Rule CIC112: The number of GETMAINs is high

Finding: CPExpert has detected that the number of GETMAINs issued by tasks is higher than should normally occur.

Impact: This finding normally has a LOW IMPACT on the performance of the CICS region, although the finding could have a MEDIUM IMPACT or HIGH IMPACT on the performance of individual CICS tasks. Additionally, excessive GETMAINs generate unnecessary overhead[1].

Discussion: The CICS statistics report the number of times that CICS acquired storage for a task (CICS issued a GETMAIN). The number of GETMAINs is related to the design and use of the tasks and applications. Consequently, there are no CICS tuning controls directly related to the number of GETMAINs.

GETMAINs impose system overhead for the system code necessary to execute the GETMAIN. It is desirable to minimize the GETMAINs per task to minimize this overhead, although tasks obviously must issue GETMAINs to acquire any necessary storage. As a "rule of thumb," there should be fewer than an average of 25 GETMAINs per task[2]. More than this value generally indicates that the task may be improperly designed or that it is experiencing performance problems of some type.

It is not uncommon to see hundreds or even thousands of GETMAINs executed by poorly designed (or poorly implemented) tasks!

CPExpert divides the number of storage acquisitions by the number of tasks. The result from this division is an average; some tasks would have issued relatively few GETMAINs, while other tasks would have issued many more than the average. CPExpert produces Rule CIC112 if the result is greater than the GETMAIN guidance variable in USOURCE(CICGUIDE). The default value for the GETMAIN guidance variable is 25, so CPExpert will produce Rule CIC112 when the average task in a CICS region issues more than 25 GETMAINs.

[1]The "Sample Performance Data" of IBM *CICS Performance Guides* show that each GETMAIN and associated FREEMAIN require almost 2000 CPU instructions.

[2]The IBM *CICS Performance Guides* specify that storage control problems would be indicated by "a very high number of storage control requests per task (perhaps 50 or 100)" but a reasonable limit on the number of GETMAINs would typically be less than 25 per task.

CPExpert reports the average number of GETMAINs resulting from the above computation. When evaluating this value, keep in mind that the value represents an average over all tasks. Some tasks will issue very few GETMAINs, which means that some tasks will issue many more than the average.

Suggestion: CPExpert suggests that you investigate the abnormal number of GETMAINs. If you conclude that the number of GETMAINs per task is excessive, consider the following alternatives:

- Discuss the tasks with system developers to determine whether the number of GETMAINs can be reduced, or whether the number of GETMAIN requests consistent with the number and types of tasks.

  Once a problem of this type has been brought to the attention of system developers, they often can quickly resolve the situation. As examples of problems that system developers might identify:

  - A GETMAIN could be enclosed within a loop, so the GETMAIN is executed with each iteration of the loop. Simply moving the GETMAIN outside the loop could dramatically reduce the number of GETMAINs.

  - A task might be issuing GETMAIN requests repetitively, each for a reasonable amount of storage, but collectively for a very large amount of storage. Performance can be improved by obtaining all required storage in one GETMAIN rather than several smaller GETMAINs.

  - The IOAREALEN (value1) or TIOAL might be smaller than most initial terminal inputs. In this case, excessive GETMAIN requests can occur, resulting in additional processor requirements (unless IOAREALEN(value1) or TIOAL is zero). In general, a value of zero is best because it causes the optimum use of storage and eliminates the second GETMAIN request.

  - In an MRO environment, IOAREALEN might be smaller than most messages transmitted on the MRO link. In this case, excessive FREEMAIN and GETMAIN requests can occur, resulting in additional processor requirements.

  - Language Environment runtime options can have a major impact on both storage usage and CPU costs of Java application programs running under CICS. The key Language Environment runtime options for Java application programs are STACK, HEAP and ANYHEAP. If the initial size for any of these options is too large, excessive storage will be allocated, which may result in a short-on-storage condition in the CICS region. If an initial value is too small, Language Environment will issue a GETMAIN to allocate additional storage, which increases the CPU cost. Additional CPU cost can also be incurred due to extra GETMAINs and FREEMAINs if the FREE parameter is specified for any option where the initial size is too small.

- The system pathlength increases when a CICS application invoked by Language Environment issues an EXEC CICS LINK request. Repeated EXEC CICS LINK calls to the same program invoked by Language Environment result in *multiple* GETMAIN/FREEMAIN requests for run-time unit work areas (RUWAs). The allocation of storage depends on the setting of the CICS system initialization parameter, RUWAPOOL.

  - RUWAPOOL=NO causes CICS to issue a GETMAIN for storage and pass it to Language Environment to use for control blocks and storage areas. The storage is freed (using FREEMAIN) when the program terminates.

  - RUWAPOOL(YES) results in the creation of a run_unit work area pool during task initialization. This pool is used to allocate RUWAs required by LE_conforming programs. This reduces the number of GETMAINS and FREEMAINS in tasks that perform many EXEC CICS LINKS to LE_conforming programs.

    CICS keeps a history of the total storage for RUWAs that is requested to run the transaction. When the transaction is run again, CICS issues a single GETMAIN for the total storage (and a single FREEMAIN at task end). If the transaction follows the same path, CICS allocates the storage from the RUWAPOOL, and no further GETMAIN has to be issued. If more storage is required for RUWAs because of different or extra CICS links, CICS issues a GETMAIN and updates the history, so that next time the single GETMAIN (and FREEMAIN) is for the larger amount. For transactions that issue a large number of CICS LINK commands, IBM reports that there can be significant performance improvement.

- You can use the AUTODST system initialization parameters to minimize the number of GETMAIN and FREEMAIN requests that CICS performs on behalf of Language Environment. The AUTODST parameter specifies whether CICS is to activate automatic dynamic storage tuning for application programs.

- AUTODST=NO specifies that automatic dynamic storage tuning is not required and CICS does not request this support from Language Environment.

- AUTODST=YES specifies that automatic dynamic storage tuning is required. This feature is activated during CICS startup when Language Environment is being initialized. CICS indicates to Language Environment that it is able to support dynamic storage tuning, and if Language Environment responds by indicating that it also supports the facility, CICS and Language Environment are synchronized to provide the required support.

  When this function is active, Language Environment monitors execution of each main program, and notes if any additional storage had to be allocated for the program while it was active. If any additional storage had to be allocated, Language Environment retains this information. The next time the program is executed, Language Environment increases the initial storage allocation to include the extra storage that was allocated. This process helps to minimize the number of GETMAINs and FREEMAINs that CICS has to perform.

  A potential problem is created because once Language Environment has increased the initial storage allocation for a program, the initial storage is never decreased. If a program execution requires an unusually large amount of storage, perhaps because the user has activated a seldom-used function of the program, this amount of storage is allocated for all subsequent executions of the program. In rare cases, automatic storage tuning leads to an excessive allocation of storage for some programs.

- When the Java program object is invoked in a CICS system, a storage report can be written[3] to the CICS CESE transient data destination (usually directed to the data set defined by the CEEMSG DD statement). The report shows the number of system-level get storage calls, such as EXEC CICS GETMAIN, that were required while the application was running. To improve performance, use the storage report numbers as an aid in setting the initial and increment size for STACK, HEAP and ANYHEAP to values that will reduce the number of times that the language environment storage manager makes requests to acquire storage.

[3]To get a report on the storage used by your Java program object, specify the following runtime option on the hpj command _lerunopts="RPTSTG(ON)". RPTSTG **should only be used in a test environment** because of the overheads incurred in writing the storage report each time the Java program object is executed.

  - When transaction isolation is used, it is necessary to activate pages of storage to the task's allocated subspace. CICS must activate user storage to a subspace every time the user task invokes the GETMAIN command to get a new page of user key task lifetime storage. Activating storage to a subspace requires overhead, so the activity should be kept to a minimum. When storage below the line is activated, performance can be improved by obtaining all the storage in one GETMAIN requests rather than several smaller GETMAIN requests.

- If necessary, analyze the task issuing an unacceptable number of GETMAINs in some detail. This can be done by performing "single-transaction measurement" as outlined in the CICS Performance Guide.

- You can change the **GETMAIN** guidance variable in USOURCE(CICGUIDE) if you feel that Rule CIC112 is produced prematurely. Section 3 describes how to change the GETMAIN guidance variable.

Reference: *CICS/MVS Version 2.1.2 Performance Guide*: pages 108 and 422.

CICS/ESA Version 3.1.1 Performance Guide: pages 117 and 192.

CICS/ESA Version 3.2.1 Performance Guide: pages 96 and 331.

CICS/ESA Version 3.3.1 Performance Guide: pages 106 and 351.

CICS/ESA Version 4.1.1 Performance Guide: Section 3.3.4.

CICS/TS Release 1.1 Performance Guide: Section 3.3.4.

CICS/TS Release 1.2 Performance Guide: Section 3.3.4.

CICS/TS Release 1.3 Performance Guide: Section 3.3.4.

CICS/TS for z/OS Release 2.1 *Performance Guide*: Section 3.1.2, Section 4.3.1 (Setting the size of the terminal input/output area), *Section* 4.6.3.2 (Language Environment runtime options)

CICS/TS for z/OS Release 2.2 *Performance Guide*: Section 3.1.2, Section 4.3.1 (Setting the size of the terminal input/output area), *Section* 4.6.3.2 (Language Environment runtime options)

CICS/TS for z/OS Release 2.3 *Performance Guide*: Section 3.1.2, Section 4.3.1 (Setting the size of the terminal input/output area), *Section* 4.6.3.2 (Language Environment runtime options)

CICS/TS for z/OS Release 3.1 *Performance Guide*: Section 4.3.1 (Setting the size of the terminal input/output area); Section 4.12.3,1 (Minimizing GETMAIN and FREEMAIN activity)

CICS/TS for z/OS Release 3.2 *Performance Guide*:
Chapter 12. Networking and VTAM: improving performance
(Setting the size of the terminal input/output area)
Chapter 21. Programming: performance considerations
Tuning with Language Environment (Minimizing GETMAIN and FREEMAIN activity)

CICS/TS for z/OS Release 4.1 *Performance Guide*:
Chapter 12. Networking and VTAM: improving performance
(Setting the size of the terminal input/output area)
Chapter 21. Programming: performance considerations
Tuning with Language Environment (Minimizing GETMAIN and FREEMAIN activity)
Sample performance data for API calls

CICS/TS for z/OS Release 4.2 *Performance Guide*:
Chapter 7. CICS storage protection facilities: Performance and tuning (Transaction isolation and applications)
Chapter 8. Tuning with Language Environment
Chapter 11. Networking and the z/OS Communications Server: performance and tuning (Setting the size of the terminal input and output area
Chapter 13. CICS VSAM and file control: Performance and tuning (Defining VSAM string settings for LSR

CICS/TS for z/OS Release 5.1 *Performance Guide*:
Chapter 9. CICS storage protection facilities: Performance and tuning
(Transaction isolation and applications)
Chapter 10. Tuning with Language Environment
(Minimizing GETMAIN and FREEMAIN activity)
Chapter 13. Networking and the z/OS Communications Server: performance and tuning (Setting the size of the terminal input and output area
Chapter 15. CICS VSAM and file control: Performance and tuning (Defining VSAM string settings for LSR)

Also, please review the “Loops” section of the *CICS Problem Determination Guide* for your version of CICS.