10 Ways to Improve RACF Performance

By Robert S. Hansel

Every minute of a typical workday, RACF is bombarded with hundreds, if not thousands, of requests to validate user IDs and passwords and to verify users are authorized to access resources. Almost every user action prompts a RACF check, whether to execute a program, log on to a particular CICS region, open a data set, enter an IMS transaction, access a DB2 table, initiate an operator command, perform a storage administration function, or run a Time Sharing Option (TSO) logon procedure. All the while, RACF is updating last used dates on user IDs during logons, responding to RACF commands, and initiating System Management Facilities (SMF) logging. Many of these tasks require RACF to perform I/O to its primary database and sometimes to the backup, too. Any requests that arrive while RACF is busy are placed in a queue to wait their turn. Many requestors can't proceed until they've received RACF's response. Work throughout the system can bog down while RACF works its way through any backlogs.

Proper tuning is essential to ensure that RACF doesn't become a performance bottleneck. This article offers the following 10 suggestions for ensuring RACF operates at peak efficiency:

1 Implement the Global Access Table (GAT)

The GAT contains a list of resources everyone is given immediate access to without further checking. The GAT check occurs early in RACF's access authorization decision process before there's any I/O to find and fetch a profile. A well-designed GAT can greatly improve performance.

The GAT is constructed from a set of profiles defined to the >

GLOBAL class. A GLOBAL profile is a resource class name (e.g., DATASET). Within each profile is a list of entries related to the associated resource class. These entries look similar to normal resource profiles to include the use of generic masking characters. Associated with each entry is an access level (e.g., READ) that's the highest level the GAT grants to everyone. Once a GLOBAL profile has been created for a specific resource class, it must be activated with the command SETROPTS GLOBAL(classname). Figure 1 shows the commands for building and activating a GAT profile and entry. Figure 2 provides sample entries.

&RACUID is a variable available for use with the GAT and is substituted with the user's logon user ID. This variable is especially useful in coding entries for resources where the user would be considered the owner of the resource and automatically granted access. An example is a TSO user data set; hence, the entry &RACUID.*.** or equally acceptable &RACUID.**. &RACGPID, another GAT variable, is substituted with the user's current connect group (i.e., the group the user logged on with).

When working with the GAT, remember that:

- The list of entries should be as short as possible to minimize time spent checking the GAT.
- If no GAT entries are to be defined for a specific resource class, deactivate GAT checking for the class to avoid wasting CPU cycles. Enter the command SETROPTS NOGLOBAL(classname).
- After changing entries in a GLOBAL profile, activate the changes with the command SETROPTS GLOBAL (classname) REFRESH.
- Access granted via the GAT isn't logged, so if logging is essential, don't add an entry.
- Be careful that GAT entries don't compromise security. An entry in the GAT supersedes any corresponding RACF resource profiles and may grant access the profiles would otherwise deny.

2 Maximize in-storage resident data blocks

The RACF database is organized into blocks, each 4K in size. A block may contain either a portion of the database index or one or more profiles or profile segments. RACF retrieves data from the database in blocks and not by individual profiles.

RACF maintains buffers in memory, known as resident data blocks, to retain copies of most recently used database blocks for reuse. You can specify the desired number of resident blocks in the database name table (ICHRDSNT). You should always specify the maximum 255. Figure 3 shows sample code to build the ICHRDSNT.

3 Use RACLIST classes whenever feasible

RACLIST is a SETROPTS option for loading general resource profiles into memory for rapid reference; it causes all profiles for the class to be retrieved from the database and stored in a z/OS data space. All subsequent access checks reference this in-storage copy of the profiles. RACLISTing is especially valuable for classes used during logon processing, particularly those that affect CICS logons.

RACLISTing can occur in one of two ways. A system application (e.g., CICS) can RACLIST a class itself, using the RACROUTE macro. Or, a system-SPECIAL administrator can issue a SETROPTS RACLIST(classname) command. For an administrator to RACLIST a class with the SETROPTS command, the class must be defined with RACLIST ALLOWED.

If you change any profiles, the RACLISTed profiles must be refreshed for the change to take effect. Enter the command SETROPTS RACLIST(classname) REFRESH. During the refresh, all profiles are retrieved from the database and loaded into a new data space. Once the new data space is created and placed in service, the old one is discarded. This avoids disruption during the refresh process.

During RACLISTing, the contents of grouping class profiles merge with the member class profiles to form a single composite list. If there are many profiles or grouping profiles with many resources, this can be a lengthy, resource-intensive process. If the class being refreshed is linked to others by a common POSIT number, all will be refreshed. Institute administrative practices that minimize the need for changes and refreshes. For example, use groups in access lists instead of individual user IDs. Avoid refreshes during the workday, if possible.

Some third-party software products give you the option of using a general resource class for protecting resources or the DATASET class. As a rule, opt for

```
RDEFINE GLOBAL DATASET
RALTER GLOBAL DATASET ADDMEM('SYS1.HELP'/READ)
SETROPTS GLOBAL(DATASET)
```

Figure 1: Building a Global Access Table Profile and Entry

```
GLOBAL      ADDMEM             PERMITTED
PROFILE     ENTRY              ACCESS
DATASET     &RACUID.*.**       ALTER
DATASET     SYS1.BROADCAST     READ
DATASET     SYS1.HELP          READ
DATASET     SYS1.MACLIB        READ
DATASET     catalog.master     READ
DATASET     catalog.user       UPDATE
DATASET     ispf.library       READ
DATASET     sdsf.library       READ
JESSPOOL    *.&RACUID.**       ALTER
JESSPOOL    *.*.$JESNEWS.**    READ
JESJOBS     CANCEL.*.&RACUID.* ALTER
TSOAUTH     JCL                READ
TSOAUTH     RECOVER            READ
```

Figure 2: Sample Global Access Table Entries

```
AL1(1)               Number of databases
CL44'RACF.PRIMARY'   Primary DB name
CL44'RACF.BACKUP'    Backup DB name
AL1(255)             # of Resident Data Blocks
XL1'80'              Statistics & Sysplex Flags
```

Figure 3: ICHRDSNT Showing Resident Blocks Specification

the resource class to take advantage of RACLISTing.

4 RACGLIST the RACLISTed classes

Aside from the initial RACLISTing, RACF performs the RACLISTing process previously described for all classes at Initial Program Load (IPL) and for specific classes whenever an administrator issues a SETROPTS RACLIST(classname) REFRESH. Each z/OS system image must perform this RACLIST process for itself.

You can avoid some of this processing by using RACGLIST. With RACGLIST, once the RACLIST processing is complete, RACF saves a copy of the post-processed profiles in the database. During IPL, RACF retrieves and loads these profiles into the data space without repeating the merging process.

RACGLIST is especially valuable for resource classes used across multiple z/OS system images where RACF Sysplex data communications is active. In this configuration, when a RACLIST REFRESH is issued on one system, RACF performs the RACLIST process, saves the post-processed profiles in the database, and informs the other systems to retrieve and load the RACGLIST copy. This is beneficial because it saves all the other systems the CPU cycles needed to RACLIST the profiles for themselves and ensures all the system images are referencing an identical set of in-storage profiles. Without RACGLIST, a profile change could be introduced when the various systems are performing their own separate RACLIST REFRESHes such that some systems incorporate the change when others don't. These inconsistencies can cause unexpected access failures, especially with CICS inter-region communications, and can be difficult to troubleshoot.

RACGLIST is a resource class whose profiles are the names of RACLISTed classes. To activate RACGLIST for a class, define the RACLISTed member class as a profile in the RACGLIST class as follows: RDEFINE RACGLIST classname. Nothing more is required. RACF will automatically update the profiles with a RACLIST REFRESH. Use of the RACGLIST class profiles does require additional space in the RACF database to store the post-processed profiles, so check available free space first.

5 Increase Enqueue Residency—(ERV)

When a RACF command executes, the TSO user or batch job initiating the command holds an exclusive enqueue on the database; no other process may access the database until the command completes. Such users are often assigned a low processing priority and may be swapped out in the midst of executing a command. This is especially likely when executing large batches of CONNECT and REMOVE commands, as these usually entail updates to both a USERID and a GROUP profile. Unfortunately, the user continues to hold the enqueue while swapped out. All other RACF requests are delayed until the user is allowed to proceed and finish the task. To avoid this, increase the value of the ERV parameter in PARMLIB member IEAOPTxx. This grants more CPU service units to any process with an enqueue on a system resource (e.g., RACF) to let it complete work before being swapped out. The default value is 500, and the recommended value is 40,000 to 50,000. This change will benefit all processes with enqueued system resources.

6 Use Virtual Lookaside Facility (VLF) caching

VLF can cache several types of RACF information for instant reuse, including user logon information, group tree structure, and z/OS Unix identity information. Figure 4 shows the entries to be made in the VLF configuration parameters to implement caching.

During logon, RACF builds an Accessor Control Environment Element (ACEE) in storage for each user. The ACEE is used in subsequent access authorization checks. It contains information about the user that can be obtained only by retrieving the user's profile from the database. This information includes the user's ID, name, and attributes, as well as installation data and connect groups. A user may perform many logons during a typical workday. A TSO user may submit multiple batch jobs; each results in a separate logon. Characteristics of these various logons are often the same and the ACEE RACF would build for each of them is identical. ACEEs can be cached in VLF and reused to avoid repeated retrieval of the user profile and ACEE re-creation.

Whenever a user attempts to exercise a group-level authority such as changing a profile using Group-SPECIAL, RACF creates a map of the pertinent portion of the group tree to determine the user's eligibility to perform the action. This involves I/O to find the owner of each profile in the ownership chain to determine if the target profile or resource falls in the user's scope of groups. RACF creates this map for each TSO user or batch job individually, storing the results in the user's address space, where it will be discarded at logoff or job completion. If many users exercise group-level authority, this process can become a burden to RACF. To reduce the work involved in checking successive group authority actions, the group tree mappings can be cached in VLF for reuse. The cached information is available to all users and isn't discarded when users log off.

When performing certain z/OS Unix functions, Unix uids and gids need to be mapped to their corresponding user IDs and groups, respectively. This mapping process requires interrogation of either every user's or group's OMVS segment, UNIXMAP profiles, or the Application Identity Mapping (AIM) index structure if the database was recently restructured. Looking at every OMVS segment certainly hampers performance. Activating the UNIXMAP class avoids this by satisfying the mapping request with a single profile check. (RACF automatically creates and

```
PARMLIB(COFVLFxx)
    CLASS NAME(IRRACEE)     /* ACEE */
     EMAJ(ACEE)
    CLASS NAME(IRRGTS)      /* GROUP TREE */
     EMAJ(GTS)
    CLASS NAME(IRRGMAP)     /* UNIX GID */
     EMAJ(GMAP)
    CLASS NAME(IRRUMAP)     /* UNIX UID */
     EMAJ(UMAP)
    CLASS NAME(IRRSMAP)     /* UNIX USP */
     EMAJ(SMAP)
```

Figure 4: VLF Entries to Cache RACF Information

updates these profiles in conjunction with changes to OMVS segments.) Better still is implementing AIM. In any case, this process can be further expedited by caching the mapping results in VLF for reuse. You also can improve the performance of applications using thread level security by caching user security packets in VLF.

7 Make efficient use of groups

When a user requests access to a resource, RACF will check the access list of the associated profile to determine if the user has been permitted the appropriate authority. During this process, RACF may perform up to three inspections of the access list.

RACF first looks through the entire access list for any entry matching the user's ID. If it finds a match, RACF decides whether to grant the access based on the level of access permitted. If RACF doesn't find an entry matching the user ID, it then takes each of the user's groups and looks through the access list for any entry matching the group name. RACF continues this process until it has checked every one of the user's groups against every entry in the access list. If any of the groups matches an entry, RACF decides whether to grant the access based on the highest access permitted. If none of the user's groups are in the access list, RACF performs a final check to see if ID * (any RACF defined user) is in the access list.

To improve performance, reduce both the number of entries in each access list and the number of groups to which each user is connected. As a measure of performance, multiply the number of groups to which a user is connected by the number of entries in the access list to calculate the number of comparisons to be made. The smaller the number, the faster the process. Good role-based access design and group consolidation can benefit performance and improve administration.

Avoid adding individual user IDs to access lists. Organize users into groups and grant the groups access to resources instead. Exceptions to this rule are started tasks, batch IDs, File Transfer Protocol (FTP) IDs, and other process-type IDs. These usually have unique access needs that don't lend themselves to grouping. Placing their IDs in the access list avoids needless group checks and boosts performance, since the user ID check occurs first.

8 Replace OPERATIONS with storage administration authorities

It isn't uncommon to find OPERATIONS authority being used for bulk storage administration tasks such as backups or DASD reorganizations. Such tasks may result in hundreds of authorization checks, one for each data set involved. Each individual access is likely to be logged to SMF. It may be possible to avoid all these checks and associated logging through the effective use of storage administration-related authorities. These authorities include access to DASDVOL profiles, FACILITY class STGADMIN profiles, and ALTER access to catalogs. A single authorization check to confirm the user has the proper storage administration authority may be enough to bypass all the individual data set access checks. Effective implementation of these other authorities may eliminate virtually all use of the powerful OPERATIONS authority, which will likely please the auditors.

9 Implement Sysplex coupling facility caching

RACF databases shared in a Sysplex can benefvit from Sysplex data sharing, which uses a coupling facility as a large store-through cache. Index and data blocks are stored in the cache, then fetched from there instead of the database. You can use a coupling facility with caching to improve performance even for a stand-alone, non-Sysplexed system. Statements in the coupling facility policy govern activation and extent of caching. As a rule, ensure the cache size is large enough to hold all non-RACLISTed profiles.

10 Cease gathering unnecessary statistics

SETROPTS STATISTICS(classname) causes the access counter in discrete, non-RACLISTed profiles to be incremented for each access. This causes needless I/O. These rarely examined statistics are of little value. To improve performance, eliminate the recording of statistics for all classes by entering the command SETROPTS NOSTATISTICS(*).

Conclusion

RACF is the focal point of high volumes of critical security requests from every system process; its performance affects the entire system. It offers a wealth of features and opportunities for optimizing the authorization process, minimizing database I/O, and making I/O more efficient. If properly tuned, RACF should go unnoticed. Otherwise, users may start clamoring for a reduction in security controls to lessen RACF's impact on their work. Implementing the tuning options and practices discussed here will help meet service-level commitments while retaining good security. Z

About the Author

Robert S. Hansel is president of RSH Consulting, Inc., a firm specializing in RACF. He has worked with RACF for nearly 20 years in the roles of manager, administrator, auditor, instructor, and consultant. He supports several of the RACF Users Groups throughout the U.S.
Email: R.Hansel@rshconsulting.com
Website: www.rshconsulting.com