SAP HANAä Memory Usage Explained

Memory is a critical SAP HANA resource. This paper explains the basic memory concepts and how to explore the memory consumption of a SAP HANA system

Chaim Bendelac, SAP SE
Panos Kokkalis, SAP SE

V5, January 2017
Updated for SAP HANA SPS12 (Rev 122+)

Table of Contents

Legal Disclaimer

1 Introduction

SAP HANA

SAP HANAä is a leading in-memory database and data management platform, specifically developed to take full advantage of the capabilities provided by modern hardware to increase application performance. By keeping all relevant data in main memory (RAM), data processing operations are significantly accelerated.

The key performance indicators of SAP HANA appeal to many of our customers, and thousands of deployments are in progress. SAP HANA has become the fastest growing product in SAP's history.

About this Document

A fundamental SAP HANA resource is memory. Understanding how the SAP HANA system requests, uses and manages this resource is crucial to the understanding of SAP HANA. SAP HANA provides a variety of memory usage indicators, to allow monitoring, tracking and alerting. This paper explores the key concepts of SAP HANA memory utilization, and shows how to understand the various memory indicators.

2 What is Memory used for?

SAP HANA consists of a number of processes running on the SUSE Linux operation environment. Under Linux, the operating system is responsible for the reservation of memory to all processes. When SAP HANA starts up, the OS reserves memory for the program code (sometimes called the *text),* the program stack, and static data. It then dynamically reserves additional data memory[1] when requested by the SAP HANA memory manager. Dynamically allocated memory consists of heap memory and shared memory.

As an in-memory database, it is particularly critical for SAP HANA to carefully manage and track its consumption of memory. For this purpose, SAP HANA manages its own data memory pool by requesting memory from the OS, possibly in advance of using it. The memory pool is used to store all the in-memory data and system tables, thread stacks, temporary computations and all the other data structures required for managing a database.

At any given point, only parts of the memory pool are really in use. SAP refers to the *total amount of memory actually in use* as the SAP HANA Used Memory. This is the most precise indicator of the amount of memory that SAP HANA uses.

The SAP HANA Used Memory, consisting of code, stack and data, is shown below:

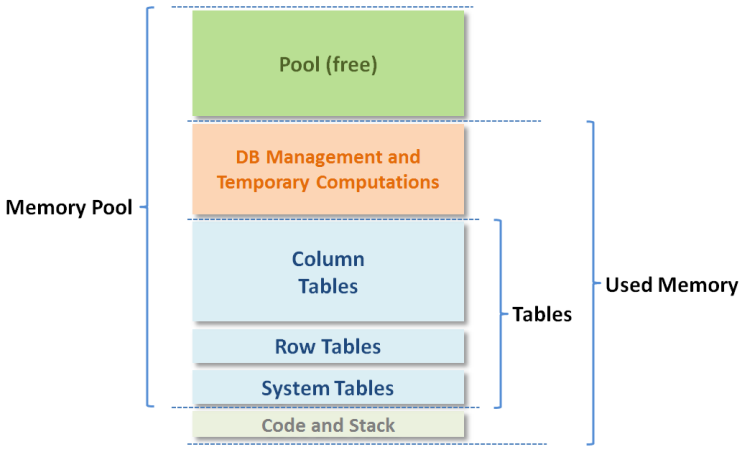

Figure 1 – SAP HANA Used Memory

Since the code and program stack size are about 6 GB, almost all of the Used Memory is in fact used for storing tables, computations and database management.

It is possible to *partition* large tables, and even distribute these partitions over multiple hosts. In such distributed scenarios, this discussion of memory usage applies to each of the hosts of an SAP HANA system, separately.

[1] Dynamically allocated memory consists of *heap* memory and *shared* memory.

Memory Sizing

Memory *sizing* is the process of estimating, in advance, the amount of memory that will be required to run a certain workload on SAP HANA. To understand memory sizing, you will need to answer the following questions:

1. What is the size of the data tables that will be stored in SAP HANA?
   You may be able to estimate this based on the size of your existing data, but unless you precisely know the compression ratio of the existing data and the anticipated growth factor, this estimate may only be partially meaningful.
2. What is the expected compression ratio that SAP HANA will apply to these tables?
   The SAP HANA Column Store automatically uses a combination of various advanced compression algorithms (dictionary, LRE, sparse, and more) to best compress each table column separately. The achieved compression ratio depends on many factors, such as the nature of the data, its organization and data-types, the presence of repeated values, the number of indexes (SAP HANA requires fewer indexes), and more.
3. How much extra working memory will be required for DB operations and temporary computations?
   The amount of extra memory will somewhat depend on the size of the tables (larger tables will create larger intermediate result-tables in operations like joins), but even more on the expected work load in terms of the number of users and the concurrency and complexity of the analytical queries (each query needs its own workspace).

SAP HANA provides some control over compression, via the "optimize_compression" configuration section of the index server. In general, HANA automatically selects the best compression technique by studying the data stored in each column of the table.

SAP Notes 1514966, 1872170 and 2296290 provide additional tools and information to help you size the required amount of memory, but the most accurate method is ultimately to import several representative tables into a SAP HANA system, *measure* the memory requirements, and extrapolate from the results.

The next section will explain how to measure and understand SAP HANA Used Memory. This description corresponds to SAP HANA revision 80 (SAP HANA 1.0 SPS8) or later.

3 Used Memory

As explained above, Used Memory is the *total amount of memory currently in use* by SAP HANA. This is the most precise indicator of the amount of memory that SAP HANA requires at any time.

To display the current size of the Used Memory, you can use the following simple SQL statement (for example with the SAP HANA Studio SQL editor):

```
select HOST, round(INSTANCE_TOTAL_MEMORY_USED_SIZE/(1024*1024*1024), 2) as "Used Memory GB"
from M_HOST_RESOURCE_UTILIZATION
```

This value provides a single summary result per host.

A SAP HANA system consist of multiple services that all consume some memory. Of particular interest is the "indexserver" service, the main database service. The indexserver holds all the data tables and temporary results, and therefore dominates the SAP HANA Used Memory.

You can drill down into Used Memory, and obtain the amount of indexserver Used Memory as follows[2]:

```
select HOST, round(TOTAL_MEMORY_USED_SIZE/(1024*1024*1024), 2) as "Used Memory GB"
from M_SERVICE_MEMORY where SERVICE_NAME = 'indexserver'
```

This will list all the hosts in the system, and the amount of indexserver Used Memory per host. Similarly, the memory consumption of other services (like the xsengine service) can be examined.

[2] Note that the amount of overall Used Memory is not simply the sum of used memory by all the SAP HANA services, as will be explained in section 6.

Used Memory over time

The value of Used Memory represents a current measurement, but it is ultimately more important to understand the behavior of Used Memory over time and under peak loads.

A snapshot copy of Used Memory is periodically saved in the HOST_RESOURCE_UTILIZATION_STATISTICS system table, providing you with a very useful time-series (for the last 42 days).

For instance, to see the *peak* amount of Used Memory in the past 42 days:

```
select top 1 HOST, SERVER_TIMESTAMP, round(INSTANCE_TOTAL_MEMORY_USED_SIZE/(1024*1024*1024), 2) as
"Peak Used GB" from _SYS_STATISTICS.HOST_RESOURCE_UTILIZATION_STATISTICS order by "Peak Used GB" desc
```

In a multi-host system, you can filter the query by host (`where HOST = 'myhost1' and …`).

You can use the time series to create utilization graphs over time, or to drill down into a particular period in the past. For instance, the following query shows the value of Used Memory at 7:00 AM during each of the last 30 days:

```
select top 30 HOST, SERVER_TIMESTAMP, round(INSTANCE_TOTAL_MEMORY_USED_SIZE/(1024*1024*1024), 2) as
"Used Memory GB" from _SYS_STATISTICS.HOST_RESOURCE_UTILIZATION_STATISTICS
where hour(SERVER_TIMESTAMP) = 7 and minute(SERVER_TIMESTAMP) = 0 order by SERVER_TIMESTAMP desc
```

Again, you can filter by host as appropriate.

Creatively, you can use this time-series to examine memory usage yesterday between 9AM and 9:15 AM, or find the peak-of-the day in the last week, etc.

A separate time-series, HOST_SERVICE_MEMORY, can be used to examine memory usage by service, for instance by focusing on the indexserver memory usage only.

Peak Used Memory

The SAP HANA database has a special used memory indicator called peak used memory. As the value for used memory is a current measurement, peak used memory allows you to keep track of the maximum value for used memory over time.

It shows the *peak* amount of Used memory since system restart and is not resettable (since SPS9).

You can query this indicator usnig the following sql statement:

```
select HOST, round(INSTANCE_TOTAL_MEMORY_PEAK_USED_SIZE/(1024*1024*1024), 2) as "Peak Used Memory GB"
from M_HOST_RESOURCE_UTILIZATION
```

4 SAP HANA Tables

The SAP HANA database supports two types of table: those that store data either column-wise (column tables) or row-wise (row tables).

Memory Management in the Column Store

The column store is the part of the SAP HANA database that manages data organized in columns in memory. Tables created as column tables are stored here.

The column store is optimized for read operations but also provides good performance for write operations. This is achieved through two data structures: main storage and delta storage.

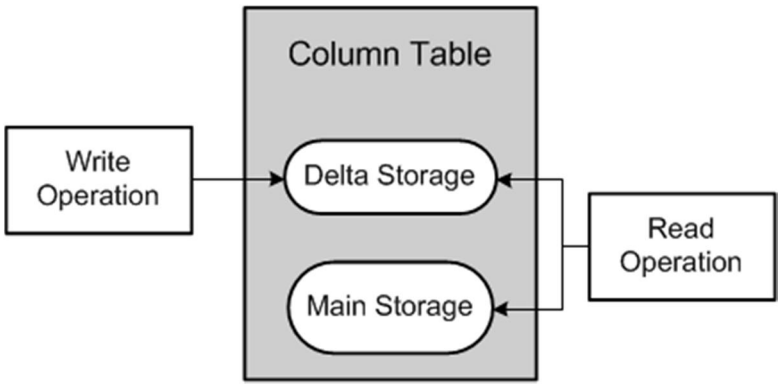

Figure 2 - Column Store Structures

The main storage contains the main part of the data. Here, efficient data compression is applied to save memory and speed up searches and calculations. Write operations on compressed data in the main storage would however be costly. Therefore, write operations do not directly modify compressed data in the main storage. Instead, all changes are written to a separate data structure called the delta storage. The delta storage uses only basic compression and is optimized for write access. Read operations are performed on both structures, while write operations only affect the delta. Changes collected in the delta storage are moved to the read-optimized main storage via a process called delta merge. For more information see the SAP HANA Administration Guide.

The dominant part of the used memory in the SAP HANA database is the space used by data tables. Separate measurements are available for column-store tables and row-store tables. These measurements are discussed below.

Column Tables

The following simple query provides a high-level overview of the amount of memory used for column tables:

```
select round(sum(MEMORY_SIZE_IN_TOTAL)/1024/1024) as "Column Tables MB Used" from M_CS_TABLES
```

Or, providing per-schema details:

```
select SCHEMA_NAME as "Schema", round(sum(MEMORY_SIZE_IN_TOTAL) /1024/1024) as "MB Used" from
M_CS_TABLES GROUP by SCHEMA_NAME order by "MB Used" desc
```

The SAP HANA database loads column-store tables into memory column by column only upon use. This is sometimes called "lazy loading". This means that columns that are never used will not be loaded and memory waste is avoided. When the SAP HANA database runs out of allocable memory, it will try to free up some memory by unloading unimportant data (such as caches) and even table columns that have not been used recently. Therefore, if it is important to measure precisely the total, or worst-case, amount of memory used for a particular table, it is important to ensure that the table is first fully loaded into memory. You can do this by loading the table into memory.

```
load <TABLE_NAME> all
```

You can use the following technique to examine the amount of memory consumed by a specific table. This also shows which of its columns are loaded, and the compression ratio that was accomplished. For example, list all tables for schema “SYSTEM”:

```
select TABLE_NAME as "Table", round(MEMORY_SIZE_IN_TOTAL/1024/1024) as "MB Used" from M_CS_TABLES where
SCHEMA_NAME = 'SYSTEM' order by "MB Used" desc
```

Or drill down into columns of a single table, for instance the table "LineItem", to view the actual size of the data, the “delta changes” and the compression ratio for each of its columns.

```
select COLUMN_NAME as "Column", LOADED, round(UNCOMPRESSED_SIZE/1024/1024) as "Uncompressed MB",
round(MEMORY_SIZE_IN_MAIN/1024/1024) as "Main MB", round(MEMORY_SIZE_IN_DELTA/1024/1024) as "Delta MB",
```

```
round(MEMORY_SIZE_IN_TOTAL/1024/1024) as "Total Used MB", round(COMPRESSION_RATIO_IN_PERCENTAGE/100, 2)
as "Compr. Ratio" from M_CS_COLUMNS where TABLE_NAME = 'LineItem'
```

In fact, the M_CS_TABLES and M_CS_COLUMNS system views contain much more information (such as cardinality, main-storage vs. delta storage and more). Another useful system view is M_CS_ALL_COLUMNS, which also exposes the internal, hidden, columns (such as index columns). See the “SAP HANA SQL and System Views Reference” for more information.

Row Tables

Some system tables are in fact row store tables. To get a sense of the total amount of memory used for these row tables, you can use the following query:

```
select round(sum(USED_FIXED_PART_SIZE + USED_VARIABLE_PART_SIZE)/1024/1024) as "Row Tables MB Used"
from M_RS_TABLES
```

To examine the memory consumption of row tables of a particular schema, for instance the schema ‘SYS’, drill down as follows:

```
select SCHEMA_NAME, TABLE_NAME, round((USED_FIXED_PART_SIZE + USED_VARIABLE_PART_SIZE)/1024/1024, 2) as
"MB Used" from M_RS_TABLES where SCHEMA_NAME = 'SYS' order by "MB Used" desc, TABLE_NAME
```

5 Statement Memory

An additional source of memory usage is the memory used by the queries and statements executed by the HANA database (e.g. for statement plan evaluation, caching, intermediate and final results calculation). While many statement executions use only a moderate amount of memory, some queries, for instance using unfiltered cross-joins, will tax even very large systems.

Expensive Statements

From SAP HANA 1.0 revision 80 (SPS8), you can track and monitor the amount of memory used by statements, via the "Expensive Statements" feature of the SAP HANA database.

The expensive statements trace records statements that exceed a given threshold in runtime, cpu time or memory consumption. If per-statement memory tracking is enabled additionally, the expensive statements trace will show the peak memory size (MEMORY_SIZE) used to process a statement.

To enable per-statement memory tracking[3], add the following section to the global.ini configuration file:

```
[resource_tracking]
enable_tracking = on
memory_tracking = on
```

Using the SQL console, you can do this as follows:

```
alter system alter configuration ('global.ini','SYSTEM') SET ('resource_tracking',
'enable_tracking')='on', ('resource_tracking','memory_tracking')='on' with reconfigure;
```

Starting with SAP HANA 1.0 revision 94 (SPS9), “Expensive Statements” can be triggered by memory threshold. All statements exceeding the configured memory threshold will be recorded in M_EXPENSIVE_STATEMENTS.

To enable “Expensive” Statements” by memory threshold, add the following section to global.ini:

```
[expensive_statement]
threshold_memory = <bytes>
```

Using the SQL console, you can do this as follows (please note that the value is in bytes! – example 50000000 bytes):

```
alter system alter configuration ('global.ini','SYSTEM') SET ('expensive_statement',
'threshold_memory')= 50000000 with reconfigure;
```

[3] At the time of writing statement memory tracking works correctly on single-node installations only.

Limiting expensive statements

In addition, starting with SAP HANA 1.0 SPS9 it is possible to protect a SAP HANA system against excessive memory usage by uncontrolled queries, by *limiting* the amount of memory used by single statement executions per host.

You accomplish this by adding the following section to global.ini:

```
[memorymanager]
statement_memory_limit = <GB>
```

Where <GB> is a numeric value, in Giga Bytes. Using SQL (example: 50 GBytes):

```
alter system alter configuration ('global.ini','SYSTEM') SET ('memorymanager',
'statement_memory_limit') = 50 with reconfigure;
```

Statement executions that would require more memory than this number will be aborted when they reach the limit. By default, there is no limit on statement memory.

To avoid canceling statements unnecessarily you can apply the “statement memory limit threshold” (available as of SAP HANA 1.0 SPS9). If a value for this parameter has been set, the statement memory limit is only respected if the current SAP HANA Used Memory exceeds the statement memory limit threshold.

To set the threshold add the following section to global.ini:

```
[memorymanager]
statement_memory_limit_threshold = <% of global_allocation_limit>
```

Using the SQL console, you can do this as follows (example: threshold set to 60% of global_allocation_limit)

```
alter system alter configuration ('global.ini','SYSTEM') SET ('memorymanager',
'statement_memory_limit_threshold') = 60 with reconfigure;
```

Limiting expensive statements for individual users

You can also set user-specific statement memory limits by using the ALTER USER statement:

```
alter user <user_name> set parameter statement memory limit =
<GB>
```

The value of the parameter is shown in USER_PARAMETERS.

If both a global and a user statement memory limit are set, the user-specific limit takes precedence, regardless of whether it is higher or lower than the global statement memory limit. If the user-specific statement memory limit is removed the global limit takes effect for the user.

Limiting statement memory using workload classes

As of SAP HANA 1.0 SPS10 it is possible to set a limit for statement memory usage per client, application, or application user.

You can do so by creating workload classes and workload class mappings. Note that the client needs to set metadata such as client or application user via the HANA session context (true for most SAP applications) for this mechanism to work.

If the parameter statement memory limit is not set for a specific workload class, by default the value defined for the statement_memory_limit in global.ini is used.

You can set values for one or more workload parameters in a single SQL statement. I.e. create a workload class called MyWorkloadClass which restricts the statement memory to 2GB:

```
Create workload class “MyWorkloadClass” set ‘STATEMENT MEMORY LIMIT’ = ‘2’
```

You can find further information in the SAP HANA Administration Guide and SAP Note 2331857.

6 Managing the Memory Pool

As mentioned, SAP HANA pre-allocates and manages its own memory pool, used for storing in-memory tables, for thread stacks, and for temporary results and other system data structures.

When more memory is required for table growth or temporary computations, the SAP HANA memory manager obtains it from the pool. When the pool cannot satisfy the request, the memory manager will increase the pool size by requesting more memory from the operating system, up to a predefined Allocation Limit.

By default, the allocation limit is set to 90% of the first 64 GB of physical memory on the host plus 97% of each further GB. You can see the allocation limit on the Overview tab of the Administration perspective of the SAP HANA studio, or view it with SQL:

```
select HOST, round(ALLOCATION_LIMIT/(1024*1024*1024), 2) as "Allocation Limit GB"
from PUBLIC.M_HOST_RESOURCE_UTILIZATION
```

There is normally no reason to change this value, except in the case when you purchased a license for less than the total of the physical memory. In such a case, you should set the global allocation limit accordingly, to remain in compliance with the license[4]. Another case when you may want to limit the size of the memory pool is on *development systems*, where more than one SAP HANA system may be installed on a single host, to avoid resource contentions or conflicts.

Memory is a finite resource. Once the allocation limit has been reached and the pool exhausted, the SAP HANA memory manager will no longer be able to allocate memory for internal operations without first giving up something else. In fact, this is exactly what will happen: buffers and caches will be released and column store tables will be unloaded, column by column, based on a least-recently-used order. When tables are partitioned over several hosts, this is managed per host; that is, column partitions will be unloaded only on hosts with acute memory shortage.

Table (column or partition) unloading is generally not a good situation, since it leads to performance degradation later, when the data will have to be reloaded later for queries that need them. You can identify pool exhaustion by examining the M_CS_UNLOADS system view. For instance, the following query will provide the number of unloads during a particular one-hour time-slot:

```
select count(*) from M_CS_UNLOADS
where UNLOAD_TIME between '19.08.2013 09:00:00' and '19.08.2013 10:00:00'
```

Despite all these abilities, it is theoretically possible that the memory manager will still face a need for more memory that it cannot satisfy. For instance, when too many concurrent transactions use up all memory, or when a particularly complex query performs a cross-join on very large tables, creating a huge intermediate result that exceed the available memory. Such situations can even lead to an Out Of Memory failure.

7 Multitenant Database Containers

SAP HANA supports multiple isolated databases in a single SAP HANA system. These are referred to as multitenant database containers.

A multiple-container system always has exactly one system database and zero to any number of multitenant database containers, called tenant databases. An SAP HANA system installed in multiple-container mode is identified by a single system ID (SID). Databases are identified by a SID and a database name. From the administration perspective, there is a distinction between tasks performed at system level and those performed at database level.

Memory Management in Multiple-Container Systems

You can manage and control the memory usage of your multiple-container system by configuring limits for individual tenant databases.

[4] Please also refer to SAP Note 1704499, for more information on memory usage auditing related to SAP HANA licenses.

You accomplish this by configuring the following parameter in global.ini:

```
[memorymanager]
allocationlimit = <MB>
```

Where <MB> is a numeric value, in Mega Bytes. Using SQL (example: 64 GBytes for tenant databes MYDB)

```
alter system alter configuration ('global.ini','DATABASE','MYDB' ) SET ('memorymanager',
allocationlimit') = 65536 with reconfigure;
```

This will limit the maximum amount of memory that can be allocated individually to processes of a tenant database. Each process of a tenant database can allocate the specified amount of memory.

If necessary, you can also reserve memory for the system database.

The system database contains only data required to monitor and manage the system, as well as statistics data related to itself. Memory consumption of the system database is average about 15 GB.

However, if the system database is experiencing out-of-memory situations, you can reserve a minimum amount of memory (MB) for the system database by configuring the following parameter in global.ini:

```
[memorymanager]
systemdb_reserved_memory = <MB>
```

System-level Monitoring in Multiple-Container Systems

For tenant-level monitoring in a multiple container system, the information provided in chapter 3 can be used. For system-level, monitoring, additional views are accessible in the system database using the SYS_DATABASES schema.

To display the current size of the Used Memory of all tenant databases (ordered by size), you can use the following simple SQL statement.

```
select HOST, DATABASE_NAME, round(TOTAL_MEMORY_USED_SIZE/(1024*1024*1024), 2) as "Used Memory GB"
from SYS_DATABASES.M_SERVICE_MEMORY where SERVICE_NAME = 'indexserver' order by HOST,
TOTAL_MEMORY_USED_SIZE desc
```

Details on Used Memory per main component and tenant database can be queried using the following SQL statement:

```
select * from (select "Host","Database","Component","Used Memory Size MB"   from (select "Host",
"Database", "Component", round (sum("Used Memory Exclusive")/(1024*1024),2) as "Used Memory Size MB"
from ( select t1.host "Host", t1.database_name "Database", component "Component", exclusive_size_in_use
"Used Memory Exclusive" from sys_databases.m_heap_memory t1 join sys_databases.m_service_memory t2 on
t1.host=t2.host and t1.port=t2.port and category != '/' and t1.component != 'Row Store Tables' where
t1.database_name != '' and t1.database_name != 'SYSTEMDB' union ( select t1.host "Host",
t1.database_name "Database", 'Row Store Tables' as "Component", allocated_size "Used Memory Exclusive"
from sys_databases.m_rs_memory t1 join sys_databases.m_service_memory t2 on t1.host = t2.host and
t1.port = t2.port and t1.database_name = t2.database_name where t1.database_name != '' and
t1.database_name != 'SYSTEMDB' )) group by "Host", "Database","Component")) order by "Host",
"Database","Used Memory Size MB" desc;
```

8 The Operating Environment

From the Linux operating system perspective, SAP HANA is a collection of separate processes. Linux processes reserve memory for their use by requesting an allocation from the Linux operating system. The entire reserved memory footprint of a program is referred to as its Virtual Memory. Each Linux process has its own virtual memory, which grows when the process requests more memory from the operating system, and shrinks when the process relinquishes memory. You can think of virtual memory size as the maximal amount that the process has been allocated, including reservations for its code, stack, data, and memory pools under program control. SAP HANA's virtual memory is logically[5] shown below:

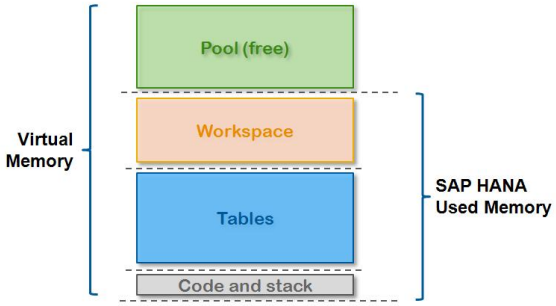

Figure 3 – SAP HANA Virtual Memory

Virtual, Physical and Resident Memory

When (part of) the virtually allocated memory actually needs to be *used*, it is loaded or mapped to the real, physical memory of the host, and becomes "resident". Physical memory is the DRAM memory installed on the host. On most SAP HANA hosts, it ranges from 256 Gigabytes (GB) to multiple Terabytes (TB). It is used to run the Linux operating system, SAP HANA, and all other programs.

Resident memory is the physical memory actually in operational use by a process.

Over time, the operating system may "swap out" some of a process' resident memory, according to a least-recently-used algorithm, to make room for other code or data. Thus, a process' resident memory size may fluctuate independently of its virtual memory size. In a properly sized SAP HANA appliance there is enough physical memory, and thus swapping is disabled and should not be observed.

See the following illustration:

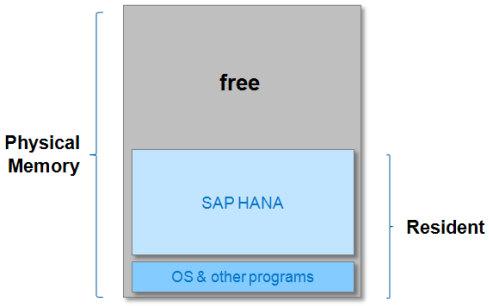

Figure 4 – Physical Memory

On a typical SAP HANA appliance, the resident part of the OS and all other running programs usually does not exceed 2GB. The rest of the memory is thus dedicated for the use of SAP HANA.

To display the size of the Physical Memory and Resident part, you can use the following SQL command:

[5] SAP HANA really consists of several separate processes, so this figure shows the combination of all SAP HANA processes

```
select HOST, round((USED_PHYSICAL_MEMORY + FREE_PHYSICAL_MEMORY)/(1024*1024*1024), 2) as
"Physical Memory GB", round(USED_PHYSICAL_MEMORY/(1024*1024*1024), 2) as "Resident GB"
from PUBLIC.M_HOST_RESOURCE_UTILIZATION
```

When memory is required for table growth or for temporary computations, the SAP HANA code obtains it from the existing memory pool. When the pool cannot satisfy the request, the HANA memory manager will request and reserve more memory from the operating system. At this point, the virtual memory size of the HANA processes grows.

Once a temporary computation completes or a table is dropped, the freed memory is returned to the memory manager, who recycles it to its pool, usually without informing Linux[6]. Thus, from SAP HANA's perspective, the amount of *Used Memory* shrinks, but the process' virtual and resident sizes are not affected. This creates a situation where the Used Memory may even shrink to *below* the size of SAP HANA's resident memory, which is perfectly normal.

The following illustration shows the relationship between physical memory, Linux virtual and resident memory, and SAP HANA's pool and Used Memory indicators. Note how changes in Used Memory (due to query execution and other internal operations) do not affect the processes' virtual and resident sizes.

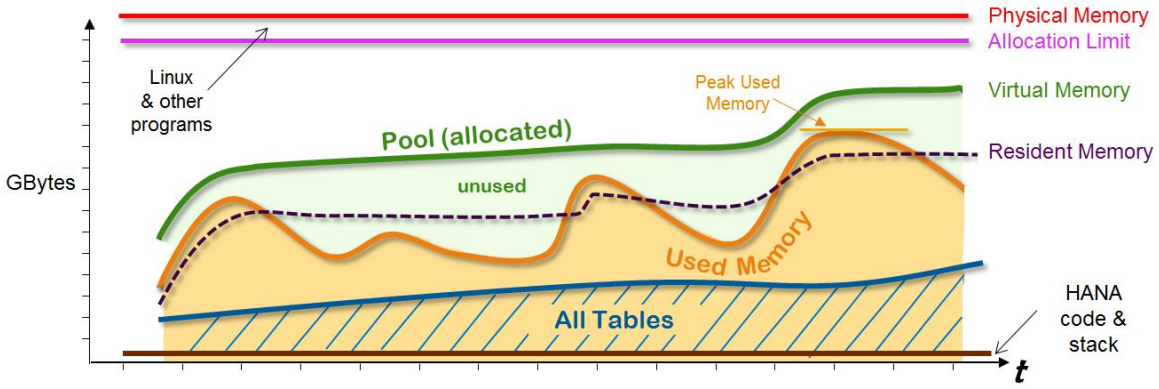

Figure 5 –Linux and SAP HANA Memory Relationship

Linux Indicators

To view the amount of physical memory on the host, and the resident part, use:

```
free –g | awk '/Mem:/ {print "Physical Memory: " $2 " GB."} /cache:/ {print "Resident: " $3 " GB."}'
```

This merely provides the host-perspective. The Linux process indicators are of course more relevant. Since the operating system treats SAP HANA as a collection of processes, we need to consider both the virtual and resident part of these processes. A process' virtual memory size is always larger than its resident size. Note also, that due to the managed memory pool, both the SAP HANA virtual size and resident sizes may appear larger than what the Used Memory indicator would lead you to expect. This is entirely normal.

Linux maintains High Water Mark (peak) indicators for the virtual and resident process sizes. In a stable system, the current virtual and resident sizes will be only slightly smaller than their respective high water marks because SAP HANA grows its pool, but does not normally relinquish unused memory. Thus, the resident size high water mark should generally track the peak Used Memory. Very large differences may indicate that parts of the SAP HANA memory pool were freed, possibly due to insufficient physical memory.

Display these process memory indicators for SAP HANA as follows (*replace "qp4adm" in this example with the username of the appropriate SAP HANA administrator*):

[6] This is for efficiency, as returning/requesting memory from Unix is expensive. The memory manager may also choose to return memory back to the operating system, for instance when the pool is close to the allocation limit, and contains large unused parts. Users have no control over this.

```
cat `ps h -U qp4adm -o "/proc/%p/status" | tr -d ' '` | awk '/VmSize/ {v+=$2} /VmPeak/ {vp+=$2} /VmRSS/ {r+=$2;} /VmHWM/ {rp+=$2} END {printf("Virtual Size = %.2f GB (peak = %.2f), Resident size = %.2f GB (peak = %.2f)\n", v/1024/1024, vp/1024/1024, r/1024/1024, rp/1024/1024)}'
```

Process memory reports on Linux are difficult to interpret for various reasons[7]. One of the main reasons is the way processes can share or re-use memory allocated to another process. Linux normally allocates *separate* memory areas to different processes, of course. There are two main exceptions to this rule.

The first is program code. When several processes use the same code libraries, there will only be *one* copy of that code in physical memory, re-used by all the processes.

This partially explains why it is not simple to determine the "size" of SAP HANA, whose processes highly re-use the same code. If we look at an individual process, its memory footprint will include the size of its entire code area (about 6GB for the index-server process), but if we naively sum the sizes of two processes, we *over-count* the re-used code areas. From SAP HANA SPS6, the formula used to calculate SAP HANA Used Memory incorporates a more sophisticated calculation of reused code areas.

The second exception is a technique called "shared memory". Shared memory, as its name implies, is used to "share" data between processes. Both processes define the same memory area as "shared", and they can then exchange information simply by writing into it. This (used to be, and still is somewhat) faster than the alternative of sending network or pipe-based messages between processes.

It is difficult to account for shared memory. Does it belong to one process? Both? Neither? If we naively sum the memory belonging to multiple processes, we grossly "over-count".

Linux reports shared memory inconsistently. When you ask it to report the resident memory map of a single process, it will report the shared-memory as part of its memory footprint. However, when you ask "how much *total memory* is resident", it does *not* account for shared memory.

This is usually not significant, because very few programs use large blocks of shared memory. However, SAP HANA is different. An early SAP HANA design decision was to use "shared memory" for row-store tables[8]. As a result, when you use large row-store tables, the shared-memory footprint of SAP HANA can become very large.

Thus, if you ask Linux to report the resident size of SAP HANA and the *total* resident size on the host, the size of SAP HANA may appear to exceed the total, which of course makes no sense.

To compensate, SAP HANA adds the resident size of the shared-memory part of the SAP HANA processes to the resident size reported by Linux[9]. This is another intentional reason why SAP HANA may report different values than Linux.

---

[7] A good reference for even more information is: “What Every Programmer Should Know About Memory”, http://www.akkadia.org/drepper/cpumemory.pdf

[8] Incidentally, this memory is not really shared between processes; it is just defined as "shared memory".

[9] This is not precise - it slightly over-counts really-shared memory and swapped-out shared memory - but is much more accurate than reporting the obviously wrong Linux value.

9 SAP HANA Administration Tools

SAP HANA Studio

You can view some of the most important memory indicators on the Overview tab of the SAP HANA studio administrative perspective:

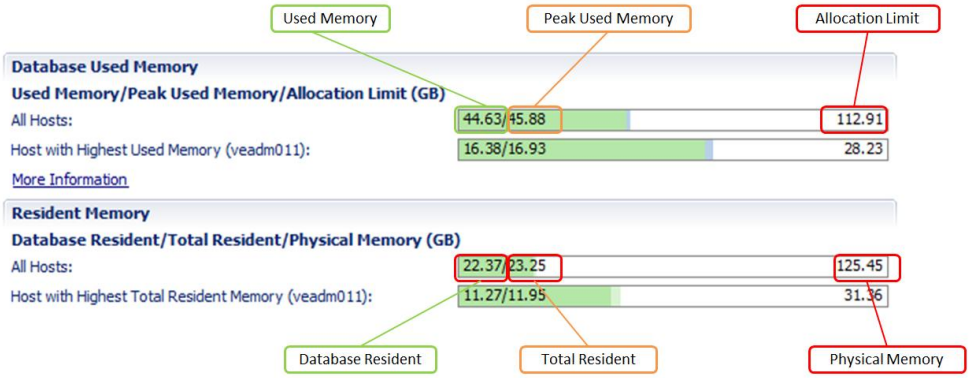

Figure 6 – SAP HANA Studio – Memory Summary

SAP HANA Cockpit

The SAP HANA cockpit is an SAP Fiori Launchpad site that provides you with a single point-of-access to a range of Web-based applications for the online administration of SAP HANA. It is accessible through a Web browser at https://<host>:43<instance>/sap/hana/admin/cockpit (recommended) or http://<host>:80<instance>/sap/hana/admin/cockpit

This will open the SAP HANA Cockpit[10], which looks as follows:

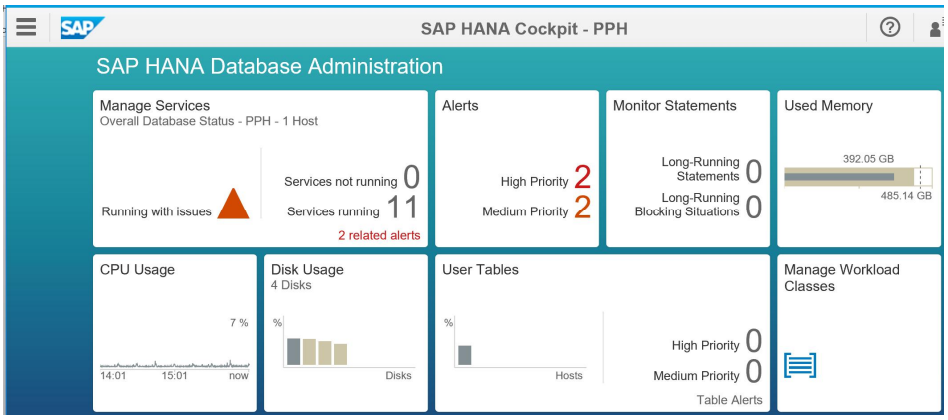

Figure 7 – SAP HANA Cockpit Launchpad

Selecting the Used Memory tile will open the performance monitor app displaying host and service specific memory KPIs like used memory, database used memory, allocation limit, etc. and a load graph. The load graph initially visualizes memory usage on the master host and master index server. You can customize it to your needs.

[10] To open the SAP HANA Cockpit you need Monitoring and Administrator privileges. For a detailed description of the prerequisites see the SAP HANA Administration Guide

Selecting the Manage Services app will display the running services.

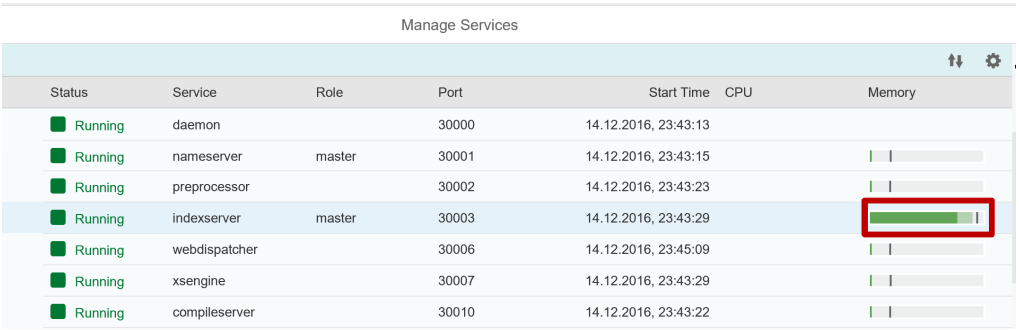

Figure 8 - SAP HANA Cockpt - Manage Services App

You can drill down into the Memory Allocation Statistics by clicking on the memory information of the respective service, i.e. of the indexserver as the main service.

You will see a screen that looks like this:

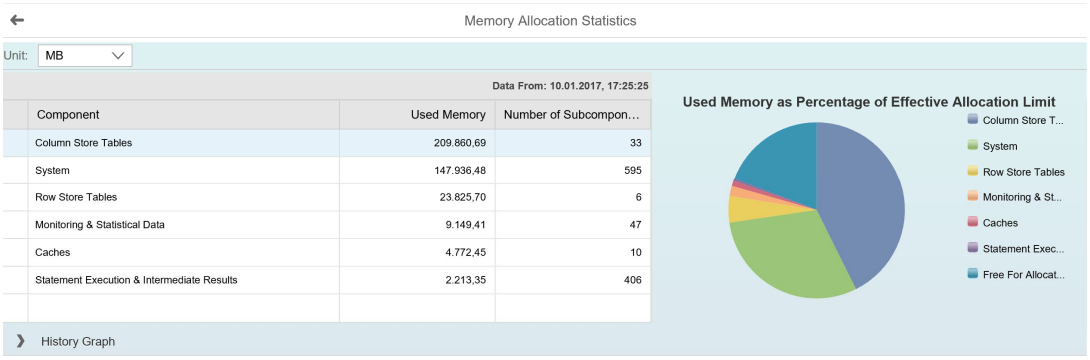

Figure 9 – Memory Allocation Statistics

The information provided is interactive; you can drill down into the main components, and understand how memory is consumed by SAP HANA's different sub-systems.

10 In Summary

SAP HANA maintains many system views and memory indicators, to provide a precise way to monitor and understand the SAP HANA memory utilization. The most important of these indicators is Used Memory and the corresponding historic snapshots. In turn, it is possible to drill down into very detailed reports of memory utilization using additional system views, or by using the SAP HANA Cockpit or Statistics from the SAP HANA studio.

Since SAP HANA contains its own memory manager and memory pool, external indicators, like the host-level Resident Memory size, or the process-level virtual and resident memory sizes, can be misleading when estimating the real memory requirements of a SAP HANA deployment.

Related information: SAP Knowledgebase Article 199997 provides a very comprehensive, up-to-date FAQ on SAP HANA MEMORY.