being compatible with the VMAX tiered storage implementation.

- The hdisk is a virtual Small Computer System Interface (SCSI) disk (refer to the "Virtual I/O in AIX 5L to AIX 7.1 with POWER5, POWER6, and POWER7" section) and the queue depth has not been set to match that on the Virtual I/O Server (VIOS)
- The hdisk is a Hitachi Data Systems device, which tends to default to a queue depth of 2 on AIX and whose value usually needs to be increased to 8 or more

A consistent, large (three digits or more) value for queue overflow (*serv qfull* for an hdisk in the AIX iostat –DRTl report) might indicate the need to increase queue depth. A high double-digit or greater value for average queue wait, also from iostat –DRTl, might also indicate the need to increase queue depth.

Because I/O service time at the hdisk level represents the time it takes for a read to return or a write to be acknowledged, it cannot really be tuned at the OS level, but is really a matter of SAN optimization. On the other hand, queue depth can be tuned, and in being tuned, can reduce queue wait time and queue overflow. The point of diminishing return, particularly for writes, is the point at which the queue is large enough to push out more writes in parallel than the SAN switch, the storage-side fibre adapter or the storage controller can handle. That then increases service time. A good general rule for tuning queue_depth is that you can increase queue_depth until I/O service times start exceeding an average of 10 ms for random reads and exceeding an average of 2.5 ms for writes.[1] Notice that the use of write cache introduces cache effects, so you can notice good I/O service times until the cache fills up. The use of a tool such as **ndisk**[2] is appropriate for simulating load on your disks and setting the queue_depth. Disk subsystems with read cache can also affect your results. So, be sure to run your test against non-cached data.

Pbufs are pinned buffers that manage I/O requests to the AIX hdisks. If the number of *pending disk I/Os blocked with no pbuf* in **vmstat -v** is large or grows between reboots, it might be required to increase the AIX **ioo** tunable **pv_min_pbuf**, which specifies the minimum number of pbufs available to each physical volume (hdisk). This is a global value that applies to all volume groups (VGs) on the system and applies when you add hdisks to any VG. You can also use the AIX **lvmo** command to tell whether a particular VG is experiencing pbuf overflow and to set a different pbuf value for that VG.

The host bus adapter (HBA) in AIX also has a queue depth. It is called **num_cmd_elems** and the value can be seen by running **lsattr –El fcs<>** for each HBA port. The **fcstat –e** command reports adapter traffic since the last restart and adapter error conditions including **No command Resource Count** which is a form of HBA queue overflow. If **No command Resource Count** is large and persistent, it is recommended to increase the value of **num_cmd_elems** above the default of 200.

AIX hdisks have a value **max_transfer** representing the maximum size of an I/O that can be processed as a unit. The default is 0x40000 or 256 KB. The Fibre Channel (FC) adapter (physical or NPIV virtual) has a corresponding value **max_xfer_size**. It is a general recommendation for database servers to increase both values to 0x100000 or 1 MB.

---

1 See Dan Braden's primer on queue depth tuning at **bm.com**/support/techdocs/atsmastr.nsf/WebIndex/TD105745

2 **ndisk** is part of the nstress package that is available at **ibm.com**/developerworks/wikis/display/WikiPtype/nstress

Asynchronous I/O methods

Asynchronous I/O allows ASE to perform other work while a disk I/O is pending. SAP Sybase ASE implements the asynchronous method for all disk I/O, both in ASE 15.7 threaded mode and in the process kernel. While the task issuing the **aio_read()** or **aio_write()** call sleeps on completion of the I/O, the engine is released to execute other tasks.

Some facility must manage the pending I/O to allow the application to do other work while the disk I/O method is performed. In the case of the AIX operating system, the application places I/O requests in a queue. In general, asynchronous I/O servers (the OS-process asynchronous I/O server) pick up the I/O requests from the queue and manage them until completion, but there are bypasses referred to as *fastpaths* or *kernelized asynchronous I/O* for raw devices and for concurrent I/O. In the ASE process mode, the AIX operating system notifies the ASE engine of an I/O completion through a flag in the ASE memory space, an optimization that avoids system calls for polling. In the ASE 15.7 threaded mode, the disk and network controllers poll IOCP for I/O completion through `GetMultipleCompletionStatus()`. The I/O completion mechanism in each case is the same for file system I/O as for asynchronous I/O methods on raw devices.

Tuning asynchronous I/O in ASE

It is worth repeating that asynchronous I/O servers require processor cycles and, therefore, you must consider this for proper processor sizing.

Even when using raw devices and AIX fastpath, ASE disk I/O is asynchronous, that is, the engine is freed to run other tasks while the I/O is pending. For ASE data servers with heavy disk I/O, the default values for the ASE configuration parameters governing asynchronous I/O often need to be adjusted. The most important ASE configuration parameter relating to asynchronous I/O is **disk I/O structures**, which has a default of 256. Disk I/O structures are shared across all configured engines (**max online engines)** plus a global pool. If an engine uses up its disk I/O structures, it must obtain a spinlock to get additional structures from the global pool. To avoid the time and processor resource consumed in negotiating a spinlock, it is important for disk I/O structures to be adequately sized.

If the **sp_sysmon** value **Max Outstanding I/Os** for any engine is close to or equal to the value of **disk I/O structures,** then **disk I/O structures** is capping the number of parallel I/O that an engine can sustain, and **disk I/O structures** must be increased.

A rule of thumb for increasing **disk I/O structures** is to take the largest value of Max Outstanding I/Os for a single engine from sp_sysmon over a period of time, and multiply that number by (**max online engines** +1) to get the new value of disk i/o structures. You need to use **max online engines** for this calculation even if the number of online engines is lower. In practice, this might result in an unnecessarily high value; a high value of **Max Outstanding I/Os** can also reflect slow physical disk I/O. So, it is preferable to increase **disk I/O structures** in stages (for example 2048, then 4096, then 8192) and see if **Max Outstanding I/Os** continues to hit the limit of **disk I/O structures.** The number of disk I/O structures that an engine can access is in turn limited by the configuration parameter **max async I/Os per engine**. This parameter has a default of 4096 (2 GB in ASE 15.7). So pre-ASE 15.7 **max async I/Os per engine** may have to be increased in order to accommodate a large value of **disk I/O structures**. On AIX 5L (refer to the “Tuning asynchronous I/O in AIX 5L only” section), ASE

consults the OS tunable **aioo maxreqs** to determine an upper limit for **max async i/os per engine**. The system administrator thus might have to increase **aioo_maxreqs** before the DBA can increase **max async i/os per engine**. This constraint does not apply to ASE 15 running on AIX 6.1 or AIX 7.1. It is recommended to set **max async i/os per engine** to its maximum value, 2 GB, in ASE 15.5 and earlier releases. This is the default value in ASE 15.7.

If sp_sysmon shows any **I/Os Delayed By Disk I/O Structures,** then the value of **disk I/O structures** must be increased to avoid performance issues. If sp_sysmon shows any **I/Os Delayed By Engine Config Limit,** then consider increasing the value of **max async I/Os per engine.** If sp_sysmon shows any **I/Os Delayed By Operating System Limit,** then consider increasing the value of **aioo maxreqs** (AIX 5L only)**.** There are other indications of disk I/O structure deficiency, particularly in sp_sysmon's asynchronous prefetch statistics, which are beyond the scope of this paper.

Tuning asynchronous I/O in AIX 5L only

For all file system-based devices, it is a good practice in AIX 5L (AIX 5.2, AIX 5.3, both out of support) to increase certain configuration options from their respective default settings:

- **`Minimum number of servers`**: 30
- **`Maximum number of servers per cpu`**: 300
- **`Maximum number of REQUESTS`**: 8192
- **`Server Priority`**: 39
- **`STATE to be configured at system restart`**: available
- **`State of fast path`**: enable

You can view and set these options through the SMIT **smitty aio** or the command line **aioo** command. This command requires root privileges to set the parameters but not to view them. The two latter configuration options –**STATE to be configured at system restart** and **State of fast path--** are also necessary for asynchronous I/O methods to raw devices.

You can display the currently running asynchronous I/O) servers with the AIX **`pstat`** command or the standard AIX **`ps`** command: **`pstat -a | fgrep aio`** or **`ps -k | fgrep aio`**

Notes:

- Unless you have enabled and used fast path (best practice), asynchronous I/O operations on raw devices also require asynchronous I/O servers, so the same guidelines regarding the number of asynchronous I/O servers applies.
- Asynchronous I/O servers are kernel processes. The `-k` parameter of the `ps` command is required to display them.
- Although the `pstat` command does not require root privileges, the resources that the command uses might require them.
- Refer to the AIX OS documentation for a description of POSIX and legacy asynchronous I/O servers. The ASE engine uses legacy asynchronous I/O servers, that is, the AIX-optimized facility rather than the platform-independent POSIX facility. Therefore, the correct reference is to asynchronous I/O servers (and not **`POSIX_aioserver`**).

The **aioo maxreqs** tunable, which corresponds to **Maximum Number of Requests** in SMIT, sets an upper limit on the number of asynchronous I/O requests that can be queued in parallel on the LPAR. Setting the queue length to 8192 is sufficient for most environments, but you might want to increase this number for high-end systems. The output from the **sp_sysmon** command indicates the need to increase **maxreqs** by reporting that the OS limit is delaying the I/O method. Similarly, the **aioo maxservers** tunable, which corresponds to maximum number of servers per processor in SMIT, sets an upper limit on the number of asynchronous I/O kernel threads that can be started. If maxservers is too low, AIX will start all the asynchronous I/O threads permitted, but will exhibit user processes blocked on I/O in vmstat.

Tuning asynchronous I/O in AIX 6.1 and AIX 7.1

The asynchronous I/O subsystem is extensively redesigned in AIX 6.1 and AIX 7.1. The asynchronous I/O kernel extension is available by default, but is only activated when an application makes an asynchronous I/O request. The tunables for *minimum servers*, *maximum servers*, *maximum number of requests* and *server priority* are now in the **ioo** command-argument set. POSIX and legacy settings are differentiated. Fastpath for raw devices (**aio_fastpath**) and file-system concurrent I/O fastpath (**aio_fs_fastpath**) are implemented as separate **ioo** values and are enabled by default. In other words, no special tuning of asynchronous I/O is required of AIX 6.1 or 7.1 for ASE at the outset. However, if you are using file systems for ASE devices, it is still important to monitor the number of asynchronous I/O kernel processes in use through **pstat** or **ps -k** as described earlier. Another tool for monitoring asynchronous I/O usage is **iostat –A,** which reports both raw and concurrent I/O fastpath usage, both of which bypass the asynchronous I/O kernel processes.

Raw devices

Using raw asynchronous disk I/O on character devices offers the best performance alternative and requires minimal AIX configuration. In AIX, ASE raw devices can be configured on raw logical volumes or on hdisks. The performance improvement results from the short code path and lack of double buffering (refer to the information on file I/O in the “File-system devices” section).

If you are using AIX 5L, use **smitty** to enable asynchronous I/O, regardless of whether you use raw devices or any file system. In AIX 6.1 and AIX 7, the asynchronous I/O subsystem is available by default and activated on demand: appropriate settings for using raw devices (and most file system implementations) are available by default. One asynchronous I/O tunable value *fastpath* (enabled by default) directly benefits raw I/O by further shortening the code path (by bypassing the asynchronous I/O server entirely). For AIX logical volumes and hdisks, the character devices have an *r* prefix (for example, `/dev/rmy_lv00` instead of `/dev/my_lv00, rhdisk25` instead of `hdisk25`). These raw character devices are given, in quotations, as the **physnames** in the ASE **disk_init** command.

The Sybase user generally owns raw devices. If the user who brings up the ASE engine does not have read and write privileges on the raw device, the ASE engine does not start, or it starts with one or more devices and databases offline. Certain AIX maintenance operations require that the root user touch the character-special file, which identifies a raw device (for example, `/dev/rmy_lv00`**)**. This changes ownership back to root. The system administrator must take care to change the raw devices’ owner back to the Sybase user after performing maintenance, such as increasing the size of a logical volume, moving

the logical volume to another physical device, exporting and importing a volume group containing ASE devices, and synchronizing the Object Data Manager (ODM). In IBM PowerHA® SystemMirror for AIX (formerly known as IBM HACMP™) clusters, it is best to include commands for reowning the Sybase devices in the start script, as these commands must precede the command that starts the ASE engine. The only perceived drawback to using raw devices is management coordination: unlike a file system, where extra space can be allocated in which the DBA can create new files for new ASE devices as needed, with raw devices, the system administrator must create a new raw logical volume or allocate a new hdisk and set its ownership before the DBA can use it.

File-system devices

Reading and writing to files is quite simple. Operating systems provide significant services to help developer productivity in coding these tasks. Using these services reliably and efficiently is harder. This section describes the file system options available to ASE 15 and their best practices.

Buffered file system I/O and dsync

The application, such as ASE, opens the file one time, then uses system calls to read from or write to the file. In the case of multiple processes that write to the same file, as with multiple SAP Sybase ASE engines, you must coordinate access.

When using the AIX **write()** system call, writes to files are buffered by default. The data that is to be written is cached in the AIX file system buffer cache and is actually written to disk at some later point in time. Applications can request that the operating system flush the written (pending) data to disk and acknowledge the successful write only after the disk I/O method has actually completed. To make this request, the AIX operating system provides flags that applications can set when opening the file. The **dsync** attribute is the ASE default device setting that provides durable writes; it corresponds to the O_DSYNC flag for the AIX **open()** system call. When the **dsync** device attribute is set, the adaptive server opens a database device file using the UNIX O_DSYNC flag, which ensures that writes to the device file occur directly to the physical storage media. ASE defaults to requesting safe I/O writes from the operating system.

Quick I/O, direct I/O and concurrent I/O: File-system access without OS buffering

The tradeoff between buffered file system I/O with **dsync** and unbuffered methods – raw devices, Quick I/O, direct I/O, and concurrent I/O-- is the opportunity for aggressive read-ahead in file cache in the former, compared to better overall performance in the latter. If your application does intensive sequential reads, then buffered file I/O with **dsync** is a viable option. Otherwise, if there is a strong preference against using raw devices, ASE supports three techniques of unbuffered file access which avoid the latency associated with buffered file access combined with **dsync** :

- **Quick I/O method:** The proprietary Veritas I/O method that is available on the Veritas File System (VxFS) and is packaged with Veritas Foundation Suite Versions 5.1 and higher versions.

- **Direct I/O method:** A standard multiplatform I/O method that is available on JFS and JFS2 with SAP Sybase ASE 15.0 and higher versions. JFS2, among its other advantages, explicitly supports direct I/O.
- **Concurrent I/O method:** A superset of direct I/O, concurrent I/O assumes that an application, such as a database server, has methods to prevent concurrent writes to the same page of data. With this assurance, concurrent I/O bypasses inode locking and permits multiple concurrent writes to the same file. JFS2 explicitly supports concurrent I/O using a file mount option or else an O_CIO or O_CIOR open call.

All three techniques use optimized code paths and avoid file-system buffer cache. This is intended to provide better performance and guarantees durable writes. Although results always vary between applications and workloads, in-house tests show that on AIX 6.1 and AIX 7.1 with ASE 15.5 and 15.7, concurrent I/O file systems perform comparably to raw character devices and that both concurrent I/O and raw I/O might perform up to 30% better than direct I/O or **dsync**. The quick I/O method on VxFS is essentially raw I/O disguised as file I/O and thus provides performance benefits comparable to raw devices.

What is the difference between **dsync** and direct I/O or quick I/O? First you need to review **dsync** By default, the OS buffers the writes and reads to and from a file system device (that is, a file). Hence, the log pages (with log records from a committed transaction) might not be on the disk, but rather in the file system buffer cache. To avoid the possible loss of in-memory data in a server crash, ASE by default adds the O_DSYNC flag to the open() call. This tells the operating system not to acknowledge writes until the write is on the actual disk. However, the file system buffer cache, inode lock and their surrounding logic are still in play.

Quick I/O methods associate a VxFS namespace with a character-mode device driver. When accessing the file, a raw-device open call is issued. With quick I/O installed, there is no specific ASE configuration requirement, as ASE automatically uses the quick I/O for SAP Sybase method on the VxFS (refer to the appropriate Veritas manuals for details).

Direct I/O methods open a file to read or write with the O_DIRECT flag. This flag instructs the operating system to bypass the OS file system buffer cache for both reads and writes, but direct I/O still must negotiate the inode lock that permits only one write to a given file at a given time.

To use direct I/O with ASE 15, use the **directio** argument with the **`disk init`** command or the **`sp_deviceattr`** system procedure to set or change the device's existing status.

```
sp_deviceattr '<device name>', 'directio', 'true'
```

Then mount the JFS2 file system without any options.

Concurrent I/O is a superset of direct I/O in bypassing file caching. But in addition, concurrent I/O assumes that the application, such as a database server, implements the file page locking. Thus, it permits multiple concurrent writes to the same file and bypasses inode locking.

Concurrent I/O is implicitly supported with ASE 15 in the form of a file system mount option, **-o cio**. This mount option can be set when creating the file system, or else on a session basis. Concurrent I/O is not yet implemented in ASE through a device attribute which would implement O_CIO, but there is a change

request CR 625429 to achieve this. When using the **cio** file mount, ASE device attributes **dsync** and **directio** should be disabled.

Notes:

- The SAP Sybase ASE device attribute **directio** is available in version 15.0 or higher.
- A SAP Sybase ASE device attribute supporting concurrent I/O is in development (at the time of publishing this paper) through CR 625429.
- In ASE, the direct I/O method and the ASE **dsync** attribute are mutually exclusive. When using `sp_deviceattr`, you must disable the **dsync** attribute, if set, before enabling direct I/O.
- If you are using an ASE 2KB page size, when creating the file system as an ASE direct I/O or concurrent I/O device, you must explicitly set the file-block size (**agblksize**) to 2 KB. Otherwise, the AIX OS silently demotes the I/O method from a direct I/O method to a regular file system I/O method and repeats the operation, with a significant performance penalty. As long as the requested I/O size is a multiple of the file-block size, the direct I/O method works as designed. For example, a 16 KB I/O request on a 4 KB block-size file system uses the direct I/O method. But, if you do not specify an **agblksize** of 2 KB for a 2 KB ASE data device, direct I/O will degrade.
- Direct I/O is implemented through asynchronous I/O kernel procedures, so you need to follow the recommendations for configuring asynchronous I/O, such as *aioservers* and other parameters that are discussed in this chapter, as if you were using ordinary buffered file I/O methods. In general, the AIX 6.1 and AIX 7.1 defaults are satisfactory.
- Concurrent I/O is implemented similar to raw character I/O using a *fastpath* mechanism, **aio_fs_fastpath**, which bypasses the OS asynchronous I/O kernel threads. This **aio_fs_fastpath** ***ioo*** option is set to on by default in AIX 6.1 and AIX 7**.**1, but could be activated on a session basis in AIX 5.3. As with direct I/O, the file system block size (**agblksize**) should be 2K for a 2K ASE page size; otherwise the default 4K file system block size is correct.

Considerations for temporary databases

ASE does not need to recover data in the tempdb database after a crash. You can use this fact for performance benefits. Since the release of ASE 12.5.0.3, SAP Sybase has introduced optimizations to decrease the amount of writes that occur in a temporary database (a feature called *lazy writes*). These options are extended in ASE 15.0.2 and higher versions. Still, if the amount of data in a tempdb table exceeds the cache size, then the ASE engine obviously needs to write the data to disk. In addition, ASE writes data to disk as part of truncating the log (as indicated by the ASE **trunc log on chckpt** value that is set on temporary databases). The ASE engine also writes data to disk during page splits. With these two exceptions, the inline writes to disk are entirely gone.

With the **dsync** attribute set to *off*, file system devices provide fast writes. Therefore, each file system device that is used solely for the tempdb file should have this option set to *off*. Use the sp_deviceattr system procedure to do this: `sp_deviceattr '<device name>', 'dsync', 'false'`. For the same reason that **dsync** is not required with tempdb, direct I/O and concurrent I/O need not be used for tempdb either.

Because ASE does not need to recover tempdb data, ramdisk is an attractive alternative for accelerating tempdb. AIX 5L, AIX 6.1, and AIX 7.1 support ramdisk. The best resource to create, size, and remove

ramdisks is the man page for **mkramdisk**. However, you must reinitialize the ramdisk through a script on reboot.

> **Note 1:** Fast devices for tempdb do not replace cache for tempdb. Fast devices help write operations; tempdb cache helps read operations.
>
> **Note 2:** If you are using the tempdb cache together with a file system on ramdisk, you are in effect triple-buffering the data, which is counterproductive. It is recommended to mount the tempdb ramdisk file system with the **release behind read-write** (rbrw) option, which avoids persisting file system cache for tempdb.

Special considerations for the 2 KB page-size ASE

Many users of ASE and AIX implement ASE with the legacy server page size of 2 KB even though 4 KB, 8 KB, and 16 KB page sizes are available with ASE 12.5 and higher versions, and 4 KB and 8 KB page sizes are known to improve performance in general purpose databases.

The reason usually given for creating new servers at 2 KB page size is for ease of loading from dumps made on other systems with 2 KB page sizes.

In ASE, disk reads and writes conform to the page size of the server, except when reading and writing into large I/O buffer pools (for example, 16 KB) where the target buffer-pool size dictates the I/O block size. On AIX, certain kinds of disk I/O tasks degrade significantly with block sizes that are less than 4 KB. This is because AIX does disk I/O tasks in block sizes that are multiples of the default memory page size of 4 KB. Thus, 2 KB or smaller I/O tasks are concatenated into 4 KB blocks. A 2 KB read of a 4 KB block can partially lock the page, forcing this I/O operation to complete before another I/O task can be initiated against the adjacent 2 KB of the page.

The degradation of 2 KB reads and writes is most conspicuous in sequential operations of 2 KB or smaller against a JFS or JFS2 file, but it is also noticeable in random I/O processes that run against file systems. You can improve performance by the judicious use of large I/O buffer pools that are associated with named caches or the default data cache. It might occur to an imaginative DBA that it is possible to avoid the liability of the 2 KB page size altogether by configuring pools for large I/O volumes. Yet, at best, this is a partial solution, because the ASE optimizer chooses I/O size. Even if I/O size were forced, there are a number of scenarios where ASE has to do single-page I/O tasks, for example, for internal structures and for the first extent in every allocation unit.

On AIX 5L, user experience and lab tests show 25% better performance of a 2 KB ASE 12.5 on raw character devices, as compared to file devices. This finding applies to tempdb in some cases, as well. The use of raw character devices with 2K ASE page size is strongly recommended for AIX 6.1 and AIX 7.1.

If a 2 KB page size ASE must be maintained, it is strongly recommended that you use raw-character devices for database storage on AIX, as they are not subject to the 2 KB block I/O liability. Even if you migrate across platforms by using dump and load, you should do a conversion to raw character devices (if necessary by creating symbolic links between the file device names and the actual raw devices).

Virtual I/O in AIX 5L to AIX 7.1 with POWER5, POWER6, and POWER7

Virtual I/O is a component of the IBM Power Systems PowerVM suite of functions, and is frequently combined with shared-processor LPAR (though technically independent of it). It is principally a means of sharing network and SAN-adapter bandwidth and for deploying root disks from a central source for rapid creation of new LPARs.

A virtual I/O server (VIOS) is a specialized LPAR (running a form of AIX 6.1 or AIX 7.1 with specialized virtualization drivers) that owns physical adapters and, hence, physical network and disk I/O resources, making them available to client LPARs through a system of virtual devices.

You can set up a VIOS with several Ethernet adapters and it can pass them along to client LPARs as a whole or as fragmentary shares of the bandwidth. Likewise, the VIOS can pass along whole LUNs to client LPARs, or it can divide them into pieces, which the client LPARs perceive as whole, but smaller, hdisks. Thus, it is common to take a 73 GB internal SCSI drive and divide it into four segments, each large enough for one client's root file systems.

You can also mix and match virtualized and dedicated resources at the LPAR level. Thus, you might serve up a network to an LPAR from a VIOS, yet give it actual HBAs for direct attachment to SAN switches.

A single client LPAR can get resources from two or more VIOS instances. Multiple VIOS instances are a matter of best practice in highly available virtualized environments. Root disks are mirrored to two VIOS instances, Ethernets are Ether-channeled from two VIOS instances, and LUNs are multipathed through two VIOS instances. The AIX built-in multipathing subsystem multipath I/O (MPIO) allows for setting up the failover mode with a specified priority for each LUN. Thus, it is possible to achieve a level of coarse-grained load balancing by setting the primary path for every other LUN to **VIOA**, and by setting the other primary path for the remainder of the LUNs to **VIOB**. Current versions of AIX 5.3, AIX 6.1, and AIX 7.1 also support an MPIO round-robin mode in which for each of **n** paths, every **nth** I/O is sent down that path.

From the client LPAR point of view, virtual resources look and act the same as dedicated resources. However, they are implemented as virtual devices: virtual Ethernet adapters, virtual SCSI devices, or virtual fiber adapters when using the N-Port ID Virtualization (NPIV) protocol. Virtualized I/O resources are operationally transparent to SAP Sybase ASE. If the system administrator does not tell the DBA that storage or network is virtualized, the DBA might never notice. Still, the DBA should be aware of all virtualized resources as it might be important when working with SAP Sybase Tech Support.

Virtual SCSI hdisks can be grouped into volume groups and carved into logical volumes (just as a dedicated SAN LUN is). Virtually hosted file systems and raw virtual SCSI devices can be specified as the *physical devices* backing ASE devices just as would be done with dedicated resources. Asynchronous I/O and other I/O tuning parameters apply in the same way. Virtual SCSI hdisks can also be passed through directly to client LPARs. It is preferable for heavily used virtual SCSI database storage to pass through entire LUNs from the VIOS to the client, rather than to carve up LUNs into logical volumes that multiple clients can share. The latter practice, sharing fractions of LUNs among several clients is the preferred method for provisioning OS images and scratch space. Along with dual VIOS for redundancy and load balancing, serving entire LUNs to each distinct ASE is the standard practice in the field when using virtual SCSI protocol, and the default behavior of NPIV protocol, which is described shortly.

A small, well-documented latency is associated with virtual SCSI I/O methods.

Listing 9 is adapted from *Using the Virtual I/O Server, Fourth Edition*, an IBM product manual. The decimal values are percentages of latency at each block size for the two types of backing device, physical disks, and logical volumes.

Backing type	**4 KB**	**8 KB**	**32 KB**	**64 KB**	**128 KB**
Physical disk	0.032	0.033	0.033	0.040	0.061
Logical volume	0.035	0.036	0.034	0.040	0.063

Listing 9: Adapted from the 'Using the Virtual I/O Server, Fourth Edition' manual

You can minimize this small overhead in the following ways:

- Ensure that the queue depth on the client virtual SCSI device matches that of the backing device on the VIOS.
- Ensure that the combined queue depths of the virtualized LUNs do not exceed the queue depth of the virtual SCSI adapter (that is, the path over which data is transmitted between the client and VIOS).
- Ensure that the VIOS OS (not only its client LPARs) is I/O-tuned according to AIX best practices.
- Ensure that the VIOS has enough processor (entitlement) and has an advantage in obtaining the processor it needs to service I/O traffic. If the VIOS is competing with its clients for processor, I/O performance will degrade and this degradation will be passed through to the client LPARs. For example, the service times experienced on the client LPARs might be significantly greater than those on the VIOS hdisks themselves. Many sites give the VIOS a greater weight (refer to the preceding discussion of shared processor parameters) than the client LPARs to ensure that it has preferred access to excess processor.

The NPIV protocol, which is gaining wide acceptance for performance and ease of administration, maps a virtual fiber adapter from the physical HBA owned by VIOS to the client LPAR. The client LPAR, instead of being presented with a generic virtual SCSI device, is presented with an hdisk carrying the identification and properties of the actual SAN device. Thus the vendor identification of the LUN – EMC, Hitachi, IBM XIV® Storage System, and so on will be presented to the client, along with virtual worldwide port names (WWPNs) and the flags which allow management of queue depth and SCSI3 persistent reservation. Moreover, NPIV virtual storage further reduces the minimal latency associated with PowerVM storage virtualization in general. NPIV requires an NPIV-capable HBA on the server, and an NPIV-capable SAN switch.

Given the lower latency and potentially greater throughput of NPIV virtualized storage, it may be asked, what use case remains for virtual SCSI (VSCSI) if a client infrastructure supports either. One good answer is rapid provisioning, for example in a *cloud* deployment. More SAN storage than is presently needed can be presented to the VIOS. When the storage is called for to provision new LPARs, if using NPIV, the virtual WWNs of the new LPAR have to log in to the SAN switch and be zoned by an administrator. However, if the reserve storage is allocated to the VIOS instance as VSCSI, the system administrator can deploy the storage to a new LPAR immediately, without involving SAN administration.

A potential issue with any type of virtual I/O method is that it adds another abstraction layer that can hide the underlying performance issues. A common performance pitfall with VSCSI shared storage is that the disks that hold ASE LUNs also hold data for other applications and file system uses. A corresponding liability with NPIV virtual storage is that the physical HBA on the VIOS instance might be shared with other client LPARs. Certain types of workloads, notably random reads, cannot be satisfied by an on-controller cache and, hence, the underlying disks are hit. Concurrent activity from other applications also hits the disks, causing performance degradations. You can take this scenario to another level with VIOS, in that multiple LPARs potentially share the same underlying LUNs or HBA adapters. Hence, the abstraction layer

is not the cause of performance hits, but a layer that a performance and tuning effort must map through to understand how the underlying resource is used.

In the previous version of this white paper, it was reported that SAP Sybase data devices on VSCSI could be redeployed through dedicated fiber-attached storage, and then *swung* back to VSCSI again, without altering the data or ASE's ability to access it The only condition is that the virtual SCSI LUN passes through the IEEE Unique Identifier of the physical LUN. At that time, this practice was not officially supported by IBM; now it is supported, along with migration of VSCSI storage to NPIV and back, as well as to direct-attached fiber and back. Again, the provision is that the IEEE Unique Identifier be passed along. The ability to migrate SAP Sybase data devices transparently from VSCSI to NPIV or from the direct-attached fibre to NPIV is a great advantage when upgrading SAN environments.

Examining network performance

Increased ASE network-packet (tabular data-stream or TDS packet) sizes usually provide a performance benefit and almost never hurt performance. Usually, a 2 KB TDS packet size is a good starting point (by no coincidence, the new default in ASE 15.0). Certain operations, such as performing bulk loads, retrieving larger result sets, and operating with large objects (LOBs), benefit from a packet size that is 8 KB to 16 KB. The *SAP Sybase System Administration Guide* (Volumes I and II) documents client and ASE network configuration (refer to the "Appendix D: Resources" section for the references.).

A common objection to increasing the TDS packet size refers to underlying maximum transmission unit (MTU) and Ethernet frame sizes, both of which are in the 1500-byte range. However, multiple TCP segments or Ethernet frames are significantly cheaper than multiple application-level protocol packets, (for example, TDS). Obviously, if network performance or utilization is of a particular concern, then you must test the application and configuration changes.

As is true for all networks, some checks are necessary (for example, to ensure consistent MTU across nodes). MTU mismatches en route between a client and the ASE engine can cause severe performance issues for OLTP applications, and decision-support systems (DSS).

The universal UNIX **netstat** command allows administrators to monitor network-level statistics. The AIX **entstat -d <ent#>** command is used to monitor the adapter level. Neither of these monitoring utilities requires root privileges. Check the output from the **netstat -ir** command for packet-level errors and for relative traffic. Check the output from the **entstat -d** command for collisions and overflows. You can also use the **netstat -a** command to check for hung sockets. Look at the values displayed on the output lines for CLOSE_WAIT or FIN_WAIT.

The majority of the 137 network tunables in AIX are suitable for most topologies and host configurations. However, certain AIX network tunables need to be modified for optimal data server performance, and others need to be coordinated with the settings of clients and application servers communicating with ASE.

You can view all of the network tunable parameters in AIX through the **no** command. Changing the settings requires root access. In AIX 5.3. AIX 6.1, and AIX 7.1, several of the most important network

tunable parameters are set by default on the interface level*, and unless this priority is deliberately changed, the interface-level settings take precedence over the global settings seen from the command **no –a.** Refer to the AIX Power Systems Infocenter Performance and Tuning Guides for your AIX level in the "Appendix D: Resources" section for more details.

The most common network parameters that can be set at the interface level are:

- tcp_sendspace
- tcp_recvspace
- rfc1323
- tcp_nodelay

In a mixed-platform environment (for example, a Linux, Hewlett-Packard HPUX, or Sun Solaris client and an AIX data server) it is important to look for differences in the TCP/IP stack parameter defaults and site-specific settings among the various platforms,

Here are four examples that compare some default parameter values between Solaris 8, 9, and 10 and AIX environments (in Linux, the names are close to Solaris but the defaults differ). These observations are not intended to be comprehensive, but to alert the database and systems administrators to the need to coordinate settings between clients, applications, and data servers.

- In the AIX environment, the **sack** (selective retransmission of dropped packets) value is set to off, by default. Instead, there is a different retransmission algorithm, **tcp_newreno**, which is on by default. In the Solaris environment, the semi-equivalent algorithm to enable **sack** is **tcp_sack_permitted=2**. There is no direct equivalent for the **tcp_newreno** algorithm. Sack is a handshake, so if it is used in the rest of your environment, you need to enable it on your AIX servers. Option tcp_newreno implements the same TCP Fast Retransmit algorithm but on the sender side only. So, if sack is enabled in your environment, tcp_newreno should be disabled.
- By default, the AIX 5L, AIX 6.1, and AIX 7.1 operating systems disable the **rfc1323** parameter. In Solaris, the equivalent **tcp_wscale_always** parameter defaults to *disabled* in Solaris 8 and Solaris 9 but enabled in Solaris 10. In general, the values of rfc1323 (by whatever name) should agree on both sides of the client-server relationship. As mentioned earlier, the setting of rfc1323 at the interface level in AIX overrides the global value of **no**, so it should be enabled or disabled on a per-interface basis.
- The **tcp_sendspace** and **tcp_recvspace** parameters must agree between the client and server. In Solaris 8, 9, and 10, the equivalent of the **tcp_sendspace** parameter is **tcp_recv_hiwat** and the equivalent of **tcp_recvspace** is **tcp_xmit_hiwat**. The AIX default for each is 16 KB and for Solaris, it is 48 KB. Both values are normally increased to 200 KB, or more, for database servers. Again, **tcp sendspace** and **tcp_recvspace** can be set at the interface level, and that setting takes precedence over the global no setting.
- Special note should be taken of the lo0 or loopback interface. If applications or batch processes are run against an ASE instance in the same LPAR, the traffic goes over the loopback interface.

---

* The TCP/IP parameters that can be set on an interface level [**interface specific network options (ISNO)**] may be viewed by the command **lsattr –El <interface, e.g. en0>.** They can be set or modified through a **chdev** command such as **chdev –l en0 –a tcp_sendspace=262144** which would set the **tcp_sendspace** parameter on en0 to 256 K.

So, the settings of **tcp_sendspace**, **tcp**_recvspace, **rfc1323**, and **tcp_nodelay** need to be set on the lo0 loopback interface. To improve overall loopback performance, the **no** option tcp_fastlo must be enabled.

- The parameter **tcp_nodelay**, when enabled, overrides the default behavior of the modern TCP/IP stack, which is to batch together many small packets before transmitting a frame. In addition, enabling **tcp_nodelay** overrides the Nagle Algorithm which stipulates that a connection can have at most one outstanding unacknowledged small segment. AIX, Solaris, and Linux call this tunable by the same name. Request-response workloads often benefit from enabling **tcp_nodelay**, and it is commonly implemented at the socket level in application code, as in SAP Sybase ASE. The ASE **tcp_nodelay** configuration parameter, which defaults to **on**, sets **tcp_nodelay** at the socket level on startup and on each connection. **Tcp_nodelay** should be enabled or disabled on both sides of a connection, as between client and server. On the receiver, you can also use the no command to set tcp_nodelayack, which causes immediate acknowledgement of the arriving small packets. If **tcp_nodelay** is enabled in ASE, it should not be necessary to do so at the OS level. But, if there is a particular reason for doing so at your site, note that, as in **tcp_sendspace** and **rfc1323,** the setting at the interface level takes precedence over the global setting through the **no** command.

Virtual Ethernet I/O buffers

Virtual Ethernet is a protocol implemented by IBM PowerVM to provide interconnects between LPARs on the same physical server. Typically, a VLAN is implemented between one or more client LPARs and a VIOS instance, which bridges the internal traffic to an external network through a *shared Ethernet adapter*, the logical combination of a physical adapter connected to the external network and a virtual adapter connected to the VLAN on which the client LPARs reside .When using Virtual Ethernet, the systems administrator should be aware of, and monitor for, dropped and retransmitted packets at the virtual Ethernet level. These events, along with their most frequent cause, a *no resource error* , are displayed by the command **entstat –d <enX>** where **enX** is the virtual adapter en0, en1, and so on. The remedy for these retransmit errors is usually to increase the size of certain buffers, which are specific to the virtual Ethernet adapter. In this example from AIX support, the recommendation is to increase the minimum buffer count for those buffers (**Small** and **Medium**) where near the maximum number have been used since the last reboot. In this example, the minimum count for **Small** buffers should be increased to 2048 and the minimum count for **Medium** buffers should be increased to 256.

Ethernet Statistics (ent0):
Receive buffers

Buffer type	Tiny	Small	Medium	Large	Huge
Min Buffers	12	**512***	**128***	24	24
Max Buffers	2048	**2048**	**256**	64	64
Max Allocated	512	**1956**	**235**	24	24

*Table 2: Virtual Ethernet I/O buffer statistics from **entstat -d**Ulimits*

It is common to encounter challenges during backups or unloads. You can trace these to issues related to creating large files. The root cause is that the default file size ulimit is 1 GB.

In the AIX operating system, the system administrator can manage configurable limits in the **/etc/security/limits** file. The administrator needs to create a stanza for the Sybase functional user and set the file-size limit (**fsize**) to **-**1. The default behavior of the **fsize** token is to set the hard limit (**hard_fsize**) as well. It is recommended that the **nofiles** parameter in the **/etc/security/limits** file remains at its default value unless there is a specific application requirement for a larger value. **ASE 15.7 update: for ASE 15.7 threaded kernel only, nofiles should be increased by a multiple of the default value times the number of engine threads.** This is because there is only one ASE 15.7 process visible to the OS, so only as many file descriptors as allowed by **nofiles** would be available to the entire ASE instance, not to each engine as before. Ulimits can also be set and managed through **smitty user.**

The network caching (netcd) daemon

The **netcd** daemon reduces the time taken by the local, DNS, NIS, NIS+ and user loadable module services to respond to a query by caching the response retrieved from resolvers. When the **netcd** daemon is running and configured for a resolver (for example, DNS) and a map (for example, hosts), the resolution is first made using the cached answers. If it fails, the resolver is called and the response is cached by the **netcd** daemon. In AIX, the **netcd** daemon is not running by default. It is started with the command **startsrc –s netcd**. Consult the AIX manual page for further details.

The use of the netcd daemon with Sybase ASE is strongly recommended to improve response time in DNS lookups by ASE and stored procedures such as **sp_sendmsg**.

* Marks those buffers which need to be increased by matching their Min Buffers to their Max Buffers for the particular interface in question.

Summary

Running SAP Sybase ASE on the AIX operating system is an increasingly common combination that can perform exceptionally well, particularly for applications with very high throughput requirements.

Since 2005, when SAP Sybase and IBM began to collaborate on best practices documentation for ASE and AIX, the POWER processor has quadrupled in throughput, PowerVM virtualization has become more sophisticated, and both ASE and AIX have evolved to meet ever higher client demands for functionality and performance. Every section of this fourth major revision of ***Optimizing SAP Sybase ASE on IBM AIX*** contains new material. The ASE 15.7 threaded kernel is the most extensive change in decades to the interaction of ASE with its host operating system. And in the past 15 months, many challenges and opportunities for learning and enhancement have come to attention from the field. The SAP Sybase and IBM co-authors often work together on case resolutions.

In concluding, the authors would like to extend thanks to the community of database and system administrators, who support SAP Sybase ASE and AIX, for their patience, cooperation, and valuable contributions to the best practices.

Appendix A: Tuning and performance testing of SAP Sybase ASE 15 on AIX 6.1 and POWER7

Rationale:

During 2011, the IBM author of this white paper conducted approximately 200 test runs consisting of four different workloads on a Power 720 server with AIX 6.1 and SAP Sybase ASE 15.

The purpose of the exercise was to validate observations made in the field during 18 months of ASE 15 on POWER7 processor-based servers, principally concerning engine count, runnable process search count, SMT, 16 MB pinned memory, and virtual processor configuration.

The workloads and the hardware platform are rather small, but the workloads have been shown over time to predict accurately which ASE and OS tuning changes are beneficial and which are harmful to ASE on AIX performance in the field.

The goal of the tuning exercise was to achieve near-optimal baseline performance of the workloads on the given Power 720 hardware and AIX 6.1, so that the effect on performance of changing engine count, runnable process search count, SMT mode, pinned memory or virtual processor count could be clearly attributed.

All the tuning changes are standard, in the sense of being commonly recommended by SAP Sybase or by IBM. This does not mean that the tuning changes that were most beneficial in this exercise are necessarily going to be beneficial to actual client workloads; conversely, as will be noted, a number of common and often effective measures made no difference in this exercise, either positive or negative. The same is likely to occur in the field. So, this exercise must be regarded as an indication of a method rather than as a prescription.

One topic of perennial interest that is not addressed in this exercise is the comparison of shared and dedicated processors. The reason is that not sufficient cores were available to make for an interesting comparison. Several lab exercises since 2005, and a large range of field experience, have shown that the equivalent of dedicated performance can be achieved in the shared pool mode through appropriate allocation of entitlement, virtual processor count, and weight (even in the face of competition from other LPARs). This is addressed in the body of the white paper.

Description of the workloads: BatchStressTest, MixedStressTest, StaggeredStressTest, and TPC-C

MixedStressTest and **BatchStressTest** use 50 slightly modified copies of the **Pubs2** database which ships as a training sample with SAP Sybase ASE. The workloads consist of simple and complex queries drawn from SAP Sybase ASE training manuals, updates, dropping and reloading tables, dropping and recreating indexes, and performing maintenance.

MixedStressTest consists of 200 users running a package of queries or updates, each connecting, disconnecting, reconnecting, and running the same script on the same Pubs copy, in an endless loop, until the test is terminated. A significant part of the workload is connection handling, and it is more processor-

intensive than the other versions of **StressTest**. **MixedStressTest** has the performance characteristics of a *chatty* OLTP workload.

BatchStressTest consists of 20 users each assigned a randomized set of queries and updates against any of the 50 Pubs copies. The 20 users stay connected for an extended period of time, while running their scripts in an endless loop until the test is terminated. This form of the test, as the name indicates, has batch-like performance characteristics. It is more disk I/O intensive than **MixedStressTest** because the ratio of updates to read-only queries is greater. As any user can hit any Pubs copy, **BatchStressTest** exhibits lock contention and some deadlocking.

StaggeredStressTest is a variant on **MixedStressTest** with random sleeps introduced to stagger the starts of each of the 200 users. The purpose is to create a more fluctuating workload than that of a typical stress test, in order to test the efficacy of reducing runnable process search count.

The industry standard Transaction Processing Performance Council Benchmark C (TPC-C) benchmark is also used in this exercise as an additional measure of ASE performance in the different SMT modes, not as a competitive performance benchmark as such. The ASE and OS configuration that is optimized for **MixedStressTest** and **BatchStressTest** are supplemented with the additional caches and cache bindings and other details of the TPC-C setup.

Hardware, operating system, and ASE configuration

- *Hardware*: Power 720 server with four 3.1 GHz processors in a single pool; 16 GB total memory.
- *Operating System*: AIX 6.1 TL5
- *LPARS*: Two client LPARs with 3 GB memory each, and two VIOS instances. Processor is shared, and allocations to the client LPARs are varied during the tuning.
- *ASE versions:* ASE 15.0.3 ESD2, ASE 15.0.3 ESD4, and ASE 15.5 ESD4 were each tested. As no significant difference in their performance with these workloads was observed, the results are not differentiated.
- *Test environment*: An ASE environment and StressTest or TPC-C client environment were built on each LPAR, and were used alternately as client or as ASE server depending on the test. In other words, the client was not running on the same processor and memory as the data server.
- *Storage* is a mixture of IBM System Storage® DS4700 and IBM XIV Storage System, achieving a steady throughput of 25 MBps per LUN.
- *ASE data devices* are on raw logical volumes, **max interpolicy** striped across the available LUNs (hdisks). Also known as **physical partition spreading** or **poor man's striping**, this form of striping is discussed in the body of the white paper.
- *Network:* All ASE client traffic between the two LPARs travels over a dedicated private internal VLAN.
- *Initial configuration:*
  - OS non-defaults
    tcp_sendspace = 256 KB

tcp_recvspace = 256 KB
AIX_THREADSCOPE = 'S' (Sybase environment only)

- Virtualization specifics
  - Storage on each client LPAR is served by virtual SCSI (VSCSI) protocol over a path from each VIOS instance.
  - Outside network connection to each LPAR is provided by a shared Ethernet adapter configured on one of the VIOS instances.
  - Test traffic is routed between the client and data server LPARs through an internal VLAN.
  - The four processors on the Power 720 server are in a single shared pool, accessed by the VIOS instance, each with an entitlement of .5 and 1 virtual processor, and by the two client LPARs, each with an entitlement of .5 to.8 and from three to six virtual processors depending on the test.

- SAP Sybase ASE initial non-defaults and parameters modified during testing.

```
[Named Cache:default data cache]
        cache size = 1000MB
        cache status = default data cache
        cache replacement policy = DEFAULT
        local cache partition number = DEFAULT                  (modified to 2 and 4)

[4K I/O Buffer Pool]
        pool size = DEFAULT
        wash size = 35016 K
        local async prefetch limit = DEFAULT


[16K I/O Buffer Pool]                   (8K pool was added taking half of the memory of 16K pool)
        pool size = 450MB
        wash size = DEFAULT
        local async prefetch limit = DEFAULT

[Named Cache:tempdb_cache]
        cache size = 400MB
        cache status = mixed cache
        cache replacement policy = DEFAULT
        local cache partition number = DEFAULT                  (modified to 2 and 4)

[4K I/O Buffer Pool]
        pool size = DEFAULT
        wash size = 35020 K
        local async prefetch limit = DEFAULT

[16K I/O Buffer Pool]
        pool size = 200MB
        wash size = DEFAULT
        local async prefetch limit = DEFAULT

[Meta-Data Caches]
        number of open databases = 60
        number of open objects = 2000
        number of open indexes = 1500
        number of open partitions = 1500

[Disk I/O]
        disk i/o structures = 1024
        number of devices = 127

[Network Communication]
        default network packet size = 4096
```

```
            max network packet size = 4096

[O/S Resources]
            max async i/os per engine = 2147483647
[Physical Memory]
            max memory = 1000000

[Processors]
            max online engines = 8
            number of engines at startup = 2                (modified to 4, 5 and 6 in various tests)
[Query Tuning]
            optimization goal = DEFAULT                     (tried allrows_oltp)
[SQL Server Administration]
            procedure cache size = 29632
            runnable process search count =DEFAULT          (3 and 100 in certain workloads)
            i/o polling process count = DEFAULT              (other values tried)
[User Environment]  number of user connections = 500
[Lock Manager]
            number of locks = 250000
```

BatchStressTest

The same optimizations (with the exception of RPSC) provided the best results for BatchStressTest, MixedStressTest, and StaggeredStressTest. Thus the full iterative tuning process will be presented only for BatchStressTest. Likewise, certain standard tuning recommendations made no difference to any of the three workloads, so they will be listed only in the context of BatchStressTest.

Three different measures were recorded for each BatchStressTest run.

- Batch iterations: The number of times the entire batch stream completed for each of the 20 users within the 5-minute test
- Total iterations: The total number of SQL scripts that ran in 5 minutes
- Update iterations: Total iterations minus the pure query scripts those scripts involving inserts, updates, and Data Definition Language (DDL) operations.

In the case of BatchStressTest, the percentage gains in these three measures were almost identical, so for economy only the batch iteration results are presented. Results are presented both in terms of percentage gain per change, and each incremental change as a percentage of the optimal configuration represented as 100%.

The iterative character of the tuning process is largely owing to three rules of method:

1. Add ASE engines only when the current online engines are more than 80% busy.
2. Add processor capacity only when you add engines.
3. Increase hdisk queue depth only when there is an evidence of queue overflow and/or queue wait, and then only in small increments.

Thus, when statement cache was added and an 8k large pool was added to the default data cache, throughput increased adequately to put pressure on the hdisk queues. The first iteration of queue-depth tuning increased engine utilization to an extent that warranted the addition of a third engine. Adding a third engine again increased disk I/O, putting more pressure on hdisk queues.

A second iteration of queue depth tuning allowed the addition of a fourth engine, an increase of virtual processors and entitlement, and a beneficial increase of data cache partitions from two to four in order to

reduce spinlock contention. Note that increasing cache partitions did not improve performance in an earlier stage of tuning.

As the four processors on the Power 720 server are shared among the two VIOS instances and the two client LPARs, any change to entitled capacity or virtual processor count on one of the LPARs affected the performance of the others. An entitlement of .8 three virtual processors and four ASE engines on the data server LPAR was optimal. Increasing entitlement to 1 and virtual processors to four had a slight negative impact because it introduced competition for processors with the VIOS instances, slightly degrading I/O performance.

Four virtual processors were necessary for the tests of SMT4, SMT2, and single threaded modes. This is because, in a single threaded mode, only four logical processors are presented, and thus only four ASE engines could be started. Hence four virtual processors were used in all the SMT comparisons of BatchStressTest, MixedStressTest, and the TPC-C benchmark.

BatchStressTest	Entitled capacity	Virtual processors	ASE engines	RPSC	Percentage of improvement	Percentage of best result
Starting point					Batch iterations	Batch iterations
No 8K pool, data cache, no statement cache	0.5	2	2	2000	0	41
Added 8K pool to data cache, added 13 MB statement cache, literal autoparam	0.5	2	2	2000	8	44
Increased queue depth on DS4K LUNs from 2 to 10	0.5	2	2	2000	11	49
Increased online engines to 3	0.5	2	3	2000	1.5	50
Global cache partitions=4	0.5	2	3	2000	0	50
Global cache partitions=2	0.5	2	3	2000	0	50
Optimization goal=allrows_oltp	0.5	2	3	2000	-1.5	49
Optimization goal=allrows_mixed	0.5	2	3	2000	1.5	50
Increased queue depth on DS4K LUNs from 10 to 16	0.5	2	3	2000	35	68
Increased entitlement to .8, virtual processors to 3, online engines to 4, cache partitions to 4	0.8	3	4	2000	3	71
Reduce RPSC to 3 with 4 online engines	0.8	3	4	3	9	77
Reduce RPSC to 100 with 6 online engines	0.8	3	6	100	-2	76
Reduce RPSC to 3 with 6 online engines	0.8	3	6	3	4	79
Rebalanced I/O and redistributed log devices	0.8	3	4	3	4	82
Reduced engines back to 4 (limit of SMT testing)	0.8	3	4	3	0	82
Implemented 16MB large pinned pages	0.8	3	4	3	25	100
Implemented log cache (no change in performance)	0.8	3	4	3	0	100
Increased online engines to 6 (no significant gain)	0.8	3	6	3	0.01	100
Reduced SMT from 4 threads to 2 (no change)	0.8	3	4	3	0.004	100
Reduced SMT from 2 threads to 1 > 7% degradation	0.8	3	4	3	-7.5	92.5

Table 3: BatchStressTest

The iterative character of tuning is further emphasized by the rebalancing of I/O and redistribution of the log devices (4% gain) and by the implementation of 16 MB pinned pages (25% gain). It was observed that the last pages of the log devices resided on a single hdisk, which was also being hit hard by data updates. After the log devices were redistributed and 16 MB pages were implemented, the difference in performance between four online engines and six online engines diminished to statistical insignificance. Moreover, the optimal value of RPSC (3 for BatchStressTest and 2000 for MixedStressTest) proved to be the same whether four or six engines were online.

The following familiar tuning recommendations made little or no difference to the three workloads, BatchStressTest, MixedStressTest, and StaggeredStressTest:

- Raising and lowering I/O polling process count
- Binding named log caches
- Raising and lowering open object spinlock ratio
- Disabling virtual processor folding (**schedo** kernel tunable)
- Optimization goal, comparing allrows_oltp with allrows_mixed

In the case of BatchStressTest, there was no statistical difference between performance in SMT4 mode and SMT2 mode, but single threaded mode degraded performance by 7.8%

MixedStressTest

Some tuning changes affected updates differently from read-only queries in MixedStressTest. For example, implementing 16 MB large-pinned pages increased overall throughput (queries+updates) by 14%, while it improved updates by 17.7%. Increasing online engines from four to five and six marginally improved update performance while marginally degrading read-only query performance. This illustrates the fact that increasing engines can improve I/O processing, but an increase in idle polling often degrades computational performance. Reducing SMT threads from four to two degraded update performance by 4% without affecting query performance. But reducing SMT threads from two to one degraded both query and update performance.

The preference of both BatchStressTest and MixedStressTest for SMT4 reinforces the inference that ASE benefits from SMT multithreading both in being able to dispatch runnable engines faster, and in being able to parallelize tasks, running both in user mode and in kernel mode.

MixedStressTest All optimizations of BatchStressTest	Entitled capacity	Virtual processors	ASE engines	RPSC	Percentage of improvement Total iterations	Percentage of best result Total iterations	Percentage of improvement Updates only	Percentage of best result Updates only
Applied pinned pages and RPSC	0.8	3	4	2000	n/a	86	n/a	82.3
Implemented 16 MB large pinned pages					14	100	17.7	100
Increased engines from 4 to 5	0.8	3	6	2000	0.05	100	2.8	102.8
Increased engines from 5 to 6	0.8	3	6	2000	0.00	100	0	102.8
Reduced SMT from 4 threads to 2	0.8	4	4	2000	-1	99	-4	96
Reduced SMT from 4 threads to 1	0.8	4	4	2000	-2.5	97.5	-3.5	96.5
With ST (SMT1) increased VP's to 6	0.8	6	4	2000	1	101	-10	90

Table 4: MixedStressTest

StaggeredStressTest

In 18 months of field experience with ASE 15 on AIX 6.1 and POWER7 processor-based servers, two use cases have been documented for reducing runnable process search count. One is the disk I/O intensive workload where a greater number of engines is needed than for computationally intensive workloads, but where idle polling needs to be controlled. This has been illustrated in this laboratory exercise by BatchStressTest. The second case is the sharply fluctuating workload, where the engine count needed to handle peak load (more than 70% processor busy) generates excessive idle polling at a low load (15% to 25% processor busy). StaggeredStressTest, by adding variable lengths of sleep into each of work threads of MixedStressTest, is intended to simulate such a fluctuating load. A runnable process search count of 100 proved to be optimal for this workload. However, StaggeredStressTest did not benefit from 16 MB pinned pages.

TPC-C benchmark

The TPC-C benchmark was implemented with 35 threads and four warehouses. These are relative measures of the size of the TPC-C database and workload. The purpose of implementing TPC-C was to provide another workload to confirm the benefit of SMT observed in BatchStressTest and MixedStressTest. The results reported are in the standard TPC-C format: maximum, minimum, and average transactions per minute (TPM).

The benefit of 16 MB large pages was comparable to that of BatchStressTest and MixedStressTest and confirms earlier (2008) findings also using TPC-C.

Throughput in SMT2 mode was 90% of that in SMT4 mode, and throughput in single threaded mode was only 75% of that in SMT4 mode. Thus, the benefit of SMT4 for ASE 15 was more pronounced in the TPC-C benchmark than in the other workloads tested.

TPC-C 32 threads	Entitled capacity	Virtual processors	ASE engines	RPSC		Maximum TPM	Minimum TPM	Average TPM	Percentage of maximum
Four warehouses									
Starting point SMT4	0.8	3	6	2000	SMT4	66986	46069	56527	100
Reduced SMT from 4 to 2	0.8	3	6	2000	SMT2	60799	46151	53475	90
Reduced SMT from 4 to 1	0.8	3	6	2000	ST	50361	42049	46205	75

Table 5: TPC-C benchmark

Conclusions

Averaged across the three main workloads, SMT4 is a better choice for SAP Sybase ASE than SMT2 or single threaded mode. SMT4 is the default for POWER7 processor-based server running in the POWER7 mode.

There was no change in performance observed in disabling virtual processor folding for these test loads. Virtual processor folding provides more efficient processor consumption, better cache coherence, and less

context switching, and is the AIX 6.1 and AIX 7.1 default. Virtual processor folding must not be disabled on AIX for SAP Sybase ASE without consulting AIX technical support.

Over-allocating ASE engines has the same undesirable effects on POWER7 as in previous POWER architectures, including: excessive idle polling under low processor load, excessive object manager, and data cache spinlock contention under heavy processor load. Both excesses can degrade performance.

Two use cases for reducing runnable process search count were tested and confirmed—the case of an I/O intensive workload, and the case of a very uneven workload. On the other hand, reducing runnable process search count had a negative effect on MixedStressTest, a computationally intensive, network-bound workload. It was also observed that reducing runnable process search count does not compensate for excess engine count. The best practice is to achieve the optimal performance per engine at runnable process search count 2000 before attempting to reduce runnable process search count.

As reported in the previous version of this white paper, implementing 16 MB large pages in pinning ASE memory yields substantial performance gains (15% or more in three of the four workloads tested).

While the Power 720 platform on which the tests were performed is small by data server standards, the observations about the need to balance processor demand among LPARs and VIOS instances sharing a processor pool apply to larger systems as well. This topic is covered in the body of the white paper.

As a final reminder, do not take this tuning exercise literally, but as an illustration of an iterative method of performance optimization, which will vary in detail from one workload to another.

Appendix B: Comparing ASE 15.7 threaded mode with ASE 15.5 and ASE 15.7 process mode on AIX 7.1 and POWER7+

This section provides an exercise in tuning the ASE 15.7 threaded kernel.

This exercise employs the same three workloads: **MixedStressTest**, **BatchStressTest** and **TPC-C** as were used in the ASE 15.5 tuning exercise, Appendix A.

Each workload is optimized on ASE15.7 for a given processor count using the methodology given in Appendix A. Then the optimized configuration is run in SMT2 and single-threaded mode and then compared to the default SMT4 mode. Finally, the best threaded mode result for a given workload and processor count is compared to the best process mode result for that workload and that processor count. Results obtained in ASE 15.5 are practically identical to ASE 15.7 process mode.

The limits on scalability of these tests are not due to any inherent limitations of ASE or AIX/Power7, but reflect the physical limits of available CPU, disk I/O and network I/O, together with the size of the workloads and the number of clients used. The phenomenon of reaching an optimal engine count for a given workload and physical environment is discussed in the section “The performance penalty in over-allocating engines” in the body of the paper.

A 4.1GHz POWER7+ processor is used with configurations of 4 and 8 cores. SMT4 [quad-threaded mode] is the default. Memory is not altered with a change in processor count.

First, the results for each workload are given for the optimal engine count, one less than optimal, and one more than optimal.

Next, the best configuration in SMT4 mode is run in SMT2 [dual-threaded] and in single-threaded mode.

Last, the best ASE15.7 threaded mode results are compared with the best process mode result for the same core count and workload.

Tests of ASE15.7 were also conducted on a Power 720 server in a shared processor LPAR with three virtual processors and an entitled capacity of .8 (the same configuration as used in the ASE 15.5 tuning exercise reported in Appendix A). The overall pattern of performance is similar to the results presented here and will be referred to when relevant.

BatchStressTest – four dedicated processors. One disk controller and one network controller is used in all threaded runs.

Mode and engine number	Idle timeout or RPSC	SMT mode	Total transactions	Percentage (+ or -) best	Updates only	Percentage (+ or -) best
Threaded 3	100	4	82200	-9%	65543	-9%
Threaded 4	100	4	84869	-6%	67730	-6%
Threaded 4	500	4	90395	Best thread	72121	Best thread
Threaded **5**	500	4	84862	-6%	67687	-6%
Threaded 4	500	2	85845	-5%	68479	-5%
Threaded 4	500	1	46581	-48%	37191	-48%
Process 4	Rpsc 2000	4	91522	+1%	73044	+1%

BatchStressTest – eight dedicated processors. One disk controller and one network controller is used in all threaded runs.

Mode and engine number	Idle timeout or RPSC	SMT mode	Total transactions	% + or - best	Updates only	% + or - best
Threaded 5	500	4	100342	-1%	80042	-2%
Threaded 6	500	4	102265	Best thread	81616	Best thread
Threaded 7	500	4	99227	-3%	79217	-3%
Threaded 6	500	2	89664	-12%	71527	-12%
Threaded 6	500	1	97057	-5%	77470	-5%
Process 5	Rpsc 2000	4	105583	+3%	84261	+3%

BatchStressTest

sp_sysmon observations:

Increasing idle timeout from default 100 to 500 improved performance over 6%, changed the ratio of full sleeps to interrupted sleeps from 12:10 to 9:1, and reduced the ratio of voluntary:non-voluntary context switches from 1:300 to 1:5. The runnable task rate is generally low.

Spinlock contention is insignificant for this workload.

Performance observations:

BatchStressTest does not scale past four engines on four dedicated processors and does not scale past six engines on eight dedicated processors. In both configurations it is observed to use primarily the first two of four SMT threads. While the limit of scaling in the case of four processors might be attributed to the saturation of these first two processor threads, the 8-core result is not thus explained. Rather, in this I/O bound workload, the finite speed of physical I/O might limit the number of engines that can work efficiently before adding disproportionate resource consumption owing to idling .

Given an optimal engine count for its processor allocation, BatchStressTest process mode yields slightly but significantly better throughput than threaded mode.

MixedStressTest – Four dedicated processors. One disk and one network controller used in all threaded runs.

Mode and engine number	Idle timeout or RPSC	SMT mode	Total transactions	Percentage (+ or -) best	Updates only	Percentage (+ or -) best
Threaded 4	500	4	137385	-1%	62349	-33%
Threaded 5	500	4	138290	Best thread	84061	-10%
Threaded 6	500	4	134553	-3%	93022	Best thread
Threaded 5	500	2	84519	-39%	64388	-31%
Threaded 5	500	1	80606	-41%	61517	-34%
Process 5	Rpsc 2000	4	134575	-2%	90953	-2%
Process 6	Rpsc 2000	4	130748	-3%	94414	+1%

MixedStressTest – Eight dedicated processors. One disk and one network controller used in all threaded runs.

Mode and engine number	Idle timeout or RPSC	SMT mode	Total transactions	Percentage (+ or -) best	Updates only	Percentage (+ or -) best
Threaded 5	500	4	164298	-7%	103815	-15%
Threaded 6	500	4	176532	Best thread	121964	Best thread
Threaded 7	500	4	169313	-4%	107511	-12%
Threaded 6	500	2	133547	-24%	106577	-13%
Threaded 6	500	1	98169	-44%	75262	-38%
Process 5	Rpsc 2000	4	141801	-19%	105851	-13%

MixedStressTest

sp_sysmon observations:

Increasing engine count reduced the runnable task rate, even with too many engines. There are no full sleeps recorded for any run of MixedStressTest. The rate of interrupted sleeps increases with engine count. Changing between idle timeout of 100 and one of 500 made no difference to MixedStressTest, because it almost never voluntarily yields. For the same reason, voluntary OS context switches are very few. Voluntary context switches increase in proportion to non-voluntary (as would be expected) as engines are added. Spinlock contention is significant in MixedStressTest (object manager, procedure cache and data cache) limiting the scalability of the query component of the workload, even as update performance increases.

Performance observations:

On four dedicated processors, MixedStressTest scales ambiguously, with maximum overall and query throughput at five engines, while at six engines, query throughput drops almost 25%, while updates continue to increase. Spinlock contention is contributing to processor saturation, while I/O tasks appear less impacted. The same behavior is observed in the process mode with four dedicated cores.

Given eight dedicated cores, MixedStressTest scales to six engines before performance starts to deteriorate. In this case the best configuration is unambiguous, and the performance in threaded mode is unambiguously superior to process mode.

TPC-C – Four dedicated processors. One disk and one network controller used in all threaded runs.

Mode and engine number	Idle timeout or RPSC	SMT mode	Transactions per minute	Percentage (+ or -) best
Threaded 2	500	4	111765	-10%
Threaded 3	500	4	124610	Best thread
Threaded 4	500	4	114792	-8%
Threaded 3	500	2	119876	-4%
Threaded 3	500	1	96422	-23%
Process 5	RPSC 2000	4	125123	+0.4%

TPC-C – Eight dedicated processors. One disk and one network controller used in all threaded runs.

Mode and engine number	Idle timeout or RPSC	SMT mode	Transactions per minute	Percentage (+ or -) best
Threaded 5	500	4	183966	-3%
Threaded 6	500	4	189713	Best thread
Threaded 7	500	4	186538	-1.6%
Threaded 6	500	2	184843	-2.5%
Threaded 6	500	1	184291	-2.5%
Process 4	RPSC 2000	4	184706	[+0.4%][1]
Process 5	RPSC 2000	4	192066	[+1.2%][2]
Process 6	RPSC 2000	4	188533	[+1%][3]

TPC-C

sp_sysmon observations:

The rate of full sleeps to interrupted sleeps remains about 8:1 even as engines are added. While voluntary engine and system context switches remain at the same order of magnitude as engines are added, non-voluntary context switches increase logarithmically. Spinlock contention is insignificant.

Performance observations:

TPC-C in thread mode is performed best with three engines on four processors and with six engines on eight processors. Similar to BatchStressTest, TPC-C maximizes the use of two processor threads, and similar to BatchStressTest, it is likely to run into the physical upper limit of disk and network I/O throughput. Whether on the small shared processor system, on four dedicated cores or on eight dedicated cores, throughput is slightly yet significantly better in process than in threaded mode on POWER7 / POWER7+ and AIX 7.1. To illustrate this, the 8-core TPC-C table (in the previous section) gives corresponding thread and process mode results for *one engine too few*, *best*, and *one engine too many*. Process mode slightly outperforms thread mode in each case.

General observations

SMT:

This series of 4- and 8-core tests, along with the tests on the smaller shared processor LPAR, strengthen the case for SMT4 (quad-threaded, autonomic) mode as providing the best performance with ASE, whether in threaded or in process mode. There are no repeatable exceptions yet found to the decline in performance when SMT mode is reduced from 4 to 2 or 1.

These tests also address an hypothesis that the single-threaded mode might improve performance of workloads with heavy I/O if exposed to more processors than engines. That is, give the I/O handlers a full processor rather than a tertiary thread on a processor already in use by an engine. The inability of any of the three workloads to scale beyond six engines makes them suitable cases; yet their degraded

[1] Compared to thread mode 5 engines
[2] Compared to thread mode 6 engines
[3] Compared to thread mode 7 engines

performance in the single-threaded mode makes this suggested use case for single-threaded mode unlikely.

Appendix C: AIX 15.7 prerequisite – I/O completion ports

IOCP is an API for performing multiple simultaneous asynchronous input/output operations in AIX and other operating systems. An input/output completion port object is created and associated with a number of sockets or file handles. When I/O services are requested on the object, completion is indicated by a message queued to the I/O completion port. A process requesting I/O services is not notified of completion of the I/O services, but instead checks the I/O completion port's message queue to determine the status of its I/O requests. The I/O completion port manages multiple threads and their concurrency.

The IOCP application programming interface acts as a front end to the AIX asychronous I/O subsystem. The application, in this case ASE 15.7 in threaded mode, makes IOCP calls to complete certain asynchronous I/O requests, rather than calling the asynchronous I/O routines directly. The IOCP management layer helps to parallelize and synchronize the asynchronous I/O requests.

ASE 15.7 implements the IOCP protocol. In order to install and start ASE 15.7, the AIX IOCP subsystem must be enabled. This is performed by the system administrator through **smitty iocp** .

There are certain defects in the IOCP subsystem in AIX, which have been found and corrected.

ASE 15.7 requires AIX 6.1 TL7 or above. Hence the APARs and service packs (SPs) pertaining to IOCP fixes are limited to:

AIX level	Required SP	oslevel -s	bos.iocp.rte level
7.1 TL 2	SP2	7100-02-02-1316	7.1.2.15
7.1 TL 1	SP6	7100-01-06-1241	7.1.1.16
7.1 TL 0	SP8	7100-00-08-1241	7.1.0.18
6.1 TL 8	SP2	6100-08-02-1316	6.1.8.15
6.1 TL7	SP6	6100-07-06-1241	6.1.7.17

This list is inclusive in that the recommended level might contain more than one IOCP patch.
It is recommended that the client applies the entire service pack whenever possible.

If the client cannot upgrade to the required service pack at this time, it is recommended that a case with AIX be opened to obtain all prerequisites for applying the required level of file set **bos.iocp.rte** without applying the entire service pack.

Appendix D: Resources

The following websites provide useful references to supplement the information contained in this paper:

- IBM Power Systems AIX Information Center—all topics
  - http://publib.boulder.ibm.com/infocenter/aix/v7r1/index.jsp (AIX 7.1)
  - http://publib.boulder.ibm.com/infocenter/aix/v6r1/index.jsp (AIX 6.1)
  - http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp (AIX 5.3)
- Power Systems on IBM PartnerWorld®
  ibm.com/partnerworld/wps/pub/overview/news/B5P00PW
- AIX on IBM PartnerWorld
  ibm.com/partnerworld/wps/pub/overview/OE600
- IBM Publications Center
  www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi?CTY=US
- IBM Redbooks
  ibm.com/redbooks
- Redbooks most relevant to the white paper *Optimizing SAP Sybase ASE on IBM AIX*:
  - AIX Version 5.3 differences guide
    ibm.com/redbooks/redbooks/pdfs/sg247463.pdf
  - AIX Version 6.1 differences guide
    ibm.com/redbooks**/**redbooks/pdfs/sg247559.pdf
  - AIX Version 7.1 differences guide
    ibm.com/redbooks/redbooks/pdfs/sg247910.pdf
  - IBM PowerVM Virtualization Introduction and Configuration
    ibm.com/redbooks/redbooks/pdfs/sg247940.pdf
  - IBM PowerVM Virtualization Managing and Monitoring
    ibm.com/redbooks/redbooks/pdfs/sg247590.pdf
  - IBM Power Systems Performance Guide: Implementing and Optimizing

    ibm.com/redbooks/redbooks/pdfs/sg248080.pdf
- IBM developerWorks®
  ibm.com/developerworks/aix/
- IBM Techdocs technical white papers (including this white paper)

  ibm.com/support/techdocs/atsmastr.nsf/Web/TechDocs
- AIX on Power -- Performance FAQ December 4, 2012, by Dirk Michel

  http://public.dhe.ibm.com/common/ssi/ecm/en/pow03049usen/POW03049USEN.PDF
- Active Memory Expansion (AME) on IBM Power Systems with SAP Sybase ASE on IBM AIX by Brian Vicknair and Joerg Droste

  http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102240

- Information about SAP Sybase products
  http://www.sybase.com
- SAP Sybase ASE general information
  http://www.sybase.com/products/informationmanagement/adaptiveserverenterprise
- SAP Sybase ASE 15.5 manuals
  http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.infocenter.help.ase.15.5/title.htm
- SAP Sybase ASE 15.5 System Administration Guides
  - http://infocenter.sybase.com/help/topic/com.sybase.infocenter.dc31654.1550/html/sag1/title.htm
  - http://infocenter.sybase.com/help/topic/com.sybase.infocenter.dc31644.1550/html/sag2/title.htm
- SAP Sybase ASE 15.5 Performance and Tuning Manuals
  - http://infocenter.sybase.com/help/topic/com.sybase.infocenter.dc20020.1502/html/basics/title.htm
  - http://infocenter.sybase.com/help/topic/com.sybase.infocenter.dc00976.1502/html/statistics/title.htm
  - http://infocenter.sybase.com/help/topic/com.sybase.infocenter.dc00938.1502/html/locking/title.htm
  - http://infocenter.sybase.com/help/topic/com.sybase.infocenter.dc00842.1502/html/spsysmon/title.htm
  - http://infocenter.sybase.com/help/topic/com.sybase.infocenter.dc00848.1502/html/monitor_tables/title.htm
  - http://infocenter.sybase.com/help/topic/com.sybase.infocenter.dc00841.1502/html/phys_tune/title.htm
  - http://infocenter.sybase.com/help/topic/com.sybase.infocenter.dc00743.1502/html/queryprocessing/title.htm

- SAP Sybase ASE 15..7 System Administration Guides
  - http://infocenter.sybase.com/help/topic/com.sybase.infocenter.dc31654.1570/pdf/sag1.pdf
  - http://infocenter.sybase.com/help/topic/com.sybase.infocenter.dc31644.1570/pdf/sag2.pdf

- SAP Sybase ASE 15.7 Performance and Tuning Manuals
  - http://infocenter.sybase.com/help/topic/com.sybase.infocenter.dc20020.1570/pdf/basics.pdf
  - http://infocenter.sybase.com/help/topic/com.sybase.infocenter.dc00976.1570/pdf/statistics.pdf
  - http://infocenter.sybase.com/help/topic/com.sybase.infocenter.dc00938.1570/pdf/locking.pdf
  - http://infocenter.sybase.com/help/topic/com.sybase.infocenter.dc00842.1570/pdf/spsysmon.pdf
  - http://infocenter.sybase.com/help/topic/com.sybase.infocenter.dc00848.1570/pdf/monitor_tables.pdf
  - http://infocenter.sybase.com/help/topic/com.sybase.infocenter.dc00841.1570/pdf/phys_tune.pdf

  - http://infocenter.sybase.com/help/topic/com.sybase.infocenter.dc00743.1570/pdf/queryprocessing.pdf
- ASE 15.7 technical overview-- Peter Thawley
  https://websmp104.sap-ag.de/~sapidp/011000358700001121852012E/assets/sybase_ase_techovervw.pdf

- ASE 15.7 Technical look inside the "hybrid kernel" -- Peter Thawley
  http://www.sybase.com/files/Product_Overviews/ASE-15.7-New-Threaded-Kernel.pdf

Acknowledgements

Peter H. Barnett (IBM Corporation), Stefan Karlsson (SAP Sybase.), Mark Kusma (SAP Sybase) and Dave Putz (SAP Sybase) authored this paper. It was built on research from the authors and their IBM and Sybase colleagues, including (but not limited to): Mathew Accapadi (IBM), the late Barry Saad (IBM), Dan Braden (IBM), Grover Davidson (IBM), Steven Knudson (IBM), Prasanta Ghosh (SAP Sybase.) and David Wein (SAP Sybase).

Special thanks to Mathew Accapadi (IBM AIX Client Advocacy and Systems Assurance F.A.S.T) and David Wein (Sybase ASE engineering).

The authors welcome readers to share their experiences and to provide feedback at: phbarn@us.ibm.com (Peter H Barnett), mark.kusma@sap.com (Mark Kusma) and Dave.Putz@sap.com (Dave Putz).

Trademarks and special notices

References in this document to IBM products or services do not imply that IBM intends to make them available in every country.

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Information is provided "AS IS" without warranty of any kind.

All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer.

Information concerning non-IBM products was obtained from a supplier of these products, published announcement material, or other publicly available sources and does not constitute an endorsement of such products by IBM. Sources for non-IBM list prices and performance numbers are taken from publicly available information, including vendor announcements and vendor worldwide homepages. IBM has not tested these products and cannot confirm the accuracy of performance, capability, or any other claims related to non-IBM products. Questions on the capability of non-IBM products should be addressed to the supplier of those products.

All statements regarding IBM future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. Contact your local IBM office or IBM authorized reseller for the full text of the specific Statement of Direction.

Some information addresses anticipated future capabilities. Such information is not intended as a definitive statement of a commitment to specific levels of performance, function or delivery schedules with respect to any future products. Such commitments are only made in IBM product announcements. The information is presented here to communicate IBM's current investment and development activities as a good faith effort to help with our customers' future planning.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon

considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Photographs shown are of engineering prototypes. Changes may be incorporated in production models.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

www.sap.com