

CheatSheet

#45 zTidBits

z/OS v1.9 and above

z10 Large-Page Support

⚠ Large page (a.k.a. super page) support is in a wide range of computing systems such as our Power5/6 and AIX OS'.

Ⓞ Large Pages in z/OS with addresses > 2 GBs the memory must be obtained in increments of 1MB as Memory Objects distinguished from 4K pages below 2 GBs.

*The IBM System z10 mainframe provides an architectural framework for 1-MB virtual pages. With the new support provided in z/OS v1.9, these large pages can be transparently used by middleware and applications.

- The use of large virtual memory pages for storage allocated at addresses above 2 GB can deliver significant performance gains.

*The availability of 64-bit addressing in both physical and virtual memory has enabled a dramatic increase in memory size available to applications.

- For example, z/OS 64-bit virtual addressing has enabled applications that were once limited to a virtual memory size of 2 GB to access a vast 16 TBs of virtual memory.

- Database management systems can have larger buffer pools than ever before and working set sizes of several gigabytes are now common.

NOTE: The working set size is the amount of memory needed to compute an answer to a process problem. In a multi processing environment managing hundreds to thousands of processes can have a significant amount of memory translation overhead. The significance is that if the Working Set Size is larger than the available memory in a virtual system then the memory manager must refer to the next level in the memory hierarchy (usually auxiliary disk) to perform a swap operation moving some memory contents from real to a paging disk to enable the program to continue working on the problem. If this swapping goes on continuously the program is slowed down significantly. This phenomena is known as *thrashing*.

Working set is defined at three levels: address space, LPAR and Sysplex. *Memory-intensive applications can suffer substantial performance degradations due to an increased number of TLB misses and the increased time penalty of each TLB miss in current machine designs.

NOTE: A Translation lookaside buffer (TLB) is a CPU cache that is used by memory management hardware to improve the speed of virtual address translation.

*Time spent servicing TLB misses and performing DAT by traversing memory DAT-tables has a negative performance impact on application performance.

NOTE: Dynamic Address Translation (DAT): while executing an instruction, a CPU fetches an instruction located at a particular virtual address, fetches data from a specific virtual address or stores data to a particular virtual address, the virtual address must be translated to the corresponding physical address in real memory or central storage.

*Long-running applications with large working sets have an opportunity for increased performance by using large pages to improve the performance of the underlying memory management infrastructure.

- Significant performance gains have been observed running 'specific' workloads on z/OS 1.9 and above on z/10 platforms by using this new support provided for large pages.

- These application performance gains are achieved seamlessly through configuration of the OS to enable it to take advantage of large pages.

However, not all applications are well suited to the use of Large Pages.

*Factors to consider when estimating the potential benefit of using large pages include the application memory usage, the application spatial locality-of-reference, and the workload page-translation overhead. Locality-of-reference is the phenomenon where the same related storage locations are frequently accessed.

*It is important to note that large pages are a special purpose performance improvement feature, and therefore, switching to the use of large pages is not recommended for all workloads.

- Large pages provide performance value to a select set of applications that can generally be characterized as being memory-access intensive and long running.

- These applications can be identified by three criteria: They have large working set sizes that consistently reference data in that working set during program execution, their addresses exhaust the private storage available below the bar (such as IBM WebSphere* application), or they use private storage above the bar (such as IBM DB2 software).

- By the same token, short-lived processes and those with smaller working sets are usually not good candidates for large pages; they may, in fact, experience severe performance degradation through the use of large pages.

- Furthermore, applications that have memory-access patterns with poor spatial locality will suffer from the overhead of using large pages; consideration should be given to whether they will be good candidates for the use of large pages.

*There are two other potential **drawbacks** to the use of large pages.

- **First**, the use of large pages will increase demands to load data in main memory because data for the entire (now larger) page must be loaded into central storage (CSTOR) whether one or all of its 256 4-KB frame pages will be referenced. This potentially can be a memory hog.

- **Second**, security issues require that upon allocation of the 1-MB large page, all 256 4KB frames are cleared to 0s, and this can put an additional workload on the paging subsystem for 4-KB frames.

NOTE: TLB metrics are captured by the z10' Extended Counters providing information about hardware facilities and structures that are specific to a machine family related to the cache hierarchy and processor. See Set-Program-Parameter and CPU-Measurement Facilities, SA23-2260.

Performance Observations

*Measurements taken with z/OS v1.9 with and without large pages (using both a 4way and 16way CPUs Java Application Benchmark) displayed on average 5% improvement to overall throughput.

- Using early prototypes engineers began to understand the potential performance benefits of large pages on z/OS using LSPR (Large Systems Performance Reference)

workloads.

- Improvements evidenced by TLB-related counters showed significant improvements employing large page support with certain workload profiles.

*The illustration in the upper right displays performance data for workloads optimized for large pages using Java multithreaded benchmark large-page 16way measurements.

- "Warehouse" refers to the number of Java threads, where each thread represents additional computational resources to handle additional workload transactions.

z/OS software support for large pages

*A size of 1MB was selected for the large-page size in z/OS because it is a natural fit with existing z/Architecture DAT structures, that is, the segment table rather than the page tables becomes the lowest level of DAT table used for address translation.

NOTE: All DAT implementations use page tables to translate the virtual addresses seen by the application program into physical addresses (also referred to as "real addresses") used by the hardware to process instructions. Each entry in a page table contains the starting virtual address of the page—either the real memory address at which the page is actually stored, or an indicator that the page is currently held in a disk file (if the system uses disk files to let applications use amounts of virtual memory which exceed real memory).

- Of course, the 4-KB page size is still supported as well.

NOTE: Hardware storage protection keys are still based on 4K frames only.

*Several **important characteristics** of the z/OS implementation of large pages are:

- Large pages are for 64-bit private (nor common) memory objects only

- They reside in non-reconfigurable storage

- They are not pageable ☺, therefore not eligible to move to auxiliary storage.

- Large-page memory objects are backed with real storage when allocated.

⚠ Fencing storage from the paging algorithms will inhibit other non-eligible storage starved applications not to achieve their intended SLOs, (Revise WLM Policy as needed),

Alternatively, the hardware does not have to keep track of the change bit for each of the 256 contiguous 4-KB frames making up the large page, thus z/OS sets the changeover write bit and achieves the performance gain in the hardware with this implementation for those that are eligible applications.

Requesting a large-page memory object

*In order for customers to have large-page frames available in a z/OS system, a large frame area must be configured after the initial program load (IPL).

- This is achieved through use of the LFArea keyword in the active IEASYSxx member of SYS1.PARMLIB.

- The system programmer specifies the amount of real storage to be used for large pages using the LFArea keyword with the following syntax:

LFArea=(xx%|xxxxxxM|xxxxxxG|xxxxxxT) keyword on IEASYSxx,

where xx% is the percentage of online storage at IPL to be used for large pages xxxxxxM, xxxxxxG, or xxxxxxT is the amount of on line storage at IPL to be used for large pages, and the default for the large frame area is 0.

NOTE: The maximum that can be specified is 80% of the IPL-time online storage above 4 GB. Once Large Pages is elected it cannot be dynamically changed at the z/OS Console – An IPL is required.

* The use of the large-page interface is available to authorized program facility code, (APF)-authorized libraries; it is possible for non-APF-authorized code to use large pages if they obtain the proper Resource Access Control Facility (RACF)5 authorization.

☺ The primary reason for not paging large pages is that there is no external page table for large pages hence, no place to store information such as where the page was placed on aux storage - other reasons include the fact that there would be tremendous delays in paging 1-MB pages, and the expense of constructing 1-MB contiguous real storage from 256 4-KB frames for a page-in operation was considered prohibitive.

