Speicherverwaltung in Oracle leicht gemacht

Dr. Frank Haney
Consultant
Jena

Schlüsselworte:

Oracle Database, Administration, SGA, PGA

Einleitung

Jedes Starten einer Oracle-Instanz und jedes Login eines Benutzers weist diesen Arbeitsspeicher zu. Die Verwaltung dieser Speicherbereiche ist eine wichtige Aufgabe für den Datenbankadministrator (DBA). Auch wenn mittlerweile manche Automatismen verfügbar sind, sollte der DBA doch wissen, wozu die verschiedenen Bereiche gut sind, und wie man gezielt eingreifen kann, um Performanceengpässe zu beheben oder zu vermeiden. Im Beitrag sollen deswegen Funktion und Bedeutung der verschiedenen Speicherbereiche erläutert werden. Anschließend wird die Verwaltung mittels der einschlägigen Parameter beschrieben. Insbesondere wird auf die Frage nach den Möglichkeiten des automatischen Speichermanagements eingegangen, wie sie von Release zu Release immer umfangreicher implementiert wurden. Aus der Sicht von Linux soll dargestellt werden, wie sich Speichereffizienz betriebssystemseitig unterstützen läßt.

Speicherbereiche im Überblick

Die Oracle-Instanz besteht aus der *System Global Area* (SGA) und den Hintergrundprozessen. Die SGA ist der zusammenhängend aus dem Hauptspeicher des Servers allokierte Speicherbereich, letztlich die zentrale Schaltstelle des Datenbankmanagementsystems. Auf diesem Speicherbereich arbeiten sowohl die Vordergrundprozesse (Serverprozesse) für die Nutzer als auch die Hintergrundprozesse der Datenbankverwaltung. Im folgenden werden die verschiedenen Bestandteile der SGA kurz charakterisiert:

SGA-Bestandteil	Funktion
Immer vorhanden bzw. zwingend erforderlich	
Database Buffer Cache	Speicherbereich für Blöcke, die vom Server-Prozeß aus den Datendateien gelesen und vom Database Writer asynchron auf Platte geschrieben werden (LRU-Mechanismus)
Redo Log Buffer	Speichert *Before-Image*- und *After-Image*-Kopien von geänderten Daten, der Log Writer schreibt die Änderungen in die Redo Log Dateien, wird zyklisch wiederverwendet
Shared Pool	Speichert geparste Versionen von SQL-Anweisungen, PL/SQL-Prozeduren und Data Dictionary-Informationen (LRU-Mechanismus)
Reserved Shared Pool	Teil des Shared Pool, der für große zusammenhängende Speicheranforderungen dient
Java Pool	Repräsentation der Java-Methoden, -Klassen und -Objekte im Speicher
Fixed SGA	Interne Metadaten (X$-Tabellen als Basis der V$-Views)
Zusätzliche Metadaten	

Optional	
Large Pool	Optionaler Speicherbereich für I/O des RMAN, parallelen I/O etc. (kein LRU-Mechanismus)
Streams Pool	Wird von Oracle Streams verwendet
Recycle Buffer Cache	Für Blöcke, die schnell wieder aus dem Cache verschwinden können (LRU-Mechanismus)
Keep Buffer Cache	Für Blöcke, die möglichst im Cache bleiben sollen (LRU-Mechanismus)
*n*k Buffer Cache	Buffer Cache für Tablespaces abweichender Blockgröße
Result Cache	Speicherung der Resultate von SQL-Anweisungen (Bereich im Shared Pool)

Bemerkungen

- Hier wurden nur die Bereiche aufgeführt, die für den DBA zentrale Bedeutung haben.
- Nicht alle Bereiche sind über separate Parameter administrierbar, so z.B. die *Fixed SGA* oder die Aufteilung der Verwendung des *Shared Pool* als *Library Cache* und *Dictionary Cache*.
- Nicht immer handelt es sich um disjunkte Bereiche im SGA, so sind z.B. *Result Cache* und *Reserved Shared Pool* Teile des *Shared Pool*.
- Die optionalen Speicherbereiche nicht zu konfigurieren, heißt nicht in jedem Fall, daß die entsprechende Funktionalität nicht zur Verfügung steht. Ohne *Streams Pool* werden z.B. für die Streams-Replikation 10% der Größe des *Shared Pool* vom *Buffer Cache* abgezweigt.
- Viele Bereiche werden nach dem LRU-Mechanismus (Least Recently Used) verwaltet. Für den *Buffer Cache* handelt es sich eigentlich um eine Modifikation von LRU. Für das „Überleben“ eines Blocks im *Buffer Cache* ist nicht nur wichtig, wann er zuletzt, sondern auch wie oft er benutzt wurde. (Hier hat sich das Verhalten gegenüber älteren Versionen geändert.)

Eine wichtige Rolle für die Speicherverwendung spielt neben der SGA die *Program Global Area* (PGA). Das ist der Speicherbereich, der mit jeder Anmeldung bei der Datenbank den für die Benutzersessions arbeitenden Server-Prozessen zugewiesen wird. Die PGA hat folgende Aufgaben:

- Session-Informationen
- Cursor-Informationen
- Arbeitsbereiche für die SQL-Ausführung (Work Area)
- Sortierbereich (Sort Area)
- Hash-Join-Bereich (Hash Area)
- Bitmap-Erstellungsbereich (Bitmap Create Area)
- Bitmap-Zusammenführungsbereich (Bitmap Merge Area)

Die Größe der PGA beeinflußt in starkem Maße die SQL-Verarbeitung. Die PGA ist kein Bestandteil der SGA, sondern wird beim Standardverbindungsmodus (Dedicated Server = Jede Benutzer-Session hat einen eigenen Serverprozeß.) aus dem freien Speicher genommen. Bei zu vielen Sessions würde die Summe der PGAs mit dem zur Verfügung stehenden Arbeitsspeicher kollidieren, was zu gravierenden Engpässen bei der Performance führt. In diesen Fällen kann als Verbindungsmodus *Shared Server* konfiguriert werden. Der Speicher für die Sessions (UGA = User Global Area) wird dann aus der SGA, und zwar aus dem Large Pool, und wenn der nicht konfiguriert ist, aus dem Shared Pool genommen. Auf diese Weise läßt sich auch bei unzureichendem Session-Management die Menge des sessionbezogenen Speichers wirksam limitieren. Erwähnt werden soll noch, daß auch die Hintergrundprozesse privaten Arbeitsspeicher beanspruchen, also bei Abschätzungen zum erforderlichen Speicherbedarf nicht vernachlässigt werden dürfen.

Speicherverwaltung mit Initialisierungsparametern

Die Größe der genannten Speicherbereiche wird durch Initialisierungsparameter gesteuert. Der Trend ist dabei, einerseits immer mehr Parameter dynamisch einstellbar zu machen, so daß die Instanz bei Veränderungen nicht neu gestartet werden muß. Andererseits gibt es mittlerweile Automatismen, die die Konfiguration einzelner Bereiche überflüssig zu machen versprechen. Auf einige Aspekte soll im folgenden eingegangen werden, ohne daß hier Vollständigkeit auch nur angestrebt werden kann.

Es gibt aber nach wie vor Speicherbereiche, deren Konfiguration mit einem Neustart der Instanz verbunden ist, die also nicht dynamisch änderbar sind. Das wichtigste Beispiel ist der *Redo Log Buffer*, eingestellt mit dem Parameter LOG_BUFFER. Die Untergrenze sind mittlerweile 2 MB und der Standard ist abhängig von Anzahl der CPUs, SGA-Größe und 32- oder 64-bit Betriebssystem. Der Standard wirkt, wenn der Parameter nicht gesetzt ist. Allerdings ist in den neueren Releases das manuelle Setzen der statischen(!) Parameter von sehr begrenzter Reichweite. Ein Beispiel mit Linux 32-bit und Oracle 11.2.0.2:

```
Parameter nicht gesetzt:     Bytes allokiert    5615616
Parameter auf 2 MB gesetzt:  Bytes allokiert    5615616
Parameter auf 10 MB gesetzt: Bytes allokiert   14004224
```

Das Beispiel zeigt zwei Dinge:

- Unabhängig von den in der Dokumentation genannten Werten hängt es von der konkreten Serverkonfiguration ab, welche Minimalwerte sich realisieren lassen.
- Zusätzliche Speicherzuweisung erfolgt immer in einer Mindestgranularität.

Im allgemeinen macht es wenig Sinn, den Redo Log Buffer beliebig zu vergrößern. Es kann unter Umständen sogar kontraproduktiv sein. In der Regel ist die Tatsache, daß der Log Writer die Log-Einträge nicht schnell genug auf Platte schreiben kann, Ursache für bei der Log-Generierung entstehende Performanceprobleme, und nicht ein unzureichend dimensionierter Redo Log Buffer. Das manifestiert sich dann in den Warteereignissen *log file sync* und *log file parallel write*.

Die meisten Parameter sind mittlerweile dynamisch (zur Laufzeit) änderbar, sowohl für die privaten Arbeitsbereiche (PGA) als auch für die SGA. Auf eine ausführliche Darstellung soll hier verzichtet werden.
Viele Einstellungen kann Oracle jetzt auch automatisch je nach Bedarf ändern. Den Anfang machte die PGA. Seit Oracle 9*i* gibt es den Parameter PGA_AGGREGATE_TARGET. Dieser gibt eine gewünschte Obergrenze für die *Summe* aller PGAs an. Vorher konnte man den privaten Arbeitsspeicher für die Serverprozesse mit Parametern wie SORT_AREA_SIZE oder HASH_AREA_SIZE für den einzelnen Prozeß begrenzen. Das ist mit dem neuen Parameter flexibler zu handhaben, weil jedem Prozeß im Rahmen der für alle geltenden Obergrenze so viel PGA wie nötig zugewiesen werden kann. Auf diese Weise läßt sich verhindern, daß der für die Serverprozesse insgesamt verwendete Speicher durch Überallokation zu Performanceproblemen (Paging) führt. Aber Achtung: PGA_AGGREGATE_TARGET ist keine absolute Grenze. So werden z.B. PL/SQL-Tabellen im Speicher gehalten und gefüllt. Sie können nicht in einen temporären Tablespace ausgelagert werden. Bei extensivem Gebrauch dieses Features durch eine oder mehrere Sitzungen kann es dann doch zur Erschöpfung des Arbeitsspeichers kommen und eventuell sogar zur Auslagerung der SGA kommen, was dann für die Performance so etwas wie der Super-GAU ist.
Informationen über die Ausnutzung der privaten Arbeitsbereiche kann man sich über die Views V$SQL_WORKAREA und V$SQL_WORKAREA_HISTOGRAM verschaffen. Man sieht dort vor allem, welche SQL-Operationen (z.B. Sortierungen) komplett im Speicher (optimal), mit einmaligem (onepass) oder mehrmaligem (multipass) Auslagern in den temporären Tablespace abgelaufen sind.

Die View V$PGA_TARGET_ADVICE zeigt dann die Auswirkungen einer Vergrößerung oder Verkleinerung von PGA_AGGREGATE_TARGET. Dieser Ratgeber (Advisor) steht auch im Enterprise Manager zur Verfügung.

Oracle hat dann in 10*g* ein übriges getan und auch eine automatische SGA-Verwaltung ermöglicht. Das nennt sich *Automatic Shared Memory Management* (ASMM). Dazu muß der dynamische Parameter SGA_AGGREGATE_TARGET auf einen von 0 verschiedenen Wert gesetzt werden. (Dieser Parameter kann nicht größer sein als der statische Parameter SGA_MAX_SIZE.) Dann werden 5 wichtige SGA-Bestandteile innerhalb dieses Rahmens je nach Speicheranforderungen dynamisch und automatisch vom System geändert. Diese 5 Bereiche sind *Shared Pool*, *Buffer Cache*, *Large Pool*, *Streams Pool* und *Java Pool*. Natürlich gibt es jeweils systembedingte Untergrenzen, die auch das ASMM nicht unterschreiten kann. Das sind z.B. 48 MB oder 4 MB mal die Anzahl der CPUs (je nach dem was größer ist) für den Buffer Cache. Normalerweise sind die separaten Parameter (SHARED_POOL_SIZE, DB_CACHE_SIZE etc.) bei der Verwendung von ASMM auf 0 gestellt. Man kann sie aber auch auf einen positiven Wert stellen. Dieser ist dann eine absolute Untergrenze, die die dynamische Verwaltung nicht unterschreiten darf. Damit kann man den Overhead reduzieren, der durch die dynamische Umallokierung von Speicher entsteht. Die Neuzuweisung von Speicher zu einem Pool erfolgt immer in diskreten Einheiten, *Granule* genannt. Die Größe dieser Einheiten ist abhängig vom Oracle-Release und der Gesamtgröße der SGA, die mit SGA_MAX_SIZE angegeben wird. (Vgl. Support Note 947152.1) Für das gegebene System kann man sie mit folgender Abfrage ermitteln:

```
SELECT bytes FROM v$sgainfo WHERE name LIKE 'Granule Size';
```

Bei ASMM kann es durchaus zu Speicherproblemen kommen, die sich in einem ORA-04031-Fehler manifestieren. Nur zwei sollen hier genannt werden:

- Bestimmte SGA-Bereiche (Shared, Java, Streams und Large Pool) werden in Subpools genannte Unterbereiche aufgeteilt. Die Anzahl der Subpools hängt von der Anzahl der Prozessoren ab. So führen z.B. nach einem internen Algorithmus 16 Prozessoren zu 4 Subpools in jedem Pool. Das Maximum ist 7. Zusätzlich sind die Subpools von Shared und Streams Pool in 4 sogenannte Durations unterteilt. D.h., wir haben es mit bis zu 70 Bereichen zu tun, für die wenigstens ein Granule allokiert werden muß. Wenn dieses nun eine Größe von 512 MB hat, dann braucht man alleine für die genannten Bereiche mehr als 32 GB Speicher, damit die Instanz startet, ohne mit dem Fehler ORA-04031 abzubrechen.
- Der Fehler kann auch auftreten, wenn ein Granule z.B. vom Buffer Cache zum Shared Pool transferiert werden soll, aber nicht zur Verfügung steht.

Man sieht daran auch, daß eine beliebige Vergrößerung des SGA nicht unproblematisch ist. Mit Unterstützung des Oracle Supports lassen sich aber die Anzahl der Subpools und die Größe der Granules verändern. Dazu gibt es entsprechende verborgene Parameter.

Ein weiterer Vorteil von ASSM ist, daß der Server die aktuellen Werte in das SPFile schreibt. Man sieht das daran, daß die entsprechenden Parameter dort mit einem doppelten Unterstrich beginnen:

```
*.__db_cache_size=150994944
*.__java_pool_size=4194304
*.__large_pool_size=4194304
*.__shared_pool_size=243269632
*.__streams_pool_size=4194304
```

Die Instanz startet dann mit den zuletzt vom Server geschriebenen Werten. Wenn man im laufenden Betrieb SGA_TARGET=0 setzt, wird das ASMM ausgeschaltet. Die 5 vom ASMM verwalteten Bereiche werden auf ihren aktuellen Werten eingefroren.

Allerdings können nicht alle Speicherbereiche automatisch verwaltet werden. Folgende Bereiche sind nicht dem ASMM zugänglich:

- Log Buffer
- Recycle und Keep Buffer Cache
- Buffer Cache für die Tablespaces mit abweichender Blockgröße
- Fixed SGA und andere Metadaten

Die SGA-Größe, vermindert um diese Bereiche, stellt den Umfang an Hauptspeicher dar, der durch ASMM automatisch zugewiesen werden kann.

Mit Oracle 11*g* erfährt diese automatische Speicherverwaltung eine Erweiterung Jetzt kann der *gesamte* von Oracle verwendete Hauptspeicher automatisch verwaltet werden. Das nennt sich *Automatic Memory Management* (AMM) und ermöglicht die automatische dynamische Umverteilung von Arbeitsspeicher zwischen SGA und PGA je nach Anforderungen des aktuellen Workload zusätzlich zum ASMM. Dafür ist der neue Parameter MEMORY_TARGET (und analog auch der statische Parameter MEMORY_MAX_TARGET) eingeführt worden, der gewissermaßen das gemeinsame Dach von SGA und PGA ist. AMM ist eingeschaltet, wenn MEMORY_TARGET auf einen von 0 verschiedenen Wert gesetzt ist. PGA_AGGREGAT_TARGET und SGA_TARGET sind dann 0. Das Setzen dieser Parameter bei gleichzeitig gesetztem MEMORY_TARGET bedeutet Untergrenzen für die automatische Anpassung von SGA und PGA. Die verschiedenen Kombinationen, die sich beim Setzen der Parameter ergeben, sollen hier nicht näher erläutert werden. Auch hier gilt, wenn das AMM mit MEMORY_TARGET=0 ausgeschaltet wird, werden SGA und PGA auf ihren aktuellen Werten eingefroren, was sich an den entsprechenden Parameter ablesen läßt.
Den Prozeß der Umverteilung von Speicher kann auch überwacht werden:

- V$MEMORY_DYNAMIC_COMPONENTS – aktueller Status aller Speicherkomponenten.
- V$MEMORY_RESIZE_OPS – Historie der letzten 800 abgeschlossenen Speicherskalierungen.
- V$MEMORY_CURRENT_RESIZE_OPS – laufende Umallokierungen.

Analoge Views gibt es wegen der Abwärtskompatibilität und für den Fall, daß kein AMM, sondern nur ASMM benutz wird, auch für die automatische SGA-Verwaltung.
So wie schon für die PGA beschrieben, gibt es mittlerweile auch diverse Ratgeber (Advisor) für die Dimensionierung der SGA-Bestandteile und auch den für Oracle reservierten Speichers insgesamt. In der View V$MEMORY_TARGET_ADVICE z.B. sieht man, wie sich die Datenbankzeit für den aktuellen Workload bei einer Vergrößerung oder Verkleinerung von MEMORY_TARGET relativ zum Ist-Stand verändern würde. Natürlich gibt es die gleiche Information auch grafisch aufbereitet im Enterprise Manager.

Im SPFile gibt es bei der Verwendung von AMM analog zu den Parametern für die automatisch verwaltbaren SGA-Bestandteile auch zwei Parameter mit doppeltem Unterstrich für SGA und PGA: __SGA_AGGREGATE_TARGET und __PGA_AGGREGATE_TARGET. An dieser Stelle muß auf eine Merkwürdigkeit aufmerksam gemacht werden. Im Beispiel stellt sich das wie folgt dar:
MEMORY_TARGET steht auf 632291328 Byte, __SGA_AGGREGATE_TARGET zeigt den aktuellen Wert 415236096, __PGA_AGGREGATE_TARGET 218103808. Soweit so gut. Wenn man aber in SQL*Plus show sga abfragt, dann findet man für *Total System Global Area* 631914496, bzw. bei Abfrage der V$SGA eine Summe von *Variable Size* und *Database Buffers*, die wiederum in etwa den 631914496 entspricht. Offenkundig wird hier aller potentiell für die SGA zur Verfügung stehende Speicher als SGA-Größe angezeigt. Das ist zumindest irreführend, zumal V$MEMORY_DYNAMIC_COMPONENTS die richtigen Werte zeigt.

Speicherverwaltung aus der Sicht des Betriebssystems (Linux)

Hier soll am Beispiel von Linux erläutert werden, wie Systemeinstellungen die Speicherverwaltung beeinflussen. Drei Kernel-Parameter müssen dabei berücksichtigt werden:

- SHMMAX definiert die maximale Größe eines einzelnen Shared-Memory-Segments in Byte. Dieser Parameter sollte mindestens so groß wie die größte SGA im System sein. Empfohlen werden von Oracle für 32bit-Linux 4 GB -1 oder die Hälfte des zur Verfügung stehenden Arbeitsspeichers, je nachdem, was unter den gegebenen Umständen kleiner ist. Für 64bit-Linux empfiehlt Oracle generell die Hälfte des zur Verfügung stehenden Arbeitsspeichers.
- SHMMNI bezeichnet die maximale Anzahl von Shared-Memory-Segmenten systemweit. Der Standard von 4096 ist hier ausreichend.
- SHMALL gibt die Gesamtmenge an Shared Memory in Speicherseiten an und sollte wenigstens SHMMAX/PAGE_SIZE, aufgerundet auf eine ganze Zahl, betragen.

Verwaltet werden die Parameter als root-User in der `/etc/sysctl.conf` bzw. mit dem Befehl `sysctl`. So schreibt z.B.

```
# sysctl -w kernel.shmmax=2147483648
```

den entsprechenden Wert in die /etc/sysctl.conf.
Mit

```
# sysctl -p
```

werden die geschriebenen Werte für das aktuelle System aktiv gemacht.

Bei zu kleinen Werten von SHMMAX kann es dazu kommen, daß die SGA auf mehrere, eventuell nicht einmal zusammenhängende Shared-Memory-Segmente aufgeteilt wird. Die komplizierte Verteilung der SGA-Bestandteile auf mehrere Segmente ist ausführlich in Support Note 15566.1 beschrieben. Wenn Oracle beim Instanzstart ein Shared-Memory-Segment nicht allokieren kann, kommt es zu einer Fehlermeldung.
Man kann sich auch Informationen über die von Oracle allokierten Shared-Memory-Segmente verschaffen, entweder mit

```
sqlplus / as sysdba
oradebug setmypid
oradebug ipc
```

und Auswertung der in `<diagnostic_dest>/rdbms/<db_name>/<sid>/trace` (11*g*) geschriebenen Datei, oder mit dem Befehl

```
$ORACLE_HOME/bin/sysresv
```

direkt auf der Konsole anzeigen. Dort sieht man auch die Semaphoren, die grob gesprochen den Zugang der Prozesse (oder Threads) zu den Shared-Memory-Segmenten regulieren. Die entsprechenden Kernel-Parameter sollten mit dem Initialisierungsparameter PROCESSES korrelieren.

Hier ein kleines Beispiel:

SGA-Größe 603 MB
`kernel.shmmax = 209715200`

`$ORACLE_HOME/bin/sysresv` zeigt nach dem Neustart der Instanz 4 Shared-Memory-Segmente.

Bei `kernel.shmmax = 1610612736` ist es dagegen nur ein Shared-Memory-Segment.

Verwaltung großer SGAs mit Hugepages

Bei großen SGAs und vielen Serverprozessen (Benutzersessions) kann es durch die Verwaltung der Speicherseiten zu einem erheblichen Overhead kommen. Jeder Prozeß pflegt seine eigene Page Table, in der alle für die SGA allokierten Seiten verzeichnet sind. Für jede Speicherseite werden dafür 12 Bit benötigt. Die Standardspeicherseite hat in Linux 4 kB (4096 Byte). Bei einer SGA von 24 GB sind das 6.291.456 Seiten. Bei 100 angemeldeten Benutzern werden also 900 MB allein für die Page Tables benötigt. Man kann sich leicht ausrechnen, was das für noch mehr Sessions und noch größere SGAs bedeutet. In `/proc/meminfo` kann man das gut verfolgen. Allein das Starten der Oracle-Instanz erhöht bei einer SGA-Größe von reichlich 400 MB die Summe der Page Tables von 5144 auf 13120 kB. (Gut erklärt ist dieses Verhalten in Support Note 361323.1)
Sowohl der verbrauchte Speicher als auch der Overhead durch die Verwaltung der Page Tables kann somit zu einem gravierenden Problem werden. Besonders deutlich zeigt sich das bei 64-bit-Systemen, die bezüglich der SGA-Größe nicht den gleichen Limitierungen wie 32-bit-Systeme unterworfen sind. Was kann man da tun? Eine Variante wäre, die Anzahl der Sessions zu begrenzen. In Frage kommen die Einschränkung der gleichzeitigen Session für einen Benutzer durch ein in der Datenbank hinterlegtes Profil oder der Einsatz von Shared Server als Standardverbindung zur Datenbank.

Es gibt aber noch eine andere, weniger restriktive Methode. Das ist der Einsatz von *Hugepages*. Diese alternative Speicherverwaltung steht auf Unix-Systemen neben der Standardspeicherseite zur Verfügung, muß aber speziell konfiguriert werden. Bei Linux ist eine solche Speicherseite 2 MB groß.

Die Verwendung von Hugepages hat zwei Vorteile:

- Der für die Page Tables total benötigte Speicher reduziert sich drastisch. Im obigen Beispiel mit 24 GB SGA und 100 Sessions kämen nur noch rund 1,76 MB zusammen. Eine beeindruckende Differenz.
- Ein Nebeneffekt ist, daß der Paging-Mechanismus von Linux nur mit den üblichen 4-kB-Seiten arbeitet. Eine Hugepage kann also nicht ausgelagert werden. Damit läßt sich also wirksam verhindern, daß SGA ausgelagert (geswappt) wird.

Wie kann man nun der Oracle-Instanz die Verwendung von Hugepages beibringen? Zunächst einmal ermittelt man die Anzahl der benötigten Seiten. Man teilt die Summe aller SGAs durch 2 MB und addiert zur Sicherheit eine Seite. (Es macht keinen Sinn, mehr Hugepages allokieren zu lassen, als Shared Memory benötigt wird.) Es gibt ein Skript, mit dem sich die Anzahl der benötigten Hugepages ermitteln läßt: Siehe Support Note 401749.1. Die dort gefundene Zahl trägt man in der `/etc/sysctl.conf` für den Kernel-Parameter `vm.nr_hugepages` ein. Dann muß der Server neu gestartet werden, damit die SGA unter der Verwendung von Hugepages startet. Ob dem so ist, sieht man dann in `/proc/meminfo` am Verhältnis der Werte von `HugePages_Total` und `HugePages_Free`.

Es gibt aber ein paar Einschränkungen, die einen vorsichtigen Umgang mit Hugepages gebieten:

- Die Anzahl der konfigurierten Hugepages muß unbedingt größer sein als die SGA, weil sonst Hugepages nicht verwendet werden und beim Start der Instanz versucht wird, die SGA aus dem verbleibenden Speicher zu allokieren. Wenn das aus Mangel an Speicher nicht gleich fehlschlägt, kann es dann im Betrieb zu Performanceeinbußen und Fehlern kommen. Wenn `Pages_Total` und `HugePages_Free` gleich sind, dann werden keine Hugepages verwendet. Seit dem Patchset 11.2.0.2 gibt es einen neuen Initialisierungsparameter USE_LARGE_PAGES, mit dem man einstellen kann, ob die gegebene Instanz das Feature verwenden soll oder nicht.
- Hugepages funktionieren unter Linux nicht mit *Automatic Memory Management*. Das liegt daran, daß bei AMM die SGA allokiert wird, indem Dateien unter `/dev/shm` angelegt werden. Das ist nicht kompatibel zu Hugepages. (siehe Support Note 749851.1) Man muß sich also entscheiden, ob man die automatische Verwaltung mit AMM will oder die Vorzüge von Hugepages. Auch hier gilt aber, solange genügend Speicher vorhanden ist, wird sich die Instanz auch bei Verwendung von AMM starten lassen, allerdings ohne Hugepages zu benutzen. Mit ASMM sind Hugepages nutzbar.
- Das Setzen des Parameters USE_INDIRECT_DATA_BUFFERS=TRUE, wie es für die Allokierung großer SGAs (VLM) unter 32-bit-Linux notwendig ist, verhindert die Verwendung von Hugepages für den Buffer Cache. (siehe Support Note 829850.1) Die anderen SGA-Bestandteile können Hugepages nutzen.

Trotzdem ist der Einsatz von Hugepages immer eine Überlegung wert und in 64bit-Linux fast schon Pflicht.

Fazit

Man sieht also, daß trotz der möglichen Vereinfachungen und Automatisierungen, Speicherverwaltung ein weites Feld bleibt. Dabei ging es an dieser Stelle allein um die reine Verwaltung und nicht um Troubleshooting. Das würde bedeuten, sich mit der Vielfalt der mit der Speicherallokation zusammenhängenden Fehler und ihren Ursachen auseinanderzusetzen. Ein weiteres Thema, das hier weitgehend ausgeblendet wurde sind Performanceprobleme, die durch unzureichende Speicherkonfiguration entstehen. Das würde bedeuten, sich mit den speicherbezogenen Warteereignissen zu beschäftigen. Von Bedeutung für die Performance sind auch die internen Serialisierungsmechanismen in den Speicherbereichen, die mit den Begriffen Latch und Mutex verbunden sind. Auch diese mußten hier außen vor bleiben.

Kontaktadresse:

Dr. Frank Haney
Anna-Siemsen-Str. 5
D-07745 Jena

Telefon: +49(0)3641-210224
E-Mail: info@haney.it
Internet: http://www.haney.it