DB2 Information Management Software

The Self-Tuning Memory Manager (STMM):

A Technical White Paper

Authors:
Christian Garcia-Arellano
Adam Storm
Colin Taylor

A. Introduction

The Self-Tuning Memory Manager (STMM) solves two key problems with memory tuning: 1) an optimal memory configuration can be difficult to determine, and 2) any static memory configuration will be sub-optimal in the presence of dynamic workloads. The feature works by iteratively modifying the memory configuration in small increments with the goal of improving overall system performance. STMM makes its decisions with the help of new internal metrics that predict the effect that additional memory will have for a given heap. These metrics, when combined with STMM's advanced tuning algorithms, can in most cases, tune a system from an out-of-the-box configuration to near-optimal memory usage in an hour or less. This approach is well suited for industries where workload memory requirements can change dramatically over time.

STMM improves performance and dramatically reduces the database's total cost of ownership for small and large businesses alike. Internal testing shows that it is effective at tuning database memory in performance-sensitive environments, with complex workloads, and in the presence of fluctuating system resource availability. It works well in single-partition and homogeneous multi-partition environments, providing fast convergence time, rapid adaptation, and stable response to noise. In the vast majority of cases, even for steady-state workloads, STMM competes with the best tuning of human administrators. The feature also has a rich set of logged data, which allows the user to gain insights into how tuning decisions are being made.

In general, STMM will be useful to the vast majority of DB2 9 users. The feature is most useful in shops where knowledge of the DB2 memory model or where overall DBA skills are limited. In shops where DB2 skills are plentiful, the feature can still be extremely useful when dealing with a new workload with unknown memory requirements or when a workload has volatile memory requirements.

In environments where the workload has stable memory requirements, and skill levels in DB2 performance tuning and administration are particularly high, STMM can still be useful as a diagnostic tool. In these environments, it is best to run STMM on a test system while closely monitoring the tuning decisions. Then, before going into production, the configuration can be fixed (by turning STMM off) to ensure that the configuration doesn't change.

STMM is also particularly useful in Database Partitioning Feature (DPF) environments where each partition has similar memory requirements. In these environments, STMM is able to tune all partitions at the same time. If each partition in a DPF system does not have similar memory requirements, STMM can still be used if set up properly (as detailed later in this document).

Similarly, on systems with many defined buffer pools, STMM can save a great deal of time and effort by automatically tuning the memory. Not only does STMM utilize an advanced tuning algorithm for determining buffer pool sizes, but it is also able to adjust the memory configuration up to 120 times each hour. These two factors combined allow STMM to reach a near optimal configuration dramatically faster than a human administrator.

Which parameters can STMM configure?

STMM can automatically tune the following parameters:

- DATABASE_MEMORY – total database memory usage
- MAXLOCKS – percentage of locklist granted to any given transaction
- LOCKLIST- memory available for row or table locks
- SHEAPTHRES_SHR – total amount of memory available for sorting
- SORTHEAP – memory available for each sort
- PCKCACHESZ – memory for compiled SQL package caching
- Buffer pool size – memory for caching data and index pages

When tuning the total database memory usage (the DATABASE_MEMORY parameter), STMM is able to balance the system's memory across multiple DB2 databases. In addition, STMM will also sense the memory requirements of other applications running on the system (even non-IBM applications) and will tune the memory accordingly so that the database's memory consumption will not negatively impact the other applications.

Additionally, STMM tunes the amount of free database memory that can be used when any of the following heaps run out of memory:

- UTIL_HEAP_SZ – memory used for running utilities such as backup and load
- DBHEAP – memory used for database control blocks
- CATALOGCACHE_SZ – memory used for caching the system catalogs

As a result of this overflow tuning, STMM is able to implicitly tune these parameters as well. This implicit tuning does not eliminate the need to suitably configure these parameters. However, it will help to prevent out-of-memory errors in cases where the memory usage for a given heap spikes unexpectedly.

Currently, STMM cannot be used to tune Application Global Memory. The same is true for Agent Private Memory and Agent/Application Shared Memory, with the exception of the SORTHEAP parameter (listed above). To enable the SORTHEAP parameter for tuning, the database manager SHEAPTHRES parameter must be set to 0.

Enabling and disabling STMM

The STMM feature is automatically enabled for all newly created, non-partitioned DB2 9 databases. If your database is a partitioned database or has been upgraded from a previous DB2 version, the feature will have to be enabled manually.

To enable the STMM feature, turn the SELF_TUNING_MEM database configuration parameter to ON:

```
update database configuration using SELF_TUNING_MEM ON
```

The SELF_TUNING_MEM database configuration parameter is the main way to enable and disable the STMM feature. In addition to setting the value of this parameter, each memory heap that will be tuned by STMM must be independently activated for tuning.

To disable the STMM feature, set the SELF_TUNING_MEM database configuration parameter to OFF:

```
update database configuration using SELF_TUNING_MEM OFF
```

If the SELF_TUNING_MEM configuration parameter is changed using the described commands (i.e., using an established database connection), it takes effect immediately so no database restart is necessary.

Enabling tuning for a parameter

With SELF_TUNING_MEM set to ON, setting any of the self-tunable parameters to AUTOMATIC allows the parameter to be tuned by STMM. Because memory is being traded between memory consumers, there must be at least two memory consumers enabled for self-tuning in order for memory tuning to occur. When SELF_TUNING_MEM is set to ON, but fewer than two memory consumers are enabled for self-tuning, memory tuning will not occur. The exception to this is the sort heap memory area, which can be tuned whether or not other memory consumers are enabled for self-tuning.

To view the current setting for the SELF_TUNING_MEM parameter, use the GET DATABASE CONFIGURATION command and specify the SHOW DETAIL option. The possible values for the parameter are:

```
Self Tuning Memory                          (SELF_TUNING_MEM) = OFF
Self Tuning Memory                          (SELF_TUNING_MEM) = ON (Active)
Self Tuning Memory                          (SELF_TUNING_MEM) = ON (Inactive)
```

If the parameter shows ON (Active), the memory tuner is actively tuning the memory on the system. If the parameter shows ON (Inactive), it means that although the parameter is set ON, self-tuning is not occurring because fewer than two memory consumers are enabled for self-tuning.

The following command can be used to enable tuning for the DATABASE_MEMORY parameter:

```
update database configuration using DATABASE_MEMORY AUTOMATIC
```

For the buffer pools, the setting of the size to AUTOMATIC is done using either the `create bufferpool` statement or the `alter bufferpool` statement. This is an example of the `alter bufferpool` syntax used to enable tuning for an existing buffer pool:

```
alter bufferpool <bufferpool_name> size AUTOMATIC
```

This command must be followed by a commit; the change takes effect after the transaction is committed.

When setting a parameter to AUTOMATIC, a starting size can be optionally specified. This can be done in the buffer pool case using the following command:

```
alter bufferpool <bufferpool_name> size <new_size> AUTOMATIC
```

In this case, the buffer pool will start tuning from an initial size of 1000 pages.

Similarly, in the case of a configuration parameter, the following command can be used:

```
update database configuration using DATABASE_MEMORY <new_size> AUTOMATIC
```

In this case, the DATABASE_MEMORY parameter will start tuning from an initial size of <new_size> pages.

Disabling tuning for a parameter

Alternatively, setting a configuration parameter to MANUAL will prevent automatic tuning of the parameter and fix the parameter's size at the current value. The following command can be used to disable tuning for the PCKCACHESZ parameter:

```
update database configuration using PCKCACHESZ MANUAL
```

Note that there is no MANUAL command for the buffer pools. Instead, you can disable tuning for a buffer pool by setting its size to a numeric value. This is an example of the `alter bufferpool` syntax used to disable tuning for an existing buffer pool:

```
alter bufferpool <bufferpool_name> size <new_size>
```

B. New or Changed in DB2 9

The new COMPUTED keyword

Prior to DB2 9, it was possible to set DATABASE_MEMORY to AUTOMATIC. This notion of AUTOMATIC was very different from the current AUTOMATIC behavior (where STMM automatically sizes the database shared memory set based on free physical memory). Prior to DB2 9, if the user set DATABASE_MEMORY to AUTOMATIC, the database would *compute* the size of the database memory by summing up the heaps that are allocated out of the database shared memory set and then allocate a statically sized shared memory set. If you want to maintain pre-DB2 9 AUTOMATIC behavior, you should set DATABASE_MEMORY to COMPUTED. All databases that are upgraded to DB2 9 from previous DB2 versions will have the AUTOMATIC setting converted to COMPUTED.

The new DB_MEM_THRESH configuration parameter

The new DB_MEM_THRESH configuration parameter specifies how DB2 9 handles excess unused database shared memory. Typically, as pages of memory are touched by a process, they are committed, meaning that a page of memory has been allocated by the operating system, and occupies space in physical memory and/or in a page file on disk. Depending on the database workload, there may be peak database shared memory requirements at a certain time of day. However, this means that the amount of database shared memory that has been committed by DB2 9 will stay at that peak requirement.

This new database configuration parameter represents the maximum percentage of committed, but currently unused, database shared memory that DB2 9 will allow before starting to release committed pages of memory back to the operating system. Acceptable values are whole numbers in the range of 0 to 100. The value 0 means disclaim all unused database shared memory, while the value 100 means never disclaim any unused database

shared memory. The default value is 10, which means disclaim memory when more than 10% of database shared memory is unused, and should be suitable for most workloads.

Care should be taken when updating this parameter because setting the value too low could cause excessive memory thrashing on the box (memory pages constantly being committed and then disclaimed), and setting the value too high may prevent DB2 9 from releasing any database shared memory back to the operating system for other processes to use. The configuration parameter is ignored (meaning that unused database shared memory pages will remain committed) if the database shared memory region is pinned through the DB2_PINNED_BP registry variable, configured for large pages through the DB2_LARGE_PAGE_MEM registry variable, or if disclaiming of memory is explicitly disabled through the DB2MEMDISCLAIM registry variable.

Sort memory configuration changes in DB2 9

Several changes were made to the sort memory configuration in DB2 9, two of which are noteworthy if you plan to use the STMM feature.

The SHEAPTHRES_SHR parameter represents a limit on the total amount of database shared memory that can be used by sort memory consumers at any one time. The first change is that this limit was a hard limit before DB2 9, and is now a soft limit. This change to a soft limit allows the sort shared memory heap to consume additional, unreserved, database shared memory if needed.

The other significant change is that in DB2 Version 8, sorts only used shared memory if INTRA_PARALLEL was set to ON or if the connection concentrator was enabled. In DB2 9, if the user sets the SHEAPTHRES database manager configuration parameter to 0, all sorts and other sort memory consumers will use shared memory (bound by the SHEAPTHRES_SHR parameter). This setting is a requirement for enabling tuning of the sort memory because the memory tuner only trades memory between memory consumers in database shared memory. Having all sorts run in shared memory is enabled by default for newly created DB2 9 instances, but note that databases upgraded from previous versions will maintain their previous value of the SHEAPTHRES database manager configuration parameter. To revert to the DB2 Version 8 behavior (where some sorts run in private memory and some in shared memory), the user must set the SHEAPTHRES database manager configuration parameter to a value greater than 0.

Setting DATABASE_MEMORY to AUTOMATIC in DB2 9

If the DATABASE_MEMORY parameter is set to AUTOMATIC, the memory tuner will automatically adjust the amount of database shared memory allocated based on the amount of free physical memory available on the system at the time. The amount of physical memory left free by a given DB2 database depends not only on the amount of physical memory on the system, but also on the database's need for memory relative to the other active databases. When a given database is deeply in need of additional memory, it will maintain a small amount of free physical memory. Conversely, when a database has no need for additional memory, it will maintain a larger amount of free physical memory. This method allows databases to cooperate in the distribution of the system memory.

We also note that a given database will not allocate more system memory unless it recognizes that it will benefit from the additional memory. If the database will not benefit from additional memory, even if there is a large amount of memory free, the database will not take more memory. Similarly, if a database has increased its memory consumption

because of a need for more memory and then this need for memory decreases, the database will not free up any memory until there is a demand for memory by another database or application on the system.

Setting DATABASE_MEMORY to a numeric value in DB2 9

If you want to specify the amount of memory that a given database should consume, you should set DATABASE_MEMORY to a numeric value. This value, in 4KB pages, represents the maximum amount of memory that the database will be allowed to consume. If the value for DATABASE_MEMORY is larger than the sum of the heaps that comprise the database shared memory set, then the heaps that are set to AUTOMATIC will be able to grow by consuming some of the unused memory. If, however, the DATABASE_MEMORY value is smaller than the sum of the heaps in the set, the AUTOMATIC parameters will be scaled down in an attempt to respect the setting of DATABASE_MEMORY.

The setting of DATABASE_MEMORY to a numeric value is most useful in cases where the administrator knows how much memory should be devoted to a given database to achieve suitable performance. The setting is also recommended on platforms that do not support the AUTOMATIC setting of DATABASE_MEMORY because it allows the AUTOMATIC heaps to grow up to the limit imposed by the specified value.

C. STMM With Other DB2 Features

STMM and the Configuration Advisor

The first step in using STMM is to define the starting configuration for the memory heaps to be automatically tuned. The starting configuration will help STMM get to the optimal configuration faster, but it is not a requirement because STMM is able to get to the optimal configuration even if it is started from the default configuration values.

If STMM is going to be activated on a previously tuned system, then the existing configuration should be a good starting point. For new systems, the best method for determining an appropriate initial configuration is to use the Configuration Advisor.

The Configuration Advisor is accessible through a Command Line Processor (CLP) command, an API, or the Control Center, and it is also automatically invoked as part of the database creation in DB2 9. After just a few seconds of computation, it sets all the important memory configuration parameters and buffer pools by surveying the underlying resources such as the number of CPUs, the memory available, and the number and speed of the physical disks. The tool will tune most workloads very well, and can tune static OLTP workloads as well as performance experts.

When running the Configuration Advisor, keep in mind the following two considerations. First, the configuration produced by the advisor is static, which is the main reason why it can only be used to generate an initial configuration for STMM. If at any time there is a change in the workload or underlying resources, the configuration generated by the advisor may no longer be valid. Second, the configuration recommended by the advisor depends greatly on the description of the workload given by the user. An inaccurate workload description will result in a sub-optimal initial configuration.

STMM with HADR

When STMM is activated on a HADR system, memory tuning will only occur on the primary server. During takeover, when the secondary server becomes the primary server, STMM will be started on the new primary node (the former secondary node) and memory tuning will begin. At the same time, tuning will be stopped on the new secondary node (the former primary node). As a result, it may be advantageous to allow each server to run as the primary for a period of time before entering into production. This will ensure that both servers are suitably configured from a memory perspective and will ease the memory tuning process after takeover occurs.

STMM with DPF

When using STMM in partitioned database environments, there are a few factors that determine how well the feature will tune the system.

Enabling STMM on a DPF system

To enable STMM in a DPF system, first set SELF_TUNING_MEM to ON on all partitions. Then set the desired configuration parameters and buffer pools to AUTOMATIC on all partitions. Finally, for STMM to tune the memory of a DPF system, you must explicitly activate it on all partitions. You can do this with the following command:

```
activate database <database_name>
```

How STMM works on a DPF system

When STMM is enabled in partitioned databases, a single database partition is designated as the *tuning partition*, and all memory-tuning decisions are based on the memory and workload characteristics of that database partition. Once tuning decisions are made on the tuning partition, the new memory configuration is distributed to all other database partitions to ensure that all database partitions maintain similar configurations. In the case where there are only logical partitions (i.e., multiple database partitions running on the same physical machine), STMM assumes each partition has the same memory requirements. Conversely, if there are physical partitions (database partitions running on separate physical machines), STMM assumes each physical machine contains the same number of logical partitions (or that the physical memory to logical partition ratio is the same for all physical machines).

If all partitions in the system are uniform (i.e., they are each running similar workloads and have similar memory requirements), then STMM will work properly across all partitions. In this case, the configuration of each database partition will be the same as it would have been had there been only a single partition. If, however, at least two partitions in the system have different memory requirements, STMM must be enabled on some partitions and disabled on others in order to function properly. Since there is only one tuning partition for each database, enabling STMM on the largest subset of similar partitions will alleviate the greatest amount of tuning effort.

In partitioned database environments, the SELF_TUNING_MEM configuration parameter will only show ON (Active) for the database partition on which the tuner is running. On all other partitions SELF_TUNING_MEM will show ON (Inactive) or OFF (if tuning has been disabled on the given partition). As a result, to determine if memory tuner is active in a partitioned database, the SELF_TUNING_MEM parameter must be checked on all database partitions.

Choosing a tuning partition

As described above, as STMM tunes more partitions in a DPF system, the amount of manual tuning required decreases. As a result, it is desirable to have a large subset of partitions that STMM is able to tune. In general, most DPF systems have partitions that are either catalog partitions, data partitions, or coordinator partitions, and each of these partition types have differing memory requirements. As a result, at any given time it is advisable to configure STMM such that only partitions of the same type are being automatically tuned.

Since there is only ever one catalog partition in a DPF system, this would generally be a bad choice for the tuning partition since only one partition will be tuned automatically. As a result, choosing a tuning partition usually involves a choice between a data or coordinator partition. Since, in most DPF systems, there are more data partitions than coordinator partitions, the best choice for the tuning partition will likely be a data partition.

If no action is taken on the part of the user, STMM will automatically choose a tuning partition in an attempt to maximize the number of partitions that can be properly tuned. To be eligible, a partition must not be the catalog partition; it must not constitute a single-partition partition group and it must contain data. The eligible partition with the most buffer pools is made the tuning partition. In the case of a tie, the lowest numbered partition is chosen.

Updating the tuning partition

Even though there is an algorithm to automatically determine the STMM tuning partition, users are able to override this decision by setting the *user-preferred* tuning partition value. This is done through a new command in the ADMIN_CMD stored procedure. The syntax of the new command is as follows:

```
CALL SYSPROC.ADMIN_CMD( 'update stmm tuning dbpartitionnum <db partition number>' )
```

See the documentation for more details on the ADMIN_CMD procedure.

The STMM tuner determines, on startup and periodically when running, whether the tuner is running on the *user-preferred* partition. If it is not running on the user-preferred partition, this indicates that the user recently updated the tuning partition value using the above command. When this is detected, the tuner will stop running on the given partition, update the current STMM partition number, and start up on the new partition.

Creating a non-tuned partition

Once a tuning partition is chosen (either automatically or manually) tuning must be disabled on all dissimilar partitions. This is done by setting the SELF_TUNING_MEM database configuration parameter on that partition to OFF.

Note that only the configuration parameters and buffer pools set to AUTOMATIC on the *tuning partition* will be tuned by the memory tuner since all tuning decisions are made based on STMM's configuration on the tuning node. Also note that if the SELF_TUNING_MEM database configuration parameter is set to OFF on the tuning partition, then no tuning will occur on any partition.

STMM and the Balanced Warehouse

Currently, STMM is not recommended in a Balanced Warehouse environment. Consult the Balanced Warehouse documentation for guidance on how to properly configure the database memory in that environment.

D. How STMM works

STMM works *iteratively* by making small adjustments to each tunable parameter and then collecting more data to determine if the adjustment made produced the simulated effect. Tuning the system with a series of small configuration changes rather than one large configuration change prevents large swings in system performance. Making small changes also allows STMM to ensure that the tuning changes are having the desired effect before large amounts of memory are transferred between consumers.

The tuning process occurs by alternating data collection and tuning cycles. In the data collection cycle, STMM maintains a given configuration while collecting data on the effect of altering the memory. In the tuning cycles, STMM uses the collected data to determine and set a new memory configuration.

The default data collection period is 180 seconds, and in most cases 20 tuning cycles should be sufficient to produce a reasonable configuration. As a result, for STMM to produce a reasonable system configuration, a workload must run on the system for at least an hour. Running a workload for a sustained period of time allows STMM to collect the data necessary to tune. On systems with large amounts of memory (greater than 50 GB), or systems that start from poor initial configurations, reaching the final configuration may take longer.

Adapting the tuning cycle

STMM is able to adjust its tuning cycle based on the running workload and can tune as quickly as every 30 seconds or as infrequently as every 10 minutes. If the workload is composed mostly of short transactions that are frequently repeated (i.e., an OLTP workload), STMM will choose a shorter data collection period and configuration changes will occur more frequently. Conversely, if the running workload is composed of mostly long running queries (i.e., a DSS workload), the data collection period will be lengthened so that the collected data will be more representative of the entire workload and not just the portion of the workload that is running at a given time. Adapting the tuning cycle is one way that STMM uses to prevent tuning oscillations.

When the workload shifts

In the presence of a workload shift, STMM can quickly adapt to the memory requirements of the new workload. Tests conducted in the lab (mentioned later in this document) have shown that performance can begin to improve in the first tuning cycle with a near optimal configuration achieved in less than an hour in most cases.

Configuration persistence

All configuration changes made by STMM are written to disk so that when the database is shutdown, the configuration changes made by STMM are not lost. As a result, while determining an optimal configuration may take some time when STMM is first enabled, once

the configuration is determined, each database activation will start from the optimal configuration.

E. Experimental results

In this section, we discuss three experimental results that show different aspects of STMM tuning. In the first experiment, we compare STMM to a benchmark configuration of an industry standard OLTP workload. Through this experiment, we show how well STMM can tune a system with a static workload. In the second test, we test STMM on a system undergoing dramatic changes to its memory requirements. These tests show how STMM is able to adapt to changing memory requirements in a single database. In the last experiment, we show how STMM is able to tune the total amount of memory used by multiple databases sharing the same machine. This test shows how users can exploit the full power of their machines at all times by using all the available memory.

Tuning From the Default Configuration

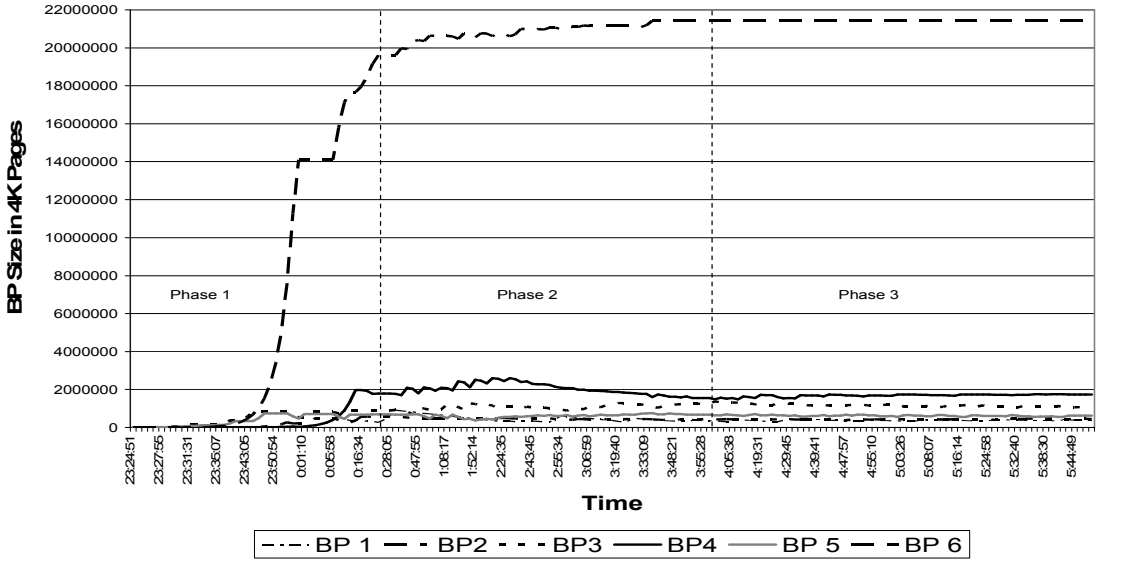
In evaluating a database memory tuning feature, the most convincing result would be to show that the tuner is able to take an out-of-the-box configuration and tune it to an "optimal" configuration in a reasonable amount of time. The main problem with conducting such a test is that, typically, there is no easy way to determine the optimal memory configuration for a given workload. In this case, we compare the results obtained by the memory tuner with those obtained using a benchmark configuration. We use a benchmark configuration because benchmark results produce the most highly tuned memory configurations, which can be considered "optimal" for this particular workload running on DB2 9. The benchmark configuration used was a result of a significant number of manual hours of tuning by a number of DB2 performance-tuning experts.

To test STMM using this new metric, we conducted experiments on an industry standard transaction-processing benchmark. The test system was configured to use 14 buffer pools, and we started each buffer pool at 1000 pages, which is the default size for newly created buffer pools in DB2 9. For these tests, sort, lock memory, and SQL query cache memory were not tuned since they are not relevant for this benchmark as its transactions are small (i.e., they use very little locking memory), have no sorts, and require very little package cache memory to run. When publishing a benchmark on this workload, it is always the buffer pool configuration that is most difficult to derive and it usually takes weeks of hand-tuning to finalize.

For these experiments, we ran the workload on a machine with 128 GB of physical memory, housing the 1.95 TB database.

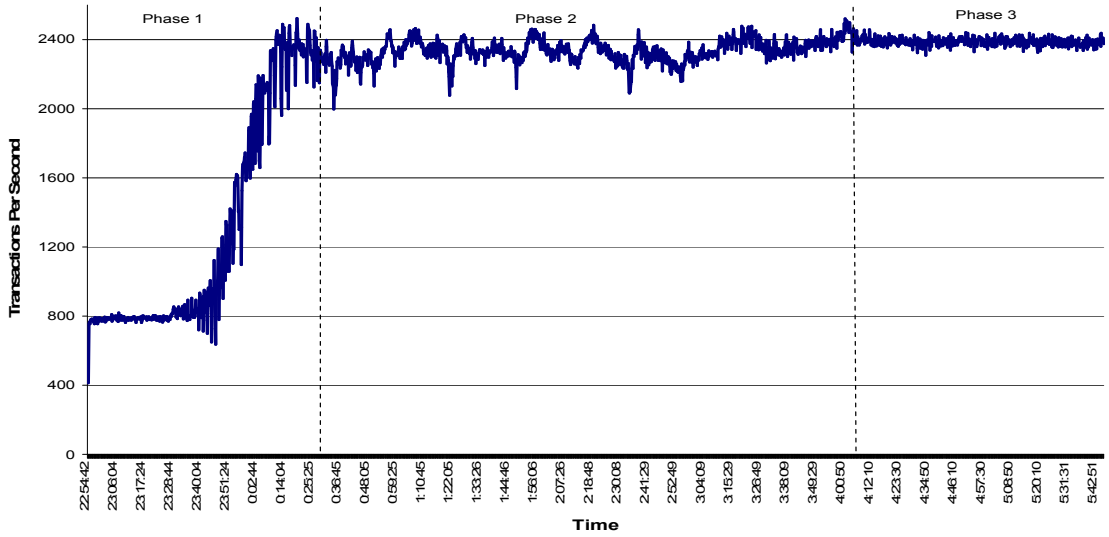
Figure 5 shows the tuning effect on the sizes of the six largest buffer pools during execution of the workload. The figure shows three phases of tuning. In the first phase, STMM takes the system from the default configuration to a configuration within 10% of the hand-tuned result. In the second phase of tuning, the buffer pools are finely tuned to arrive at the desired final configuration. Finally, in the third phase, STMM makes only very minor adjustments to the system.

Figure 5: Sizes of the six largest buffer pools during transaction processing workload



The workload performance during the run is shown in Figure 6. The performance of the system as illustrated in the figure can be seen in the same three phases. In the first phase, STMM takes the system from 783 transactions per second to 2,318 transactions per second. In the second phase, while STMM is fine-tuning the configuration, performance oscillates around 2,330 transactions per second. Finally, in the third phase, performance stabilizes at 2,385 transactions per second. This shows the dramatic impact that STMM can have on a workload; in this case, STMM improved performance by over 300%, most of which is achieved in the first hour and a half of tuning.

Figure 6: System performance during STMM tuning



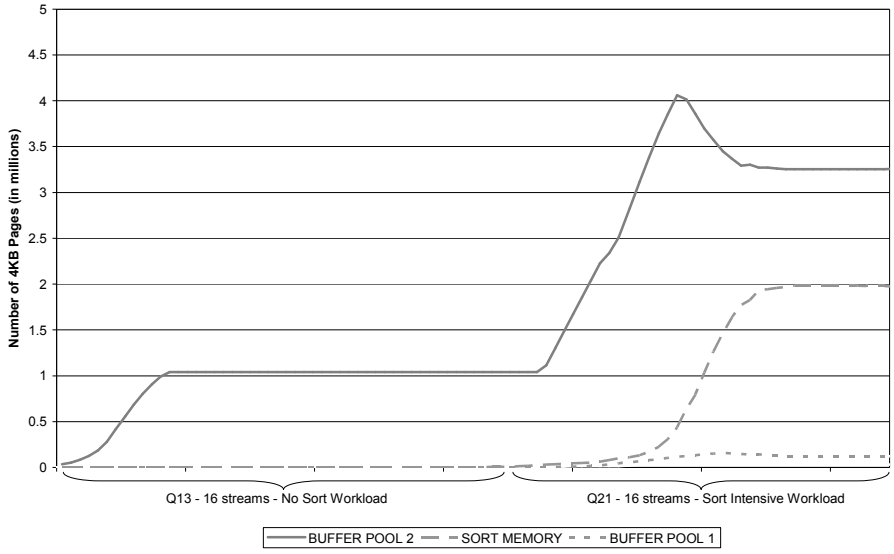
To determine how close the final configuration was to the hand-tuned result, we performed a second run using the final memory configuration and turning STMM off (also removing any small effect that tuning might have on the system). In this test, we found that the STMM-generated configuration resulted in an average transaction rate of 2,423 transactions per second compared to the baseline configuration of 2,419 transactions per second (a difference of 0.16%, which is within the inter-run variability of the workload on the test machine). From this second run with STMM off, it can also be observed that even with STMM actively tuning a system, the performance can be within 1.4% of the hand-tuned result.

Dramatic Workload Shift

One common problem with memory tuning arises from the fact that memory demands are not uniform throughout a typical day. For example, during regular business hours a database server may be processing simple transactions. Then, once the business day ends, the database will spend the next 8 hours running complex decision-support queries to provide data to be used for the next business day. This presents a challenge for an automated memory tuning system because the tuner must be able to quickly shift the memory to where it is most needed.

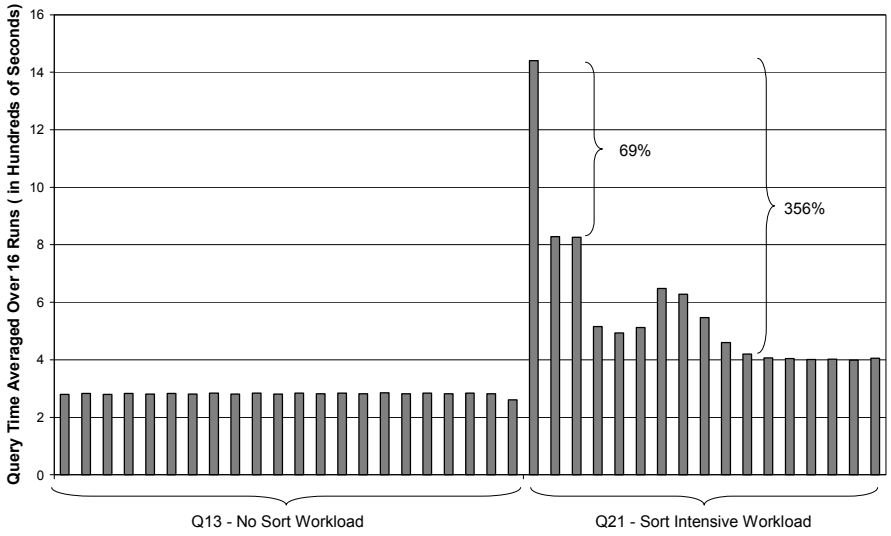
To simulate such an environment, we conducted an experiment where the database began by running one type of query and then, once the memory configuration stabilized, the workload shifted to more complex queries. At first we ran 16 concurrent streams of one of the queries in an industry standard decision-support testing workload with low requirements for sort memory (Q13). Then, once the memory configuration stabilized, we changed the workload to 16 concurrent streams of another query of the same workload, which is substantially more complex, contains multiple sub-queries, and has much higher requirements for sort memory (Q21). This shift from query Q13 to query Q21 places considerable pressure on the sort memory and should force the memory to be dramatically reallocated.

Figure 7: Workload Shift - Memory Distribution



In Figure 7, we can see the memory distribution shift over the course of the run. Once the streams of query Q13 stop and query Q21 starts running, we see a dramatic increase in the amount of sort memory allocated to the database. By the time the system has converged, the database has reserved more than 8 GB of memory for sorting. As Figure 8 shows, this memory distribution shift has a dramatic effect on the workload performance.

Figure 8: Workload shift query performance



In the first stage of the run, we can see that the memory distribution is stable as the 16 streams of query Q13 complete consistently in around 281 seconds. Once the workload shifts, however, it is clear that the system's memory is not properly configured for query Q21. At this point, STMM begins redistributing the database memory and the resultant dramatic effect on performance can be observed as quickly as the second run of the queries; by this time, performance has already improved by 69%. After several more runs of the query, response time stabilizes and a performance improvement of 356% can be observed when compared to the first execution of query Q21. This not only shows how critical sort memory can be to a database system, but also how effective STMM can be at supplying the sort memory when necessary.

Tuning multiple databases

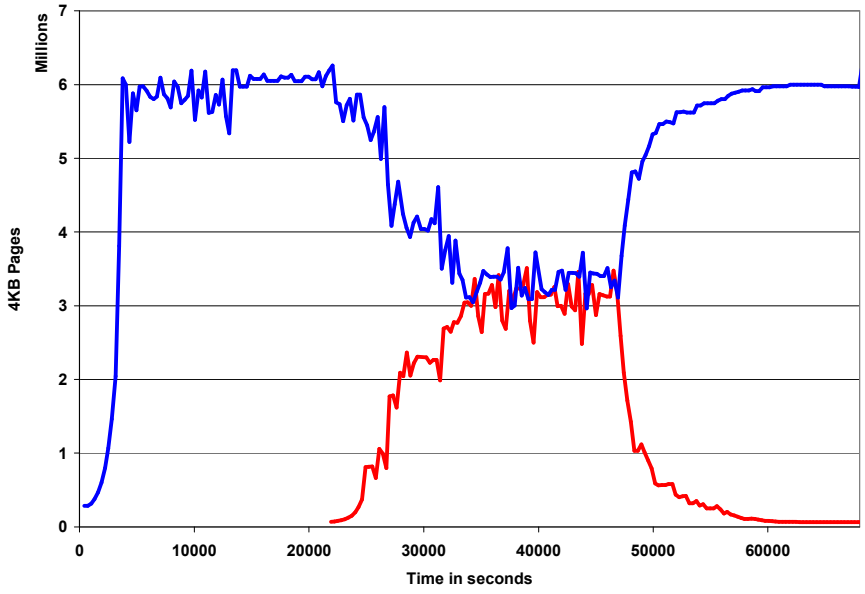
One difficult issue database administrators face when tuning memory is determining the total amount of system memory to dedicate to a given database. The problem is compounded when the system administrator is dealing with multiple databases, each of which may run at different periods of time in a single 24-hour window. If each database is configured with a static amount of memory, which is commonly the case, a good portion of the system memory will be unutilized during periods where one or more of the databases are not active.

To test STMM in an environment where multiple databases are competing for a single system's memory, we conducted an experiment with two identical databases running the same workload. In building the databases, it was necessary to ensure that both databases had the same physical design, resided on the same number of disks, and that the disks

were of the same speed, since even the slightest difference in any of these variables could have skewed the memory requirements for the databases. The tests were run on a machine with 8 processors and 32 GB of physical memory. Each of the 2 databases was loaded with 15 GB of raw data. The workload being run by each of the databases consisted of 4 clients, each running 22 queries used in an industry standard decision-support workload.

Figure 9 shows the database memory usage for the two databases during the 20-hour run. The first database is activated with the default configuration and the workload is started. In the first hour, STMM gives all of the system memory to this database because there are no other applications running on the system. After six hours, the second database is activated with the default configuration and begins running the same workload. As expected, two hours later both databases are sharing the system memory equally. A few hours after the memory is evenly distributed, the second database stops running the workload but remains activated. The dramatic difference in relative database activity that follows causes STMM to take memory from the second database and give it back to the first database.

Figure 9: Total database memory tuning



F. Special Considerations

Tuning databases in separate instances

STMM is able to tune databases in separate instances the same way it tunes databases in the same instance. In fact, the "Tuning Multiple Databases" result presented in the previous section was a case of two databases in separate instances. No special considerations are required if the databases being tuned are in separate instances.

Related configuration parameters

Two sets of configuration parameters are related in that the setting of one parameter has a dependency on the other parameter. The first set of parameters is MAXLOCKS and LOCKLIST. When either MAXLOCKS or LOCKLIST is set to AUTOMATIC, the other parameter

must also be set to AUTOMATIC. This implies that when one of the parameters is not set to AUTOMATIC, neither must be set to AUTOMATIC.

The second set of parameters is SHEAPTHRES, SHEAPTHRES_SHR, and SORTHEAP. As mentioned above, before sort memory can be tuned, SHEAPTHRES must be set to 0. If SHEAPTHRES is not set to 0 and both SORTHEAP and SHEAPTHRES_SHR are set to AUTOMATIC, no tuning will occur. Additionally, when SHEAPTHRES_SHR is set to AUTOMATIC, SORTHEAP must also be set to AUTOMATIC. If SHEAPTHRES_SHR is not set to AUTOMATIC, SORTHEAP can still be set to AUTOMATIC and, if SHEAPTHRES is set to 0, the SORTHEAP value will be tuned within the limit specified by SHEAPTHRES_SHR.

Platform support

The support for setting DATABASE_MEMORY to AUTOMATIC (thus tuning the total amount of memory available to the database) is only available on AIX® and Windows® platforms. On other platforms (Linux®, Solaris, HP, etc.), DATABASE_MEMORY can only be set to COMPUTED or a numeric value.

File System Caching

Disabling file system caching for all table spaces that use a buffer pool that STMM will tune may improve system performance.

In some cases, particularly when LOBs or LONG data is involved, file system caching can help system performance. In most cases, however, file system caching can cause pages moving between disk and DB2 to be double cached – once by the operating system and once by a DB2 buffer pool. To avoid this double caching, you can disable the file system cache, thereby enabling non-buffered I/O. By moving to non-buffered I/O, you free up system memory that STMM can use to improve database performance. Additionally, by allowing STMM to tune the buffer pool serving table spaces for which you have disabled file system caching, you decrease the likelihood that the performance of the table space will be negatively impacted by the move to non-buffered I/O.

If you are going to take the step to disable file system caching for one or more table spaces, it is best to first move LOBs and LONG data into their own table space, and then disable file system caching for the remaining table spaces. You can disable file system caching by creating or altering a table space with the NO FILE SYSTEM CACHING clause.

G. Monitoring STMM

DB2 Tools

Database configuration

To retrieve the current values of the configuration parameters set to AUTOMATIC, use the `show detail` clause when issuing a "get database configuration" command. To use the `show detail` clause, you must first establish a connection to the database. This is an example of the result (only showing the relevant configuration parameters):

```
get database configuration show detail

Database Configuration for Database mydb1

Description                              Parameter          Current Value     Delayed Value
-------------------------------------------------------------------------------------------
Size of database shared memory (4KB)   (DATABASE_MEMORY) = AUTOMATIC(32224)  AUTOMATIC(25380)
Max storage for lock list (4KB)        (LOCKLIST)        = AUTOMATIC(12928)  AUTOMATIC(12928)
Percent. of lock lists per application (MAXLOCKS)        = AUTOMATIC(98)     AUTOMATIC(98)
Package cache size (4KB)               (PCKCACHESZ)      = AUTOMATIC(6185)   AUTOMATIC(6463)
Sort heap thres for shared sorts (4KB) (SHEAPTHRES_SHR)  = AUTOMATIC(9306)   AUTOMATIC(5115)
Sort list heap (4KB)                   (SORTHEAP)        = AUTOMATIC(122)    AUTOMATIC(122)
```

In the above output, "Current Value" is the value used by the currently running database. The "Delayed Value" is the value of the configuration parameter on disk, which will be applied the next time the database is stopped and restarted.

When a buffer pool size is set to AUTOMATIC, the NPAGES value in the buffer pools catalog view will be set to -2. In the following example of a query of the syscat.bufferpools view, you can see that BUFFERPOOL_16K is set to AUTOMATIC.

```
select distinct char(bpname,18) as bpname, bufferpoolid, npages from
syscat.bufferpools

BPNAME             BUFFERPOOLID NPAGES
------------------ ------------ -----------
IBMDEFAULTBP                  1      136730
BUFFERPOOL_16K                2          -2
```

To retrieve the current size of a buffer pool set to AUTOMATIC, use db2pd or the Snapshot Monitor. This is the db2pd output:

```
$ db2pd -database MYDB1 -bufferpools

Database Partition 0 -- Database MYDB1 -- Active -- Up 0 days 01:00:56

Bufferpools:
First Active Pool ID      1
Max Bufferpool ID         1
Max Bufferpool ID on Disk 1
Num Bufferpools           5

Address            Id   Name                PageSz     PA-NumPgs  BA-NumPgs
      BlkSize    NumTbsp    PgsToRemov CurrentSz  PostAlter  SuspndTSCt
...
0x070000003044F940 1    IBMDEFAULTBP        4096       30445      0
      0          4          0          30445      30445      0
...
```

Here is the Snapshot Monitor output:

```
$ db2 get snapshot for bufferpools on MYDB1


             Bufferpool Snapshot

Bufferpool name                            = IBMDEFAULTBP
Database name                              = MYDB1
Database path                              =
                                 /home/colint/colint/NODE0000/SQL00001/
Input database alias                       = MYDB1
Snapshot timestamp                         = 05/18/2007 12:07:13.684986
...
Node number                                = 0
Tablespaces using bufferpool               = 4

Alter bufferpool information:
 Pages left to remove                      = 0
 Current size                              = 30445
 Post-alter size                           = 30445
```

Database memory

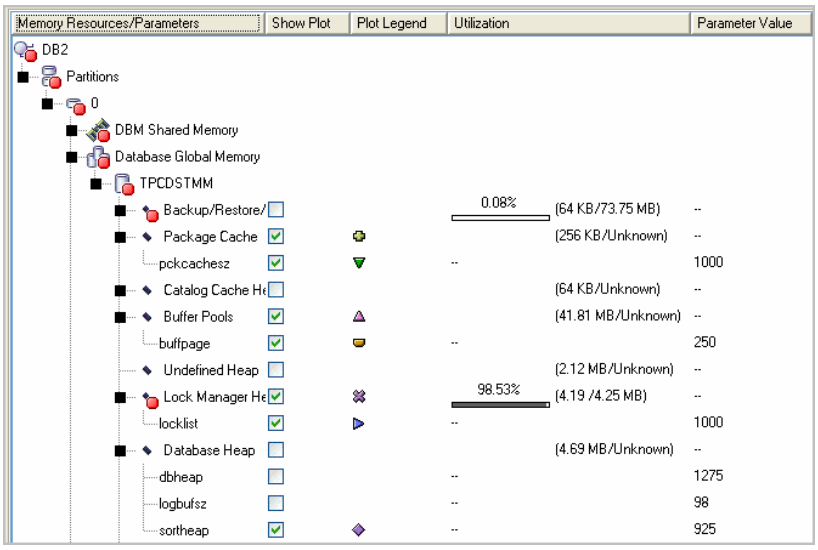
The DB2 tools for monitoring the sizes of the memory heaps are db2pd, the Snapshot Monitor, the DB2 Memory Tracker and the DB2 Memory Visualizer. These tools allow the user to know the current size of the memory heaps. Here is a db2pd example:

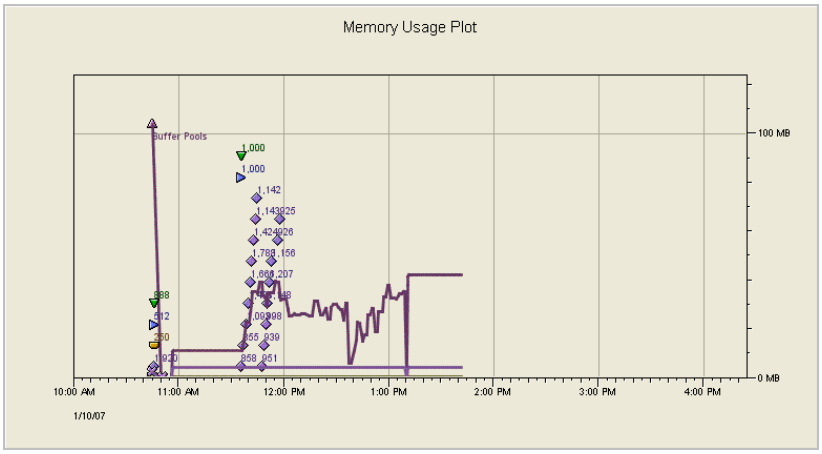
```
$ db2pd -database MYDB1 –mempools

Database Partition 0 -- Database MYDB1 -- Active -- Up 0 days 00:19:45

Memory Pools:
Address            MemSet   PoolName   Id    Overhead   LogSz       ...
...
0x0700000030001650 MYDB1    pckcacheh  7     151744     1103562     ...
0x0700000030001130 MYDB1    bph        16    97728      130878528   ...
0x0700000030000EA0 MYDB1    bph        16    0          543616      ...
0x0700000030000C10 MYDB1    bph        16    0          281472      ...
0x0700000030000980 MYDB1    bph        16    0          150400      ...
0x07000000300006F0 MYDB1    bph        16    0          84864       ...
0x07000000300005A8 MYDB1    shsorth    18    0          12736       ...
0x0700000030000460 MYDB1    lockh      4     0          16401408    ...
0x0700000030000318 MYDB1    dbh        2     409696     19472159    ...
...
```

Here is a DB2 Memory Tracker example:

```
$ db2mtrk -d
Tracking Memory on: 2007/05/18 at 11:29:48

Memory for database: MYDB1

   utilh       pckcacheh   other       catcacheh   bph (1)     bph (S32K)
   64.0K       1.2M        192.0K      320.0K      125.4M      640.0K

   bph (S16K)  bph (S8K)   bph (S4K)   shsorth     lockh       dbh
   384.0K      256.0K      192.0K      192.0K      15.7M       19.4M
...
```

The DB2 Memory Visualizer provides the same information and can also be used to plot the recent memory heap size history.





The tuning partition in a DPF environment

The ADMIN_CMD procedure can be used to determine the user-preferred and current STMM tuning database partition number. The syntax of the new command is as follows:

CALL SYSPROC.ADMIN_CMD('get stmm tuning dbpartitionnum')

And this is an example of the result:

```
USER_PREFERRED_NUMBER  CURRENT_NUMBER
---------------------- --------------
                     1              1
```

If the two values in the above output are not equal, it indicates that the user has requested that the tuner move to a new partition but that request has not yet been processed. In general, the process will be carried out within 10 minutes of being issued. If the user wants to force the movement of the tuner to the new partition, this can be done by deactivating and reactivating the database, at which point the tuner will move immediately.

See the documentation for more details on the ADMIN_CMD procedure.

The STMM agent

If you want to monitor the STMM agent, you can obtain some rudimentary information through the `list applications` command. That command has the following output (with the STMM agent information bolded):

```
db2 list applications

Auth Id  Application   Appl.    Application Id                DB       # of
         Name          Handle                                 Name     Agents
-------- ------------- -------- ----------------------------- -------- -----
USER     db2wlmd       411      *LOCAL.DB2.070323132434       SAMPLE   1
USER     db2taskd      410      *LOCAL.DB2.070323132433       SAMPLE   1
USER     db2stmm       409      *LOCAL.DB2.070323132432       SAMPLE   1
USER     db2bp         408      *LOCAL.user.070323132429      SAMPLE   1
```

Operating System Tools

A number of operating system tools can be used to monitor the memory utilization, and can be useful when running DB2 9 with the DATABASE_MEMORY tuning feature of STMM enabled. Since this STMM feature is only available for the AIX and Windows operating systems, we include below a brief summary of some of the tools that can be used on each platform to monitor the free memory on the system and the usage of paging space.

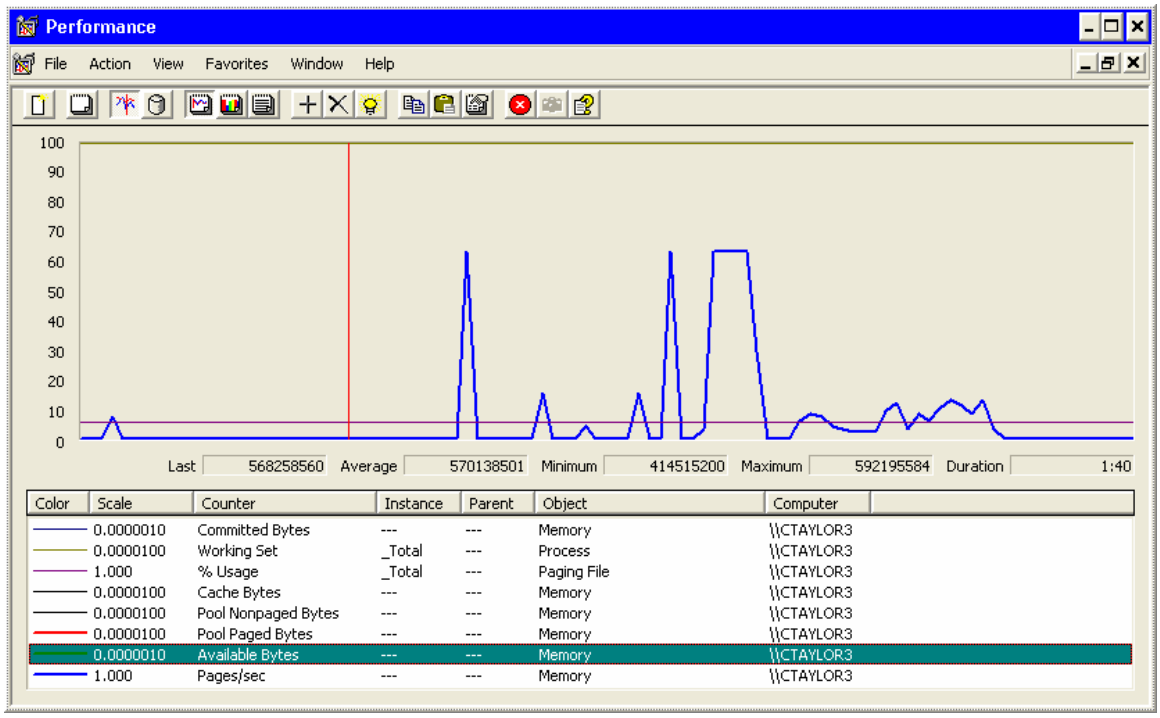
Virtual memory

The `vmstat` command is one of the most commonly used tools for monitoring virtual memory usage that is available on Linux and multiple UNIX platforms. In the case of AIX, it also reports statistics about kernel threads, disk activity, traps and CPU activity.

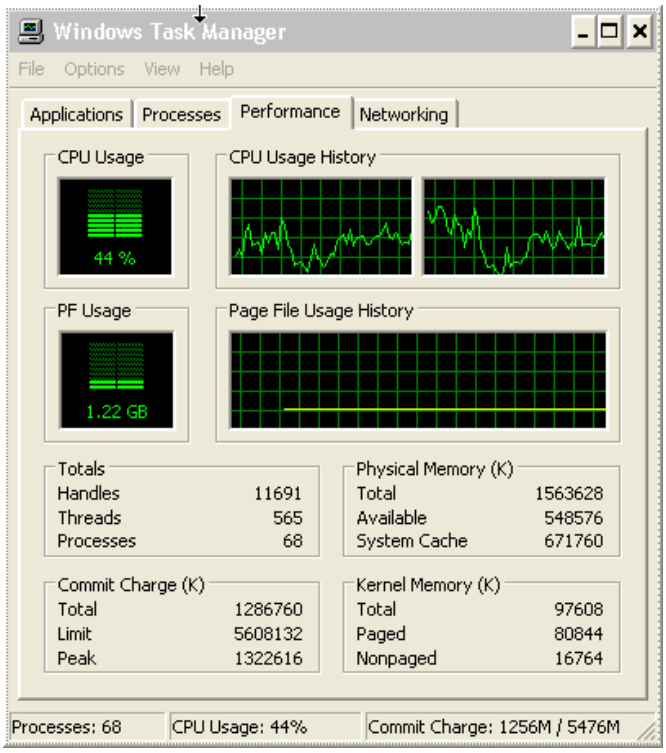
Of all the options available in the AIX version, option –v is the most interesting to use in conjunction with the DATABASE_MEMORY tuning functionality because it reports various statistics maintained by the Virtual Memory Manager. Further discussion of the file cache is outside the scope of this document; more details about this can be found in the white paper "Optimizing AIX 5L performance: Tuning your memory settings, Part 1" (http://www-128.ibm.com/developerworks/aix/library/au-aixoptimization-memtun1/).

```
$ vmstat –v | grep –e "memory pages" –e free –e perm –e "file pages"
4194304 memory pages
  66237 free pages
    5.0 minperm percentage
   15.0 maxperm percentage
    1.5 numperm percentage
  59669 file pages
```

On the Windows platform, the Performance Monitor can be used to monitor virtual memory.



The Windows Task Manager's Performance tab can also be used to monitor memory usage.

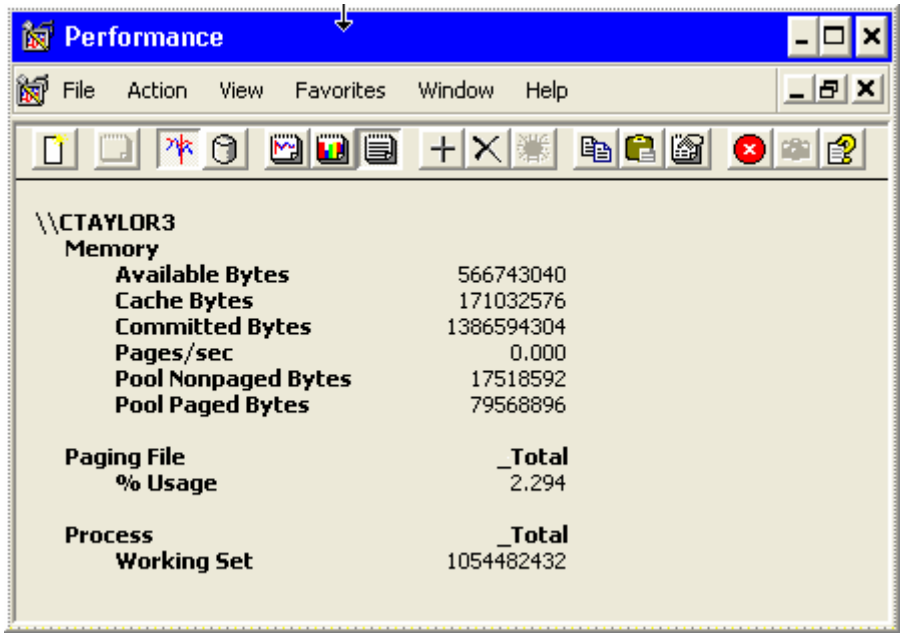


Paging space

On the AIX operating system, the paging space usage statistics can be seen using the lsps command. The -s flag displays the percentage of paging space allocated (reserved), whether the paging space has been assigned (used) or not.

```
$ lsps -s
Total Paging Space   Percent Used
      32768MB              3%
```

On the Windows platform, the Performance Monitor can also be used to monitor paging space.



Logging

The db2diag.log file

All changes made by STMM are logged in two places. Simple information on each configuration change can be found in the db2diag.log file. In this file, you will see records that indicate a configuration change such as this:

```
2006-10-17-19.10.00.912218-240 I408210A457        LEVEL: Event
PID     : 946302               TID  : 1           PROC : db2stmm (MYDB1) 1
INSTANCE: ewhhr                NODE : 001         DB   : MYDB1
APPHDL  : 1-52                 APPID: *N1.cgarciaa.060809150048
AUTHID  : CGARCIAA
FUNCTION: DB2 UDB, config/install, sqlfLogUpdateCfgParam, probe:20
CHANGE  : STMM CFG DB DEWHR000: "Sheapthres_shr" From: "109306" <automatic>
                                          To: "105115" <automatic>
```

You will also see records that indicate a buffer pool change such as this:

```
2006-10-17-19.03.58.672185-240 I395047A488        LEVEL: Event
PID     : 946302               TID  : 1           PROC : db2stmm (MYDB1) 1
INSTANCE: ewhhr                NODE : 001
APPHDL  : 1-52                 APPID: *N1.cgarciaa.060809150048
AUTHID  : CGARCIAA
FUNCTION: DB2 UDB, buffer pool services, sqlbAlterBufferPoolAct, probe:90
MESSAGE : Altering bufferpool “BUFFERPOOL_16K" From: “117268" <automatic>
                                    To: “109666" <automatic>
```

In addition to these log entries, the changes are also logged in much greater detail in the STMM logs, which are in a directory named stmmlog, in the same directory as the db2diag.log file. The STMM logs are meant for DB2 Support, to assist with problem determination.

IBM Canada
8200 Warden Avenue
Markham, ON
L6G 1C7
Canada

Published June 2007