A Practical Guide to Greenplum Database Resource Group Management

By Wen Wang, VMware Greenplum Product Manager, Beijing China

Version 1, 8/9/2022

Introduction to Resource Group Management

In terms of resource management, Greenplum has undergone an upgrade from resource queue to resource group. Different database users belong to different groups, and the new resource group management can assign resources to different groups. Resource management groups can individually set their specific CPU, memory and concurrency limits for each group, so as to control the allocation and use of resources at different levels.

Comparison of Resource Group and Resource Queue

The main comparative differences between resource groups and resource queues are shown in the following table.

Metric	**Resource Queues**	**Resource Groups**
Concurrency	Based on query level management	Transaction-level management
CPU	Specify the priority of the query	Linux - based Control Groups. Specify the percentage of CPU resources or the specific number of CPUSET cores .
Memory	Managed at the queue and operator level; users can over-subscribe	Managed at the transaction level with enhanced allocation and tracking; users cannot over-subscribe
Memory Isolation	none	Memory is isolated between resource groups and between transactions within the same resource group
Users	Restrictions only apply to non-admin users	Restrictions apply to SUPERUSER and non-admin users
Query Failure	If there is not enough memory, the query may fail immediately	When there is not enough memory in the slot, try to get it from the shared memory in the resource group; if there is not enough memory in the shared memory in the resource group, try to get it from the global shared memory; if there is still not enough memory in the global shared memory, the Query failed.
Limit Bypass	No restrictions on the SUPERUSER role and certain operators and functions	SET, RESET and SHOW commands are not restricted
External Components	none	CPU and memory resources of PL / Container

From this, we can see that the resource group manages resources in a more fine-grained manner. Concurrency control is based on the transaction level, which better guarantees transaction consistency. And realizes the flexible use of multiple levels of resources such as intra-group memory sharing and global memory sharing.

Resource Group Resource Group Management

Greenplum Database supports two types of resource groups: groups that manage role resources, and groups that manage resources of external components such as PL/Containers. In the group that manages role resources , the Linux-based CGroup is used for CPU resource management, and the memory is counted and tracked through vmtracker. Users can also use resource groups to manage external components such as PL/Container, which use Linux CGroup to manage the total CPU and total memory resources of the component .

Main configuration introduction

The parameters and limits of the resource group are as follows:

type of restriction	describe
MEMORY_AUDITOR	The memory auditor of the resource group, the default value is vmtracker. Vmtracker is used to audit Greenplum internal memory usage; cgroup It is used to specify the memory usage of external components, such as PL/ Container, and CONCURRENCY must be set to 0 at this time.
CONCURRENCY	The maximum number of concurrent transactions allowed in the resource group, including active and idle transactions . Note : When the transaction is not over, even if there is no statement executing (idle state) , the slot will be occupied .
CPU_RATE_LIMIT	The percentage of CPU resources available to this resource group .
CPUSET	Specific CPU logical cores or logical threads in hyperthreading reserved for this resource group .
MEMORY_LIMIT	The percentage of memory resources available to this resource group .

MEMORY_SHARED_QUOTA	The percentage of memory resources shared between transactions in this resource group
MEMORY_SPILL_RATIO	Memory usage threshold for memory-intensive transactions. When a transaction reaches this threshold, it will be spilled to disk .

Note: Resource limits are not enforced for SET, RESET and SHOW commands.

Main command introduction

The main related creation commands are:

1. Start the resource group :
    a. Set the gp_resource_manager server configuration parameter to the value "group ":
        i. gpconfig -s gp_resource_manager
        ii. gpconfig -c gp_resource_manager -v "group"
    b. Restart Greenplum Database
    c. After enabling the resource group option, 2 resource groups will be created by default
        i. admin_group and default_group:
        ii. E.g:

type of restriction	admin_group	default_group
CONCURRENCY	10	20
CPU_RATE_LIMIT	10	30
CPUSET	-1	-1
MEMORY_LIMIT	10	30
MEMORY_SHARED_QUOTA	50	50
MEMORY_SPILL_RATIO	20	20
MEMORY_AUDITOR	vmtracker	vmtracker

2. Create a resource group.
    a. =# create resource group rg_new with (concurrency=5, cpu_rate_limit =20, memory_limit =30, memory_shared_quota =20, memory_spill_ratio=30);
3. Modify the configuration of the resource group.
    a. =# alter resource group rg _group set memory_spill_ratio 50;
    b. =# alter resource group admin_group set memory_limit 70;
4. SQL to bind a resource group to a user :
    a. =# CREATE ROLE r1;
    b. =# ALTER ROLE r1 RESOURCE GROUP rg_new ;
5. Delete a resource group.
    a. =# drop resource group rg_new;

Introduction to Transaction Concurrency Limits

The transaction concurrency limit refers to the maximum number of concurrent transactions allowed in the resource group. The resource group of each role is logically divided into a fixed number of slots, that is, the concurrency limit. Greenplum Database allocates memory resources to these slots at the same fixed percentage .

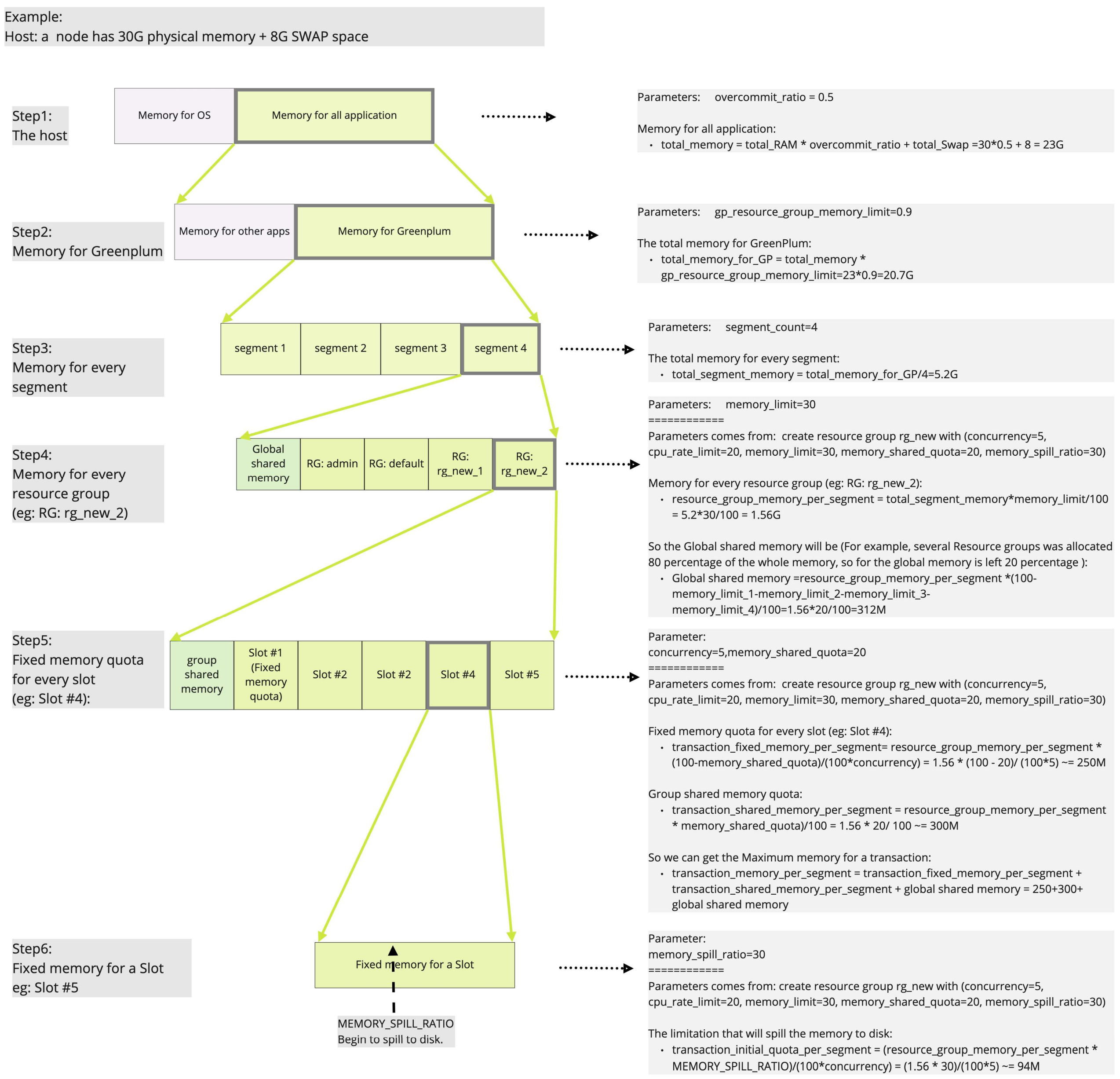
1. Greenplum Database queues committed transactions after the resource group reaches the concurrency limit. When a running transaction completes, if sufficient memory resources exist, Greenplum Database removes the oldest transaction from the waiting queue to run the queue , that is, obeys the first-in, first-out principle (FIFO).
2. If you want to bypass the resource group concurrency limit, you can configure the parameter gp_resource_group_bypass through the server.
    a. Enables or disables concurrent transaction limits on the Greenplum Database resource group. The default value is false, which enforces a limit on the resource group transaction concurrency value.
    b. Setting this parameter to true bypasses the resource group concurrent transaction limit so that queries can run immediately.
    c. When setting this parameter to true and running the query:
        i. Queries are run in resource groups. The resource group assignment for the query does not change.
        ii. 10 MB per query. Memory is allocated from resource group shared memory or global shared memory. If there is not enough shared memory to satisfy the memory allocation request, the query will fail. This parameter can be set for a single session. Cannot be set within a transaction or function.
3. You can set the server configuration parameter gp_resource_group_queuing_timeout to specify how long a transaction remains in the queue before Greenplum Database cancels it. The default timeout is zero, and Greenplum Queue processes transactions indefinitely.
    a. Cancels transactions that have been queued in the resource group for longer than the specified number of milliseconds, in milliseconds. This time limit applies to each transaction individually. The default is 0, meaning that transactions are queued indefinitely and never time out.
4. Note: The concurrency limit does not apply to resource groups for external components and must be set to zero (0).

Introduction to Memory Management

Let's take an example of how Greenplum resource group memory works.

The hardware configuration of the client's host's memory is " 30G physical memory and 8G SWAP space. First, take a look at the following figure, and then we will explain its meaning step by step .

Note : SWAP is memory swap space, or virtual memory, which is virtualized from a part of the space opened up on disk space. From the point of view of running speed and efficiency, if the customer's memory configuration is sufficient, it is not recommended for customers to use SWAP space. But if you use SWAP , you need to know that this part will participate in the calculation of memory management allocation.



Parameters: overcommit_ratio = 0.5

Memory for all application:
- total_memory = total_RAM * overcommit_ratio + total_Swap =30*0.5 + 8 = 23G

Parameters: gp_resource_group_memory_limit=0.9

The total memory for GreenPlum:
- total_memory_for_GP = total_memory * gp_resource_group_memory_limit=23*0.9=20.7G

Parameters: segment_count=4

The total memory for every segment:
- total_segment_memory = total_memory_for_GP/4=5.2G

Parameters: memory_limit=30
============
Parameters comes from: create resource group rg_new with (concurrency=5, cpu_rate_limit=20, memory_limit=30, memory_shared_quota=20, memory_spill_ratio=30)

Memory for every resource group (eg: RG: rg_new_2):
- resource_group_memory_per_segment = total_segment_memory*memory_limit/100 = 5.2*30/100 = 1.56G

So the Global shared memory will be (For example, several Resource groups was allocated 80 percentage of the whole memory, so for the global memory is left 20 percentage):
- Global shared memory =resource_group_memory_per_segment *(100-memory_limit_1-memory_limit_2-memory_limit_3-memory_limit_4)/100=1.56*20/100=312M

Parameter:
concurrency=5,memory_shared_quota=20
============
Parameters comes from: create resource group rg_new with (concurrency=5, cpu_rate_limit=20, memory_limit=30, memory_shared_quota=20, memory_spill_ratio=30)

Fixed memory quota for every slot (eg: Slot #4):
- transaction_fixed_memory_per_segment= resource_group_memory_per_segment * (100-memory_shared_quota)/(100*concurrency) = 1.56 * (100 - 20)/ (100*5) ~= 250M

Group shared memory quota:
- transaction_shared_memory_per_segment = resource_group_memory_per_segment * memory_shared_quota)/100 = 1.56 * 20/ 100 ~= 300M

So we can get the Maximum memory for a transaction:
- transaction_memory_per_segment = transaction_fixed_memory_per_segment + transaction_shared_memory_per_segment + global shared memory = 250+300+ global shared memory

Parameter:
memory_spill_ratio=30
============
Parameters comes from: create resource group rg_new with (concurrency=5, cpu_rate_limit=20, memory_limit=30, memory_shared_quota=20, memory_spill_ratio=30)

The limitation that will spill the memory to disk:
- transaction_initial_quota_per_segment = (resource_group_memory_per_segment * MEMORY_SPILL_RATIO)/(100*concurrency) = (1.56 * 30)/(100*5) ~= 94M

1. As can be seen from the example in the above figure, the parameters that can be obtained through hardware configuration and deployment are:

parameter	explain
total_RAM	physical memory of the host
total_Swap	host's swap space
segment_count	The number of segments deployed on the host

2. The parameters that need to be set by the system are:

Scope of influence of configuration item	parameter	explain
Global	Overcommit_ratio	The OS(Operation System) gives memory to all applications, and the rest of the memory is used by the OS itself. This Linux kernel parameter is set in /etc/sysctl.conf and identifies the percentage of RAM used by application processes; the rest is left to the operating system. Adjust settings as needed. If your memory utilization is too low, increase this value; if memory or swap usage is too high, lower this setting.
Global	overcommit_memory	This Linux kernel parameter is set in /etc/sysctl.conf, this file contains the kernel virtual memory accounting mode, which identifies how the operating system can allocate how much memory to the processes. The Greenplum Database system overcommit_memory must always be set to 2, which means that overcommit usage is not allowed. Its value range is "0,1,2", where: • 0: Heuristic overuse (this is the default value), allowing a certain amount of memory over commit, controlled by the kernel itself • 1: always overcommit, never check, allow any over commit • 2: Always check , never overuse, don't allow any over commit

Global	gp_resource_group_memory_limit	Greenplum Database allocated memory percentage that allocates to all applications. The default value is 0.7 (70 %).
RG	concurrency	Upper limit of allowed concurrent transactions
RG	cpu_rate_limit	Greenplum Database allocated the percentage of CPU of queries in a resource group. The amount used "account" is allocated to Greenplum Database in "Total CPU " percentage
RG	memory_limit	Greenplum Database allocated the percentage of "memory usage of queries in a resource group" in "total memory allocated to Greenplum Database"
RG	memory_shared_quota	The percentage of "shared memory within the group" that allocated to a resource group.
RG	memory_spill_ratio	A transaction contains multiple pieces of SQL. Each SQL may be divided into multiple motions. This value is the sum of the upper memory limits of all operators. Exceeding this value will spill to disk.

3. Through the configuration of "hardware configuration and deployment" and "parameters set by the system" obtained above, the status of various resources obtained in the relevant resource group can be obtained. The example is calculated according to the hardware configuration of the client's host's memory in the above figure " 30G physical memory and 8G SWAP space, as shown in the following table :

step	**Calculation formulas and examples**	**Example explanation**
Step1: Memory for all applications	Parameters: • total_RAM = 30G • total_Swap = 8G • overcommit_ratio = 0.5 Memory for all applications: • total_memory = total_RAM * overcommit_ratio + total_Swap =30*0.5 + 8 = 23G	In the host, there is 23G allocated to all the applications
Step2: Memory for Greenplum	Parameters: • gp_resource_group_memory_limit = 0.9 The total memory for GreenPlum: • total_memory_for_GP = total_memory * gp_resource_group_memory_limit =23*0.9=20.7G	The total memory allocated by Greenplum is 20.7G
Step3:	Parameters: • segment_count =4	The total allocated

Memory for every segment	The total memory for every segment: • total_segment_memory = total_memory_for_GP /4=5.2G	memory for every segment is 5.2G
Step4: Memory for every resource group (eg: RG: rg_new_2)	Parameters: • memory_limit =30 ○ Parameters comes from: create resource group rg_new with (concurrency=5, cpu_rate_limit=20, memory_limit=30, memory_shared_quota=20, memory_spill_ratio=30) Memory for every resource group (eg: RG: rg_new_2): • resource_group_memory_per_segment = total_segment_memory*memory_limit/100 = 5.2*30/100 = 1.56G So the Global shared memory will be （For example, several Resource groups was allocated 80 percentage of the whole memory, so for the global memory is left 20 percentage）： • Global shared memory = resource_group_memory_per_segment *(100-memory_limit_1-memory_limit_2-memory_limit_3-memory_limit_4)/100=1.56*20/100=312M	The resource group RG: rg_new_2, the total allocated memory is 1.56G. Within a segment, the "global shared memory" that can be shared by multiple resource groups is 312M. Global shared memory: It means to remove the memory allocated for all resource groups (eg : 80%), and the remaining part belongs to the global shared memory (eg : 100%-80%=20%):
Step5: Fixed memory quota for every slot (eg: Slot #4):	Parameter: • concurrency=5,memory_shared_quota=20 ○ Parameters comes from: create resource group rg_new with (concurrency=5, cpu_rate_limit=20, memory_limit=30, memory_shared_quota=20, memory_spill_ratio=30) Fixed memory quota for every slot (eg: Slot #4): • transaction_fixed_memory_per_segment= resource_group_memory_per_segment * (100-memory_shared_quota)/(100*concurrency) = 1.56 * (100 - 20)/ (100*5) ~= 250M Group shared memory quota:	The fixed memory that can be allocated to each query is: 250M. For the concurrent queries in the group, the memory that can be shared is 300M. From this, it can be obtained that

	• transaction_shared_memory_per_segment = resource_group_memory_per_segment * memory_shared_quota)/100 = 1.56 * 20/ 100 ~= 300M So we can get the Maximum memory for a transaction: • transaction_memory_per_segment = transaction_fixed_memory_per_segment + transaction_shared_memory_per_segment + global shared memory = 250+300+312=862M	the maximum amount of memory that a query can use is: 250+300+312=862M
Step6: Fixed memory for a Slot eg: Slot #5	Parameter: • memory_spill_ratio=30 ○ create resource group rg_new with (concurrency=5, cpu_rate_limit =20, memory_limit =30, memory_shared_quota =20, memory_spill_ratio =30) Start overflowing to the upper limit of disk : • transaction_initial_quota_per_segment = (resource_group_memory_per_segment * MEMORY_SPILL_RATIO)/(100*concurrency) = (1.56 * 30) /(100*5) ~= 94M	When a query's memory usage exceeds 94M, it will begin to spill memory to disk.

4. What does the operation mentioned "memory spill to disk" in Step 6 refer to?
    a. For example, create a table, insert 8,000,000 rows, the value range is 1~1,000,000, and sort it. During the sorting process, many intermediate calculation results will be generated, which will occupy a large amount of memory. For example, if there is no limit, the main memory that this query needs to use is about 200M or more.
    b. If the "upper limit of spill to disk" is set to 125M, GP will spill the intermediate results related to sorting to disk storage after the memory usage reaches 125M, it can reduce memory usage. The final query usage may be around 128M. Among them, the use of 3M (128M-125M=3M) is the occupancy that cannot be spilled to the disk part.
    c. If the "upper limit of spill to disk" is set to 3M, GP will spill the intermediate results related to sorting to disk storage after the memory usage reaches 3M, reducing the memory usage. The final query usage may be around 6M. Among them, the use of 3M (6M-3M=3M) is the occupancy that cannot be spilled to the disk part.

5. What kind of operations will be spilled to disk?
    a. Each plan node in a Query plan can be divided into two categories:
        i. Memory intensive: sort, hash, materialize, ...
        ii. Not memory intensive: all others, like scans, motions, ...
    b. When executed memory-intensive plannodes, need to produce a large number of intermediate temporary results, which may require external storage, which we call spill

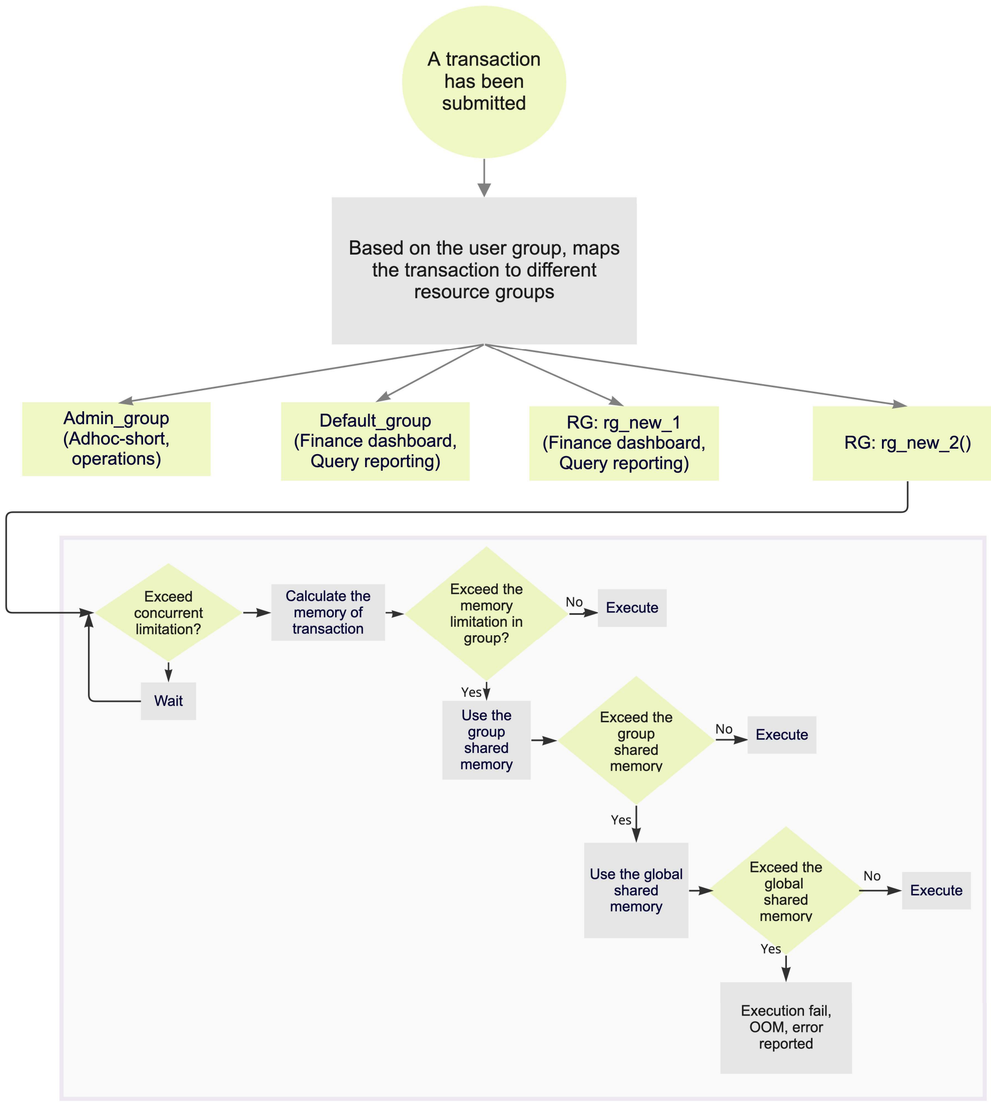
to disk. Control when to overflow by adjusting memory_spill_ratio to increase or decrease. When memory_spill_ratio is greater than 0, it represents the percentage of resource group memory allocated to the query.

c. When using, be aware that if concurrency is high, this amount of memory may be small even if memory_spill_ratio is set to a maximum value of 100. When you set memory_spill_ratio to 0, Greenplum Database uses the statement_mem setting as the upper limit for spilling into memory.

d. Another issue to be aware of is that after the memory usage "exceeds the spill to disk limit", the memory may still increase. Because only the types of operations that need to produce a large number of intermediate temporary results are subject to this control, there are many operations that cannot spill to disk and continue to consume memory. Therefore, it is still necessary to consider comprehensively when configuring.

e. Therefore, the setting of the upper limit of spill to disk parameter "memory_spill_ratio" will affect the overall performance of the memory. In use, it needs to be tuned according to the real situation of the customer environment.

6. After the above configuration, we probably get that the maximum memory that a query can use is the sum of the three values of "fixed allocation value of memory in the group" + "shared memory in the group" + "global shared memory" . This memory is based on a first-come, first-served basis for all queries. The specific process can be seen in the following figure. OOM occurs when the memory exceeds the sum of all available memory.


A transaction has been submitted
Based on the user group, maps the transaction to different resource groups
Admin_group (Adhoc-short, operations)
Default_group (Finance dashboard, Query reporting)
RG: rg_new_1 (Finance dashboard, Query reporting)
RG: rg_new_2()
Exceed concurrent limitation?
Wait
Calculate the memory of transaction
Exceed the memory limitation in group?
No
Execute
Yes
Use the group shared memory
Exceed the group shared memory
No
Execute
Yes
Use the global shared memory
Exceed the global shared memory
No
Execute
Yes
Execution fail, OOM, error reported

7. For queries managed by resource groups configured to use the vmtracker memory auditor, Greenplum Database supports automatic termination of queries based on the amount of memory used by the query. The relevant configuration parameters are runaway_detector_activation_percent .
    a. For resource group-managed queries, this parameter determines Greenplum when to terminate running queries based on the amount of memory used by the query.
    b. The value range is 0-100, and the default value is 90.
    c. If the value is set to 100 or 0: this feature is disabled, suppressing the automatic termination of queries based on the percentage of memory used.
    d. When the resource group is enabled - this parameter sets the global shared memory usage of the resource group. The resource group has a global shared memory pool. For example, if you have 3 resource groups configured with memory_limit values of 10, 20 and 30, then the global shared memory is 40% = 100% - (10% + 20% + 30 %).

e. If the percentage of used global shared memory exceeds the specified value, Greenplum Database terminates the query based on memory usage, selecting the query from the queries managed by the resource group configured to use the vmtracker memory auditor. Greenplum Database starts with the query that consumes the largest amount of memory. The query will terminate until the percentage of global shared memory used falls below the specified percentage.

Introduction to CPU Management

In the resource group management based on the roles, the resource management of the CPU uses the Linux-based CGroup. GreenPlum allocates CPU resources in two ways :

- By assigning specific CPU cores to resource groups
- By assigning a percentage of CPU resources to resource groups .

You can also use both CPU resource allocation modes in a Greenplum Database cluster, and you can change the CPU resource allocation mode for resource groups at runtime.

The relevant parameters are :

CPU_RATE_LIMIT	The percentage of CPU resources available to this resource group.
CPUSET	The number of CPU cores reserved for this resource group

Allocate CPU resources by percentage of resources

1. For the mode allocated according to the percentage of CPU resources, the parameters that need to be set by the system are :

Scope of influence of configuration item	**parameter**	**explain**
Global	gp_resource_group_cpu_limit	The percentage of CPU resources allocated for GP within all applications. The remaining unreserved CPU resources are used for the operating system kernel and Greenplum Database helper daemon . The default gp_resource_group_cpu_limit value is .9 (90 %).
RG	concurrency	The upper limit of the allowed concurrent query
RG	cpu_rate_limit	The percentage value that "CPUs assigned to a resource group within the amount used "account" allocated to GreenPlum Database total CPU usage".

2. Through the configuration of the "parameters set by the system" above, you can get the information of various resources obtained in this related resource group. For example :
   - gp_resource_group_cpu_limit = 0.9
   - Parameters comes from: create resource group rg_new with (concurrency=5, cpu_rate_limit =20, memory_limit =30, memory_shared_quota =20, memory_spill_ratio=30)
   - It can be concluded that the CPU resources allocated by this resource group are : gp_resource_group_cpu_limit*cpu_rate_limit/100 = 0.9*20/100=0.18.
   - Note: A resource group specified the range of cpu_rate_limit percentage is an integer value between 1 and 100. For specified by all resource groups defined in the Greenplum Database cluster, the sum of cpu_rate_limit cannot exceed 100.

3. For the mode allocated according to the percentage of CPU resources, there are also two types of policies, which are defined by gp_resource_group_cpu_ceiling_enforcement global variables to configure .
   - The first is the elastic mode
     - This mode is active when gp_resource_group_cpu_ceiling_enforcement is set to false (the default).
     - Its resilience lies in Greenplum Database can allocate CPU resources of idle resource groups to other busy resource groups. When a previously idle resource group needs CPU resources, the CPU resources will be reassigned to that resource group. If multiple resource groups are busy, the CPU resources of the idle resource groups are allocated according to the proportion of CPU_RATE_LIMIT. For example, a resource group with a CPU_RATE_LIMIT of 40 will allocate twice as much extra CPU resources as a resource group with a CPU_RATE_LIMIT of 20.
   - The second is the Ceiling Enforcement mode
     - This mode takes effect when "gp_resource_group_cpu_ceiling_enforcement" is set to true.
     - Force the CPU resource used by the resource group to not exceed the defined CPU_RATE_LIMIT value to avoid using the CPU elastic properties of the burst feature.

Allocate CPUSET resources by number of cores

For the CPUSET mode according to the number of cores, the parameters that need to be set by the system are:

Scope of influence of configuration item	**parameter**	**explain**
RG	concurrency	The upper limit of the allowed concurrent query

RG	cpuset	CPU cores reserved for the resource group. User-specified CPU cores must be available in the system and must not overlap any CPU cores reserved for other resource groups . When configuring CPUSET, specify a comma-separated list of single-core numbers or an interval of numbers. Core numbers/intervals must be enclosed in single quotes, e.g. "1,3-4".

1. Both modes of CPU resource allocation can be used simultaneously in a Greenplum Database cluster. For example : For example, the CPU resource of a host is 32 cores, and 4 segments are deployed on it. The configuration of the resource group above it is as follows :

```
gp_resource_group_cpu_limit = 0.9

=# postgres =# select * from gp_toolkit.gp_resgroup_config ;
groupid | groupname | concurrency | cpu_rate_limit | memory_limit | memory_shared_quota |
memory_spill_ratio | memory_auditor | cpuset ---------+---------------+-------------+----------------+------------
--+---------------------+--------------------+----------------+--------
6437 | default_group | 20 | 50 | 0 | 80 | 0 | vmtracker | -1
6438 | admin_group | 10 | -1 | 10 | 80 | 0 | vmtracker | 1-3
16388| rg_new |10|20|10|80|0| vmtracker |-1
(3 rows)
```

2. From the above it can be seen that:
   1. 90% of the host's CPU resources are allocated to GreenPlum Database.
   2. Where resource group admin_group using CPUSET , the assigned cores are 1-3. Note that when configuring CPUSET for a resource group, Greenplum Database disables the group's CPU_RATE_LIMIT and sets the value to -1.
   3. The resource group default _group used mode is based on the CPU resource percentage allocation , the cpu_rate_limit allocation ratio is 50%.
   4. The resource group rg_new used mode is based on the percentage allocation of CPU resources, The cpu_rate_limit allocation ratio is 20%.
   5. The base number of CPU resource percentage allocation, which refers to the part other than the cores allocated by CPUSET. For mixed configurations of CPUSET and CPU percentage, this limit is still observed here: the sum of the cpu_rate_limit specified by all resource groups defined in the Greenplum Database cluster cannot exceed 100.

3. Note on the use of CPUSET :
   1. When using the CPUSET mode by the number of cores , avoid wasting the system's CPU .
      a. Because when it is specified as CPUSET, the core can only be used by this resource group and cannot be used by other resource groups. So, if the specified core cannot be confirmed whether it can be fully used, the mode of CPU resource percentage allocation will be more efficient .

2. Core 0 in CPUSET has its special function, so use this number as little as possible:
    a. Because the system will allocate CPU core 0 to the default admin_group and default_group. In this case, if other groups are also assigned CPU core 0, those resource groups will share core 0 with admin_group and default_group.
    b. If Greenplum Database cluster is restarted after a physical node replacement, and the new node does not have enough cores to service all CPUSET resource groups, CPU core 0 is automatically assigned to these groups to avoid system startup failures.
3. When using CPUSET by core number , use the lowest possible core number .
    a. In the scenario "Replace a Greenplum Database node and the new node has fewer CPU cores than the original node" , or in the scenario "Backup the database and want to restore it on a cluster with a node with fewer CPU cores", if the new node cannot find the corresponding CPUSET number of cores that will fail.

Monitoring Commands and Interpretation

1. Via "gp_resgroup_config gp_toolkit" system view to view the configuration and status of master resources.
    a. View Resource Group Limits
        i. =# SELECT * FROM gp_toolkit.gp_resgroup_config;
    b. View resource group query status and CPU/memory usage
        i. =# SELECT * FROM gp_toolkit.gp_resgroup_status;
    c. View resource group CPU/memory usage per host
        i. =# SELECT * FROM gp_toolkit.gp_resgroup_status_per_host;
    d. View resource group CPU/memory usage for each segment
        i. =# SELECT * FROM gp_toolkit.gp_resgroup_status_per_segment;
    e. View resource groups assigned to roles
        i. =# SELECT rolname, rsgname FROM pg_roles, pg_resgroup
           WHERE pg_roles.rolresgroup=pg_resgroup.oid;
2. Use the gp_toolkit view to check the resource usage of resource groups and monitor how the group is working. The fields are explained:

rsgname	resource group name
groupid	resource group oid
cpu	CPU utilization for this resource group
memory_used	Real-time memory usage of this resource group , including fixed allocated memory, intra-group shared memory, and global shared memory ;

memory_available	In this resource group, how much memory is still in the fixed allocated memory in the group and the shared memory in the group, excluding the global shared memory. So when this value is negative, it means that the global shared memory part is being used.
memory_quota_used	The memory usage of fixed allocation in this resource group, it does not refer to how much fixed allocation memory is used, it just refers to how many fixed allocation values are initialized.
memory_quota_available	A fixed allocation of this resource group, the total amount of memory available
memory_shared_used	Group shared memory used by this resource group , including global shared memory
memory_shared_available	The amount of shared memory available to this resource group, excluding global shared memory, so, like memory_available, it may be negative if global shared memory is used.

3. The value of some parameters will have a negative value to indicate whether shared memory is used. For example, if "memory_available" has a negative value:
   a. The following are 4 concurrent queries, and the upper limit of the concurrency setting is 10. It can be seen from the first line that for the resource group "admin_group" on "segment_id = 2" , memory_used =512, memory_available =-175, indicating that it uses a total of 512M of memory, of which the global shared memory accounts for 175M. memory_available = 30, indicating that 30M is allocated when the fixed allocation memory in the group is initialized (although, not necessarily all used already). memory_shared_used = 488, memory_shared_available =-211, indicating that its shared memory uses a total of 488M, of which the global sharing uses 211M, and the group shared memory uses 377M (488-211=377).

```
Query=4, concurrency=10
postgres=# select * from gp_toolkit.gp_resgroup_status_per_segment;
  rsgname    | groupid | hostname | segment_id |  cpu  | memory_used |
memory_available | memory_quota_used | memory_quota_available |
memory_shared_used | memory_shared_available
```

```
---------------+---------+----------+------------+-------+-------------+------------------+------------------+-
----------------------+--------------------+-------------------------
admin_group   |   6438 | stable   |       2 | 90.01 |       512 |          -175 |             30 |
30 |            488 |                  -211
default_group |   6437 | stable   |      -1 |  0.00 |         0 |             0 |              0 |
0 |              0 |                  0
default_group |   6437 | stable   |       2 |  0.00 |         0 |             0 |              0 |
0 |              0 |                  0
admin_group   |   6438 | stable   |       0 | 90.01 |       512 |          -175 |             30 |
30 |            488 |                  -211
admin_group   |   6438 | stable   |       1 | 90.01 |       512 |          -175 |             30 |
30 |            488 |                  -211
default_group |   6437 | stable   |       0 |  0.00 |         0 |             0 |              0 |
0 |              0 |                  0
admin_group   |   6438 | stable   |      -1 | 90.97 |        18 |           319 |             30 |
30 |              0 |                  277
default_group |   6437 | stable   |       1 |  0.00 |         0 |             0 |              0 |
0 |              0 |                  0
(8 rows)
```

Tuning parameters

In the process of using Greenplum in the resource group, it is necessary to perform parameter tuning according to the customer's time deployment environment and specific types of job requirements. The parameters to be adjusted mainly include (some configuration items have been mentioned in the first half of this article, summarized again here):

Scope of influence of configuration item	**Parameter**	**Explain**
Global	Overcommit_ratio	OS gives memory to all applications, and the rest of the memory is used by the OS itself . This Linux kernel parameter is set in /etc/sysctl.conf and identifies the percentage of RAM used by application processes; the rest is left to the operating system. Adjust settings as needed. Increase the value if your memory utilization is too low; decrease the setting if memory or swap usage is too high

Global	overcommit_memory	This Linux kernel parameter is set in /etc/sysctl.conf, this file contains the kernel virtual memory accounting mode, which identifies how the operating system can allocate how much memory to a process. The Greenplum Database system's overcommit_memory must always be set to 2, which means that overcommit usage is not allowed. The allowable range of its values is "0,1,2", where: • 0: Heuristic overuse (this is the default value), allowing a certain amount of memory over commit, controlled by the kernel itself • 1: always overcommit, never check, allow any over commit • 2: Always check , never overuse, don't allow any over commit
Global	gp_resource_group_memory_limit	Greenplum Database allocated memory percentage that allocates to all applications. The default value is 0.7 (70 %).
Global	gp_resource_group_cpu_limit	The percentage of CPU resources allocated for GP within all applications. The remaining unreserved CPU resources are used for the operating system kernel and Greenplum Database helper daemon . The default gp_resource_group_cpu_limit value is .9 (90 %).
Global	gp_workfile_limit_files_per_query	The maximum number of temporary spill files (work files) allowed per query. Spill files are created when a query requires more memory than is allocated. When the limit is exceeded, the query will be terminated. The default value is zero, which allows an unlimited number of spill files and it may fill up the filesystem.
Global	gp_workfile_compression	If there are a large number of spill files, set gp_workfile_compression to compress the overflow files. Compressing overflow files may help avoid overloading the disk subsystem with IO operations.
Global	statement_mem	When memory_spill_ratio is 0, Database uses statement_mem to determine the upper limit to spill files into memory.
Global	runaway_detector_activation_percent	For resource group-managed queries, this parameter determines when Greenplum Database terminates a running query based on the amount of memory used by

		the query. The value range is 0-100, and the default value is 90. If the value is set to 100 or 0: this feature is disabled, suppressing the automatic termination of queries based on the percentage of memory used. When the resource group is enabled - this parameter sets the global shared memory usage of the resource group. The resource group has a global shared memory pool. For example, if you have 3 resource groups configured with memory_limit values of 10, 20 and 30, then the global shared memory is 40% = 100% - (10% + 20% + 30 %). If the percentage of used global shared memory exceeds the specified value, Greenplum Database terminates the query based on memory usage, selecting the query from the queries managed by the resource group configured to use the vmtracker memory auditor. Greenplum Database starts with the query that consumes the largest amount of memory. The query will terminate until the percentage of global shared memory used falls below the specified percentage.
Session	gp_resource_group_bypass	Enable or disable Greenplum Database resource group concurrent transaction limit. The default value is false, which enforces a limit on the resource group transaction concurrency value . Setting this parameter to true bypasses the resource group concurrent transaction limit so that queries can run immediately. When setting this parameter to true and running the query: 1. Queries are run in resource groups. The resource group assignment for the query does not change. 2. 10 MB per query. Memory is allocated from group shared memory or global shared memory. If there is not enough shared memory to satisfy the memory allocation request, the query will fail. This parameter can be set for a single session. Cannot be set within a transaction or function.

Transaction	gp_resource_group_queuing_timeout	Cancels transactions that have been queued in the resource group for longer than the specified number of milliseconds, in milliseconds. This time limit applies to each transaction individually. The default is 0, meaning that transactions are queued indefinitely and never time out.
Global/session	gp_resgroup_memory_policy	Used by resource groups to manage memory allocation for query operations. The default is eager_free. 1. When set to eager_free, Greenplum Database allocates memory among operators more optimally by reallocating memory freed by operators that have completed processing to operators in later query stages. 2. When set to auto, Greenplum Database uses the resource group memory limit to allocate memory among query operators, allocating a fixed size of memory to non-memory-intensive operators and the rest to memory-intensive operators.
RG	concurrency	The upper limit of allowed concurrent transactions.
RG	cpu_rate_limit	Greenplum Database allocated the percentage of CPU of queries in a resource group. The amount used "account" is allocated to Greenplum Database in "Total CPU " percentage
RG	memory_limit	Greenplum Database allocated the percentage of "memory usage of queries in a resource group" in "total memory allocated to Greenplum Database"
RG	memory_shared_quota	The percentage of "shared memory within the group" that allocated to a resource group.
RG	memory_spill_ratio	A transaction contains multiple pieces of SQL. Each SQL may be divided into multiple motions. This value is the sum of the upper memory limits of all operators. Exceeding this value will spill to disk.

Suggestions for tuning :

1. Note that if you set memory_shared_quota close to 100: this will result in the fixed allocation memory almost 0, and transaction_initial_quota_per_segment (the upper bound that starts spilling to disk) may be too small to run most queries.

2. Note that if memory_spill_ratio is set to a very large value, this will result in a large transaction_initial_quota_per_segment (starting to spill to the upper limit of disk), many queries may use too much memory, exhaust the slot memory limit, or the resource group memory limit, eventually lead to failure.
3. Regarding tuning memory_shared_quota (shared memory within a group): When setting memory_shared_quota, consider the workload of the resource group. If almost all queries require a similar memory size, then memory_shared_quota can be as small as 0. If your resource group will support mixed workloads with small and large memory queries, then you can set memory_shared_quota to a relatively large value so that small memory queries can run using only a fixed portion of memory, while large memory Queries can use both fixed and group-shared parts.

The following are some descriptions of problem scenarios, for different scenarios with different tuning suggestions:

Scenarios description	**Analysis and Recommendations**
If there is too much memory spilled to the disk file system . 1. For example , the customer's system has a lot of memory, but the memory is not fully utilized, and the system is still generating a lot of memory spill to disk. 2. Moreover, the customer found that the memory usage is: the estimated memory usage value should be greater than the fixed memory allocated in the SLOT, but in fact the intra-group shared memory and global shared memory in the RG are not used.	1. Consider whether the memory_spill_ratio setting needs to be increased. If memory is rich, reducing the amount of spills to disk. It allows more operations to run faster in memory. 2. At the same time, consider whether it is necessary to reduce concurrency, increase the total amount of memory (memory_limit) that can be allocated by the resource group, and ensure that the amount of memory that can be used by each query is more abundant. 3. Increase or decrease the memory Greenplum Database allocates to queries. When memory_spill_ratio is greater than 0, it represents the percentage of resource group memory allocated to the query operator. If concurrency is high, this amount of memory may be small even with memory_spill_ratio set to a maximum value of 100. When you set memory_spill_ratio to 0, Greenplum Database uses the statement_mem setting to determine the upper limit to spill files into memory.
Query cannot work properly due to insufficient memory , resulting in memory leak OOM.	1. Understand the approximate memory usage required by the query by "reducing concurrency and increasing resource group memory" . 2. Analyze the type of Query to see if there will be a large number of intermediate results using memory. If it does exist, set a reasonable value of memory_spill_ratio to spill a lot of memory to disk, thereby reducing memory usage.
It is not easy to find the cause of memory leak OOM. It feels that the amount of configuration is	1. Analyze machine configuration. Whether the hardware configuration of the hosts are different between the Master and segment, or between segments. In the current implementation of resource group management, query_mem

enough, but there are still segments that generate OOM.	in the plan tree uses the system memory of QD and master segment number, not QE's own system memory and the QE segment number. This can lead to various issues like OOM, underutilization of resources, etc. 2. If the above scenario exists, GUC needs to be turned on : gp_resource_group_enable_recalculate_query_mem , then the client decides whether to recalculate query_mem proportionally on QE, and repopulate operatorMemKB in the planning tree according to this value. So that QE can calculate its correct parameter configuration independently . 3. For users new to RG, it is strongly recommended to turn on GUC: gp_resource_group_enable_recalculate_query_mem . For customers who are already using RG, it is strongly recommended to try to turn it on based on some experiments to obtain more accurate resource allocation.
A memory leak OOM and the system is under high concurrent load.	1. Analyzing the system, when the system starts to clean up runaway sessions, when the system is under high concurrent load, OOM error messages may still be thrown. Due to the current design, we cannot speed up the Runsway Session cleanup process at the code level . 2. The solution to this problem is to adjust the runaway_detector_activation_percent to 0.85 or 0.8, or even lower, and increase the available memory of the node.
If you need a scenario: some transaction requests only run during a certain period of time, other times do not run.	1. Dynamically change the resource group configuration to match the actual workload and time requirements of the group. 2. The configuration of resource groups can be dynamically changed regularly, and resource allocation can be customized at different time periods to achieve higher efficiency. For example: change the resource configuration in the group; or add or delete different groups to meet the rich and diverse scene collocation.
For different queries, there are requirements: the settings for spill to disk are different.	1. A lower statement_mem setting (for example, in the 10MB range) can improve the performance of queries with low memory requirements. 2. Using the two parameters memory_spill_ratio and statement_mem, set the server configuration parameters so that the previous settings are overridden in some resource groups. For example, set the memory_spill_ratio of a resource group to 0; statement_mem = '10 MB'. While other resource groups, use other flexible memory_spill_ratio setting.
The memory_spill_ratio has been set to 100, why is there still a large number of files overflowing	Analysis : 1. According to the calculation formula of the spill to disk limit introduced above, when memory_spill_ratio is set to 100, the

into memory . But in fact, then the intra-group shared memory and global sharing in the RG are not used either.	maximum value is "the number of memory / concurrencies allocated by a resource group ". A small portion of the system's global memory is used. 2. According to the example mentioned in the article, when memory_spill_ratio is set to 100, the upper limit of overflow to disk is: transaction_initial_quota_per_segment = (resource_group_memory_per_segment * MEMORY_SPILL_RATIO)/(100*concurrency) = (1.56 * 100)/(100*5) ~ = 312M Recommendation: 1. To increase the upper limit of spill to disk by reducing concurrency and increasing the total amount of memory (memory_limit) that the resource group can allocate. 2. Set the memory_spill_ratio for this resource group to 0. At this time, statement_mem is used as the upper limit value of the overflow file, which can be set to increase the upper limit value of the spill file to the disk. 3. Note that it is also mentioned in this article that it is still possible for memory to increase after memory usage "exceeds the spill to disk limit". Because only the types of operations that need to produce a large number of intermediate temporary results are subject to this control, there are many operations that cannot spill to disk and continue to consume memory. Therefore, when reconfiguring, it is still necessary to comprehensively consider avoiding the situation of OOM.
It is found that the actual concurrent number of the resource group is greater than the configured concurrency upper limit.	1. Because the Resource group does not enforce resource restrictions on the SET, RESET and SHOW commands. 2. Greenplum Database resource groups concurrency limitation can be enabled or disabled through the configuration item " gp_resource_group_bypass ". 3. So, you will see that in some cases the actual number of concurrencies is greater than the set concurrency limit value. This is normal.
After the customer upgrades, they notice a drop in performance.	There are many factors that affect performance degradation. The following is a possible factor: 1. Some customers have upgraded to GP 6, and the performance is much slower. After many investigations, it was found that SWAP was not closed. Using SWAP will affect performance. Some operating systems are enabled by default, and customers are advised to disable it. 2. SWAP is memory swap space, or virtual memory, which is virtualized from a part of the space opened up on disk space. From the point of view of running speed and efficiency, if the customer's memory configuration is sufficient, it is not recommended for customers to use SWAP space. But if a

	customer uses SWAP, he needs to know that this part will participate in the calculation of memory management allocation.