Cheat Sheet #120 zTidBits z/OS V2.x Overview Module Placement Part 2 of 2

Use #52 & #109 as references for this issue.

Module placement effect on virtual storage

- When a module is loaded into the private area for an address space, the region available for other things is reduced by the amount of storage used for the module.
- Modules loaded from anywhere other than LPA (FLPA, MLPA, dynamic LPA, or PLPA) will be loaded into individual address spaces or into CSA.
- When a module is added to LPA below 16 megabytes, the size of the explicitly-allocated common area below 16 megabytes will be increased by the amount of storage used for the module.
  - When the explicitly-allocated common area does not end on a segment boundary, IPL processing allocates additional CSA down to the next segment boundary.
  - Therefore, which virtual storage boundaries change when modules are added to LPA depends on whether a segment boundary is crossed or not.
- When modules are added to LPA below 16 megabytes, and this does **not** result in the expansion of explicitly-allocated common storage past a segment boundary, less virtual storage will be available for CSA (and SQA overflow) storage.
  - The amounts of CSA and SQA specified in IEASYSxx will still be available, but the system will add less CSA to that specified during IPL.
- When the addition of modules to LPA does not result in a reduction in the size of the private area below 16 megabytes, adding load modules to LPA increases the amount of private area available for address spaces that use those load modules.
  - This is because the system uses the copy of the load module in LPA rather than loading copies into each address space using the load module.

    NOTE: There is no change to the private area storage available to address spaces that do not use those load modules.
- When modules are added to LPA below 16 megabytes, the growth in LPA can cause the common area below 16 megabytes to cross one or more segment boundaries, which will reduce the available private area below 16 megabytes by a corresponding amount; each time the common area crosses a segment boundary, the private area is reduced by the size of one segment.

  NOTE: The segment size in z/OS is one megabyte.
- When the size of the private area is reduced as a result of placing modules in LPA below 16 megabytes, the private area virtual storage available to address spaces that use these modules might or might not be changed.
  - For example, if an address space uses 1.5 megabyte of modules, all of them are placed in LPA below 16 megabytes, and this causes the common area to expand across two segment boundaries, .5 megabytes less private area storage will be available for programs in that address space.

    NOTE:If adding the same 1.5 megabytes of modules causes only one segment boundary to be crossed, **.**5 megabytes more will be available, and adding exactly1 megabytes of modules would cause no change in the amount of private area storage available to programs in that address space, (these examples assume that no other changes are made to other common virtual storage area allocations at the same time.)
- When the size of the private area is reduced as a result of placing modules in LPA below 16 megabytes, less storage will be available to all address spaces that do **not** use those modules.
- A process similar to the process described for LPA is used when ELPA, the other Extended common areas, and the Extended private area are built above 16 megabytes.
  - The only difference is that common storage areas above 16 megabytes are built from 16 megabytes upward, while those below 16 megabytes are built from 16 megabytes downward.
- Modules can also be loaded in CSA, and some subsystems (like IMS) make use of this facility to make programs available to multiple address spaces.

  NOTE: The virtual storage considerations for these modules are similar to those for LPA.

Recommendations for Improving System Performance

The following recommendations should improve system performance.

- They assume that the system's default search order will be used to find modules.
  -You should determine what search order will be used for programs running in each of your applications and modify these recommendations as appropriate when other search orders will be used to find modules.
- Determine how much private area, CSA, and SQA virtual storage are required to run your applications, both above 16 megabytes and below.
- Determine which modules or libraries are important to the applications you care most about.
  - From this list, determine how many are reentrant to see which are able to be placed in LPA.
  - Of the remaining candidates, determine which can be safely placed in LPA, considering security and system integrity.
- All modules placed in LPA are assumed to be authorized and IBM publications identify libraries that can be placed in the LPA list safely, and many list modules you should consider placing in LPA to improve the performance of specific subsystems and applications.
- The system will try to load RMODE(ANY) modules above 16 megabytes whenever possible where RMODE(24) modules will always be loaded below 16 megabytes.
- To the extent possible without compromising required virtual storage, security, or integrity, place libraries containing a high percentage of frequently-used reentrant modules (and containing no modules that are not reentrant) in the LPA list.

  For example, if TSO/E response time is important and virtual storage considerations allow it, add the CMDLIB data set to the LPA list.
  - To the extent possible without compromising available virtual storage, place frequently or moderately-used refreshable modules from other libraries (like the linklist concatenation) in LPA using dynamic LPA or MLPA.
  - Make sure you do not inadvertently duplicate modules, module names, or aliases that already exist in LPA. For example, if TSO/E performance is important, but virtual storage considerations do not allow CMDLIB to be placed in the LPA list, place only the most frequently-used TSO/E modules on your system in dynamic LPA.
  - Use dynamic LPA to do this rather than MLPA whenever possible.
  - Modules that might be used by the system before a SET PROG command can be processed cannot be placed solely in dynamic LPA.
  - If these modules are not required in LPA before a SET PROG command can be processed, the library in which they reside can be placed in the linklist so they are available before a SET PROG can be processed, but enjoy the performance advantages of LPA residency later.

    For example, Language Environment® runtime modules required by z/OS UNIX System Services initialization can be made available by placing the SCEERUN library in the linklist, and performance for applications using Language Environment (including z/OS UNIX System Services) can be improved by also placing selected modules from SCEERUN in dynamic LPA.

Adding optional **storage-class memory** (SCM) on Flash Express cards to your auxiliary storage can increase paging performance and flexibility. Even with SCM usage, page data sets on DASD are required for auxiliary storage. With the exception of VIO data, which must remain on page data sets, all other data types can be paged out to SCM. ASM pages out from main memory to either storage medium based on the response times and on data set characteristics, critical paging requirements, disk availability (during a HyperSwap® failover for example) and available storage space.

Modified link pack area (MLPA/Extended MLPA)

This area may be used to contain re-enterable routines from either APF-Authorized or non-APF-authorized libraries that are to be part of the pageable extension to the link pack area during the 'current' IPL.

NOTE: Any module in the modified link pack area will be treated by the system as though it came from an APF-authorized library.

Ensure that you have properly protected any library named in IEALPAxx to avoid system security and integrity exposures, just as you would protect any APF-authorized library.

The MLPA exists *only* for the duration of an IPL. Therefore, if an MLPA is desired, the modules in the MLPA *must be* specified for each IPL (including quick start and warm start IPLs).

Note: Loading a large number of modules in the MLPA can increase fetch time for modules that are not loaded in the LPA. This could affect system performance.

You can improve the performance of module fetching on your system by allowing **library lookaside (LLA)** to manage your production load libraries. LLA reduces the amount of I/O needed to locate and fetch modules from DASD storage by maintaining (in the LLA address space) copies of the library directories the system uses to locate load modules.

Recommendations for Improving System Performance continued (from lower left)

To load modules in dynamic LPA, list them on an LPA ADD statement in a PROG*xx* member of PARMLIB.

- You can add or remove modules from dynamic LPA without an IPL using SET PROG=xx and SETPROG LPA operator commands.

- By contrast, do not place in LPA infrequently-used modules, those not important to critical applications (such as TSO/E command processors on a system where TSO/E response time is not important), and low-use user programs when this placement would negatively affect critical applications.
  - Virtual storage is a finite resource, and placement of modules in LPA should be prioritized when necessary.
  - Leaving low-use modules from the linklist (such as those in CMDLIB on systems where TSO/E performance is not critical) and low-use application modules outside LPA so they are loaded into user subpools will affect the performance of address spaces that use them and cause them to be swapped in and out with those address spaces.

    NOTE: This placement usually has little or no effect on other address spaces that do not use these modules.
- Configure as much storage as possible as central storage.
- If other measures (like WLM policy changes, managing the content of LPA, and balancing central and expanded storage allocations) fail to control storage saturation, and paging and swapping begin to affect your critical workloads, the most effective way to fix the problem is to add storage to the system.
  - Sometimes, this is as simple as changing the storage allocated to different LPARs on the same processor.
  - You should consider other options only when you cannot add storage to the system.

    NOTE: For additional paging flexibility and efficiency, you can add optional storage-class memory (SCM) on Flash Express® solid-state drives (SSD) as a second type of auxiliary storage. DASD auxiliary storage is required.
- If you experience significant PLPA paging, you can use the fixed LPA to reduce page-fault overhead and increase performance at the expense of central storage.
  - You can assure that specific modules are kept in central storage by adding them to the fixed LPA by listing them in IEAFIXxx.
  - This trade-off can be desirable with a system that tends to be CPU-bound, where it can be best to divert some of the central storage from possible use by additional address spaces, and use it for additional LPA modules.
- High-usage PLPA modules probably need not be listed in IEAFIX*xx* because they tend to be referenced frequently enough to remain in central storage.
  - A large FLPA makes less central storage available for pageable programs.
  - Accordingly, fewer address spaces might be in central storage than would otherwise be the case.
  - No loss in throughput should occur, however, if CPU use remains reasonably high.

    NOTE: A large FLPA can increase other demand paging and swapping activity and that this will impede the system's normal self-tuning actions because keeping these modules in storage might prevent other, more frequently-used modules, from being in storage as workloads shift over the course of time. Also, like module packing lists, fixed LPA lists need to be maintained when installing new releases of software, installing significant amounts of service, or when your workloads change. If you can prevent LPA paging by adding central storage, the system will be simpler to manage.
- When there is significant page-in activity for PLPA pages, and you cannot take other actions to reduce it economically, you can minimize page faults and disk arm movement by specifying module packing through the IEAPAKxx member of parmlib.
  - Module ***packing*** reduces page faults by placing in the same virtual page those small modules (less than 4K bytes) that refer to each other frequently.
  - Module groups that frequently refer to each other but that exceed 4K bytes in combined size can be placed in adjacent (4K) auxiliary storage slots to reduce seek time

    > Thus, use of IEAPAKxx should improve performance compared with the simple loading of the PLPA from the LPALST concatenation.
- You *must* maintain module packing lists whenever you install new levels of software or significant service, or when your workloads change.
  - If you can increase the amount of central storage enough to control PLPA paging rather than using a module packing list, the system will be simpler to manage.
- If the first PLPA page data set specified during IPL is large enough, all PLPA pages are written to the same data set.
  - If the first page data set is not large enough to contain all PLPA pages (for example, when allocated as a one-cylinder data set as recommended below), the remaining pages are written to the common page data set (the second one specified during IPL).

⇨**For best performance, all PLPA pages would be written to a single page data set on a single DASD volume.**

However, a reasonable alternative is to define the PLPA page data set as a single cylinder and define enough storage for the common page data set to contain both the PLPA and common pages. When defined this way, the PLPA and common page data sets should be contiguous, with the small PLPA data set followed immediately by the large common data set on the volume. You should consider allocating these data sets this way unless you experience significant PLPA paging, because it reduces the number of page data sets for which space must be managed and simplifies support.

- If you have significant swapping or paging activity that affects critical applications, and you cannot add central storage or **storage-class memory (SCM)** to manage it, you can tune the paging subsystem.
  - When most paging subsystem activity is for swapping, a large number of page data sets can outperform a small number of page data sets, even on high-speed or cached devices.
  - If you have substantial swapping, consider using eight or more page data sets on different low-use volumes on low-use control units and channel paths. However, these should generally be considered stop-gap solutions.
  - If the storage demand continues to grow, tuning the paging subsystem will usually delay the inevitable for only a short time.
  - In the long run, adding central storage is always a better solution.

  NOTE: Some cached devices also do not support cached paging.

March 2016