How to Configure x86 Memory Performance for Large Databases

Using Linux HugePages

by Ed Whalen

This article shows how to improve x86 memory performance for large databases by using Linux HugePages. Performance tuning of large database systems can be a challenge.

Published December 2013

Table of Contents

About Memory Performance

Performance tuning of large database systems can be a challenge. Depending on the operating system (OS) and hardware, there might be performance issues that are not readily detectableusing normal analysis methods such as AWR reports and OS tools such as `sar`, `top`, and `iostat`. Memory utilization in x86 environments is one of the issues that is not readily identifiable, but it can result in significant performance improvements if properly analyzed and configured.

Now more than ever, with systems that have larger amounts of memory, memory utilization is a critical issue that needs to be addressed. This article describes how to best configure x86 system memory performance for large databases.

Virtual Memory Architecture for x86 Platforms

The memory architecture of the x86 and x86-64 chipsets has changed significantly since its inception; however, the default memory page size has not changed. This can result in inefficiency and excessive overhead when large amounts of memory are used for large applications, such as databases.

The x86 architecture is a virtual memory architecture, which allows for more memory to be addressed than is physically available in the hardware. This is accomplished by allowing each process to have its own memory that it can address. The process believes that this memory is available to it for its use. This is known as the process's *virtual memory*. In reality, this memory can be either physical memory that is actually residing on the RAM chips, or it can be stored in a dedicated area on physical disk, which is known as the *swap* or *paging area*.

The process does not know whether the virtual memory is stored in RAM or on disk; the memory is managed by the OS. If more memory is needed than is physically available, the OS will move some memory out to the paging area. This activity is very inefficient and is a common cause of performance problems. Since disk is many times slower than RAM, a process that is "paging" will experience significant performance problems.

Oracle Database and Linux Memory Management

The more memory used in the system, the more resources are required to manage that memory. With the Linux OS, memory management is accomplished via the Linux `kswapd` process and the Page Tables memory structure, which consists of one record for each process that exists in the system. Each record consists of every page of virtual memory used by the process and its physical address (RAM or disk). This process is assisted via the use of the processor's translation lookaside buffer (TLB), a small cache.

When using large amounts of memory for Oracle Database, the OS consumes significant resources to manage the virtual-to-physical translation, which often results in a very large Page Tables structure. Since each Page Tables entry contains the virtual-to-physical translation of all memory pages being used by the process, for a very large System Global Area (SGA), the Page Tables entry can be very large for each process. For example, an Oracle Database process that uses 8 GB of memory will have a Page Tables entry of 8 GB/4 KB or 2,097,152 records, or pages. If there are one hundred Oracle Database sessions/processes, multiply the number of pages by 100. As you can see, that is a huge number of pages to manage.

Again, the Page Tables entries are used by the operating system to manage the memory used by the processes in the system. In Linux, the OS process that does this management is called `kswapd` and can be seen by operating system tools.

The TLB cache will cache the Page Tables entries in order to improve performance. The typical TLB cache holds between 4 and 4,096 entries. With millions or billions of Page Tables entries, this cache is insufficient.

As mentioned earlier, for systems using large SGAs, the Page Tables structure can get very large. The sample Linux system output in Listing 1 shows the Page Tables entries taking up 766 MB of RAM. This can be a significant overhead on the system. I have personally seen Page Tables entries in the gigabytes.

In Linux operating systems, HugePages is a kernel feature that allows the OS to support the large page size capabilities of modern hardware architectures. For Oracle Database, enabling HugePages and using the large page size reduces the operating system maintenance of page states and increases the TLB cache hit ratio by managing more memory with a single page table entry for a large page, rather than with many entries for a smaller page. In Linux, the large page size is 2 MB.

In Oracle Linux 6 or Red Hat Enterprise Linux 6 (RHEL 6), the number of HugePages allocated is available in `/proc/meminfo`, as shown in Listing 1:

```
[root@ptc1 ~]# cat /proc/meminfo
MemTotal:      4045076 kB
MemFree:         14132 kB
Buffers:           656 kB
Cached:        1271560 kB
SwapCached:       6184 kB
Active:        2536748 kB
Inactive:       625616 kB
HighTotal:           0 kB
HighFree:            0 kB
LowTotal:      4045076 kB
LowFree:         14132 kB
SwapTotal:     1052216 kB
SwapFree:            0 kB
Dirty:               0 kB
Writeback:           0 kB
Mapped:        2036576 kB
```

```
Slab:             49712 kB
CommitLimit:    3074752 kB
Committed_AS:   8054664 kB
PageTables:      766680 kB
VmallocTotal:536870911 kB
VmallocUsed:     263168 kB
VmallocChunk:536607347 kB
HugePages_Total:     0
HugePages_Free:      0
Hugepagesize:     2048 kB
```

Listing 1

In Oracle Linux 6, the HugePages allocated are slightly different, as shown in Listing 2:

```
AnonHugePages:          0 kB
HugePages_Total:    1508
HugePages_Free:       60
HugePages_Rsvd:       57
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:       10240 kB
DirectMap2M:    16766976 kB
```

Listing 2

The Oracle Linux 6 HugePages values are as follows:

`AnonHugePages`	The number of Anonymous HugePages. This counter has been removed in Oracle Linux 6.5. Related to Transparent HugePages. （See the "Transparent HugePages and Oracle Databases" section for more information on Transparent HugePages.)
`HugePages_Total`	The number of HugePages. The amount of space is the number of HugePages times 2M.
`HugePages_Free`	The number of HugePages in the pool that are not yet allocated.
`HugePages_Rsvd`	Short for "reserved," and is the number of HugePages for which a commitment to be allocated from the pool has been made, but no allocation has yet been made. Reserved HugePages guarantee that an application will be able to allocate HugePages from the pool of HugePages at the time they are requested, even if the system has been up for a while.
`HugePages_Surp`	Short for "surplus," and is the number of HugePages in the pool above the value in `/proc/sys/vm/nr_hugepages`. The maximum number of surplus HugePages is controlled by `/proc/sys/vm/nr_overcommit_hugepages`. It is not unusual for this to be `0`.
`Hugepagesize`	The size of the HugePage. This is currently 2048 or 2MB.

Solution

By enabling HugePages in Linux, the number of TLB entries can be reduced by increasing the page size. With Linux, the HugePages size is 2 MB. By using larger pages for the Oracle Database SGA, the number of pages to manage is greatly reduced.

In the example shown in Listing 1, the number of virtual-to-physical translations for a single record will be reduced to 4,096 from 2,097,152. This will reduce the size of the Page Tables structure, improve the TLB cache hit ratio, and reduce `kswapd` usage.

Note: The performance improvement when enabling HugePages can be dramatic.

Enabling HugePages in Linux

In Linux, the HugePages capability is configured by setting the Linux initialization parameter `vm.nr_hugepages` to the number of 2 MB pages that you want to make available for the Oracle Database SGA. Setting this parameter can reduce the number of pages by making them larger.

Note: The automatic memory management feature of Oracle Database that was introduced in Oracle Database 11*g* is not compatible with Linux HugePages. The performance improvement that is provided by HugePages outweighs the ease of use provided by automatic memory management.

The details on implementing the HugePages configuration can be found in the following My Oracle Support documents:

My Oracle Support Document ID	Document Name
1557478.1	"ALERT: Disable Transparent HugePages on SLES11, RHEL6, OEL6 and UEK2 Kernels"
361323.1	"HugePages on Linux: What It Is… And What It Is Not"
361468.1	"HugePages on Oracle Linux 64-bit"
749851.1	"HugePages and Oracle Database 11*g* Automatic Memory Management (AMM) on Linux"
1134002.1	"ASMM and LINUX x86-64 HugePages Support"
401749.1	"Shell Script to Calculate Values Recommended Linux HugePages / HugeTLB Configuration"

In addition to configuring `vm.nr_hugepages`, the optional parameter `vm.hugetlb_shm_group` can be set with the OS group that has permissions to use HugePages. This parameter is set to `0` by default, thus allowing all groups permissions to use HugePages. This parameter can be set to an OS group that the Oracle Database process is part of, such as `oinstall`.

Verifying that Large Pages Are Enabled for Your Oracle Database Instance

You can verify that large pages are enabled for your database instance by checking the alert log. At startup of the instance, you should see entries such as the following in your alert log before the parameter listing:

```
****************** Large Pages Information *****************
```

```
 Total Shared Global Region in Large Pages = 28 GB (100%)

 Large Pages used by this instance: 14497 (28 GB)

Large Pages unused system wide = 1015 (2030 MB) (alloc incr 64 MB)

Large Pages configured system wide = 19680 (38 GB)

Large Page size = 2048 KB
```

Transparent HugePages and Oracle Databases

Recently, in RHEL 6, Oracle Linux 6, and SUSE Linux Enterprise Server 11 a new feature, transparent HugePages was introduced. Transparent HugePages is an attempt to make the use of HugePages automatic and dynamic. Unfortunately the use of transparent HugePages in conjunction with the use of traditional HugePages is currently causing problems that can result in performance issues and system reboots. In My Oracle Support note 1557478.1, Oracle recommends not using transparent HugePages in conjunction with Oracle databases.

Note: In Oracle Linux version 6.5 Transparent HugePages have been removed.

Conclusion

By using larger pages, the number of Page Tables entries is reduced and, thus, significant overhead is minimized. The improvement achieved by the use of HugePages is very significant and increases with the amount of memory in the system and the size of the SGA.

See Also

- Performance Tuning website
- Ed Whalen's blog
- Ed Whalen on Twitter

About the Author

Edward Whalen is an Oracle ACE and the Chief Technologist at Performance Tuning Corporation, a consulting company specializing in database performance, administration, migrations, virtualization, and disaster recovery solutions with over 25 years of experience. He has extensive experience in system architectural design for optimal performance. His career has consisted of hardware, OS, database, and virtualization projects for many different companies. Edward has written six books on Oracle products and five books on Microsoft SQL Server, and he just completed the *Oracle Enterprise Manager Cloud Control 12c Deep Dive* book for Oracle Press. He has also worked on numerous benchmarks and performance tuning projects with both Oracle products and Microsoft SQL Server.

Edward has extensive architectural experience consisting of all layers of the cloud/application stack from storage and hardware, to hypervisor and OS, and to the database. This experience provided the foundation for the system architecture work he has done in the past.

Edward started out in college as an experimental physicist working in High Energy Physics on projects at both Fermi lab and the Stanford Linear Accelerator Center. After working briefly on the Superconducting Super Collider, he moved into computer software and then into computer hardware and OS development, which evolved into database performance engineering.