# 1999997 - FAQ: SAP HANA Memory

| | | | |
|---|---|---|---|
| **Version** | 608 | **Type** | SAP Knowledge Base Article |
| **Language** | English | **Master Language** | English |
| **Release Status** | Released to Customer | **Category** | How To |
| **Component** | HAN-DB (SAP HANA Database) | **Released On** | 27.07.2018 |

Please find the original document at **https://launchpad.support.sap.com/#/notes/1999997**

## Symptom

You have questions related to the SAP HANA memory.

You experience a high memory utilization or out of memory dumps.

## Environment

SAP HANA

## Cause



## Resolution

## 1. Which indications exist for SAP HANA memory problems?

Tracefiles with the following naming convention are created:

<service>_<host>.<port>.rtedump.<timestamp>.oom.trc
<service>_<host>.<port>.rtedump.<timestamp>.oom_memory_release.trc
<service>_<host>.<port>.rtedump.<timestamp>.compositelimit_oom.trc
<service>_<host>.<port>.rtedump.<timestamp>.after_oom_cleanup.trc
<service>_<host>.<port>.emergencydump.<timestamp>.trc (if memory related errors like "allocation failed" are responsible)

The following error messages can indicate OOM situations. Be aware that some of the errors can also be issued in other scenarios. To make sure that they are really memory related, you have to check the related trace file.

```
SQL error -9300: no more memory
SQL error -10760: memory allocation failed
SQL error -10108: Session has been reconnected
SQL error 129 while accessing table <table_name> transaction rolled back by an internal error:
Memory allocation failed transaction rolled back by an internal error: exception during deltalog
replay. transaction rolled back by an internal error: TableUpdate failed transaction rolled back
by an internal error: exception 1000002: Allocation failed ; $size$=1191936; $name$=TableUpdate;
$type$=pool; $inuse_count$=2180; $allocated_size$=8180736; $alignment$=16# transaction rolled
back by an internal error: TrexUpdate failed on table <table_name> with error:
commitOptimizeAttributes() failed with rc=6900, Attribute engine
failed;object=<object_name>$delta_1$en, rc=6900 - enforce TX rollback transaction rolled back by
an internal error: TrexUpdate failed on table '<table_name>' with error: Attribute load
failed;index=<table_name>en,attribute='$trexexternalkey$' (207), rc=6923 - enforce TX rollback
transaction rolled back by an internal error: TrexUpdate failed on table '<table_name>' with
error: AttributeEngine: not enough memory, rc=6952 - enforce TX rollback
SQL error 403 while accessing table <table_name>
internal error: "<schema_name>"."<table_name>": [133] (range 2)
SQL error 1024 while accessing table <table_name>
SQL message: Allocation failed $REASON$
SQL error 2048 while accessing table <table_name>
column store error: search table error: [2] message not found column store error: search table
error: [9] Memory allocation failed column store error: search table error: [1999] general error
(no further information available) column store error: search table error: [2575] flatten
scenario failed; Allocation failed column store error: search table error: [6900] Attribute
engine failed column store error: search table error: [6923] Attribute load failed column store
error: search table error: [6952] Error during optimizer search column store error: search table
error: [6952] AttributeEngine: not enough memory column store error: [2450] error during merge
of delta index occurred column store error: [6924] Attribute save failed column store error:
merge delta index error: [6924] Attribute save failed
SQL error 3584 while accessing table distributed SQL error: [9] Memory allocation failed
distributed SQL error: [2617] executor: plan operation execution failed with an exception
SQL error 3587 at CON invalid protocol or service shutdown during distributed query execution:
```

```
[2613] executor: communication problem plan <plan> failed with rc 9: Error executing physical
plan: Memory allocation failed
persistence error: exception 70029020: ltt::exception caught while operating on
DISK_NCLOB:<id>:<id>
exception 1000002: Allocation failed ; $size$=<size>; $name$=Page; $type$=pool;
$inuse_count$=<count>; $allocated_size$=<size>

Error 423 has occurred on the current database connection "DEFAULT". Database error text: AFL
error: OmsTerminate: error=-28530, liveCache BAD_ALLOCATION in <routine> liveCache ERROR -
1028000
```

Delta merges (SAP Note 2057046) fail with the following errors:

```
2048 column store error: [2009] Memory allocation failed 2048 column store error: [2450] Error
during merge of delta index occurred 2048 column store error: [2484] not enough memory for table
optimization 2048 column store error: [6923] Attribute load failed 2048 column store error:
[6924] Attribute save failed 2048 column store error: [6952] AttributeEngine: not enough memory
```

Backups fail with errors like:

```
Allocation failed ; $size$=<size>; $name$=ChannelUtils::copy; $type$=pool; $inuse_count$=0;
$allocated_size$=0
```

The following entries in the SAP HANA database trace files (SAP Notes 2380176, 2467292) exist:

```
mergeDeltaIndex failed for <schema>:<table> (<id>) rc=245
memAllocSystemPages failed with rc=12, 12 (Cannot allocate memory)
```

A consistency check with CHECK_TABLE_CONSISTENCY (SAP Note 2116157) fails with:

```
5088: The check for some table failed due to memory allocation failure
(ERROR_MEMORY_ALLOCATION_FAILED)
```

The following thread states and locks indicate issues in the memory area (SAP Note 1999998):

| Thread state | Lock name |
|---|---|
| Mutex Wait | HugeAlignmentPool |
| Mutex Wait | LimitOOMReport |
| Mutex Wait | PoolAllocator-MemoryPool |
| Semaphore Wait | IpmmTaskWait |
| Semaphore Wait | MemoryReclaim |

The following SAP HANA alerts indicate problems in the memory area:

| Alert | Name | Description |
|---|---|---|
| 1 | Host physical memory usage | Determines what percentage of total physical memory available on the host is used. All processes consuming memory are considered, including non-SAP HANA processes. |
| 12 | Memory usage of name server | Determines what percentage of allocated shared memory is being used by the name server on a host. |
| 40 | Total memory usage of column store tables | Determines what percentage of the effective allocation limit is being consumed by individual column-store tables as a whole (that is, the cumulative size of all of a table's columns and internal structures). |
| 43 | Memory usage of services | Determines what percentage of its effective allocation limit a service is using. |

| | | |
|---|---|---|
| 44 | Licensed memory usage | Determines what percentage of licensed memory is used. |
| 45 | Memory usage of main storage of column store tables | Determines what percentage of the effective allocation limit is being consumed by the main storage of individual column-store tables. |
| 55 | Columnstore unloads | Determines how many columns in columnstore tables have been unloaded from memory. This can indicate performance issues. |
| 64 | Total memory usage of table-based audit log | Determines what percentage of the effective allocation limit is being consumed by the database table used for table-based audit logging. |
| 68 | Total memory usage of row store | Determines the current memory size of a row store used by a service. |

*SQL: "HANA_Configuration_MiniChecks"* (SAP Notes 1969700, 1999993) returns a potentially critical issue (C = 'X') for one of the following individual checks:

| Check ID | Details |
|---|---|
| M0230 | Current memory utilization (%) |
| M0231 | Time since memory utilization > 95 % (h) |
| M0240 | Current swap utilization (GB) |
| M0241 | Time since swap utilization > 1 GB (h) |
| M0245 | Swap space size (GB) |
| M0410 | Current allocation limit used (%) |
| M0411 | Current allocation limit used by tables (%) |
| M0413 | Time since allocation limit used > 80 % (h) |
| M0415 | Curr. max. service allocation limit used (%) |
| M0417 | Time since service alloc. limit used > 80 % (h) |
| M0420 | Heap areas currently larger than 50 GB |
| M0421 | Heap areas larger than 100 GB (last day) |
| M0422 | Heap areas larger than 200 GB (history) |
| M0423 | Heap areas with potential memory leak |
| M0425 | Pool/RowEngine/CpbTree leak size (GB) |
| M0426 | Row store table leak size (GB) |
| M0430 | Number of column store unloads (last day) |
| M0431 | Time since last column store unload (days) |
| M0435 | Number of shrink column unloads (last day) |
| M0437 | Size of unloaded columns (GB, last day) |
| M0440 | Shared memory utilization of nameserver (%) |
| M0445 | Number of OOM events (last hour) |
| M0450 | Tables with memory LOBs > 2 GB |
| M0453 | Size of non-unique concat attributes (GB) |
| M0454 | Size of non-unique concat attributes (%) |
| M0455 | Unused large non-unique concat attributes |

| | |
|---|---|
| M0460 | Calc engine cache utilization (%) |
| M0470 | Heap allocators with many instantiations |
| M0472 | Booked vs. allocated memory (%) |
| M0480 | Address space utilization (%) |
| M0530 | Shared memory row store size (GB) |
| M0645 | Number of OOM tracefiles (last day) |
| M0746 | Histories with primary key |
| M0747 | Number of zero entries in HOST_SQL_PLAN_CACHE |
| M0748 | History of M_CS_UNLOADS collected |

*SQL: "HANA_TraceFiles_MiniChecks"* (SAP Note 2380176) reports one of the following check IDs:

| Check ID | Details |
|---|---|
| T0300 | Memory allocation failed |
| T0302 | Out of memory (OOM) |
| T0304 | Out of memory (OOM) exception |
| T0306 | Operating system cannot allocate memory |
| T0308 | Statement memory limit reached |
| T0310 | Resource container shrink |
| T0312 | Leaking allocator destroyed |
| T0320 | Dubious NUMA configuration |

*SQL: "HANA_Tables_ColumnStore_UnloadsAndLoads" (UNLOAD_REASON = 'LOW MEMORY')* (SAP Note 1969700) shows significant amounts of column unloads for the considered time frame.

### 2. How can I collect information about the current SAP HANA memory consumption?

SAP Note 1969700 provides the following SQL statements to collect information related to the current SAP HANA memory allocation:

| SQL statement name | Description |
|---|---|
| SQL: "HANA_Memory_Caches" | Overview of existing SAP HANA caches (SAP Note 2502256) |
| SQL: "HANA_Memory_ContextMemory" | Current context memory utilization, useful to map used memory to connections |
| SQL: "HANA_Memory_MemoryObjects" | Current memory objects in SAP HANA resource container (heap + row store); only used for areas taking advantage of caching, so temporary SQL areas like Pool/itab aren't part of it. |
| SQL: "HANA_Memory_Overview" | Provides information about current memory allocation (including heap, row store, column store, allocation limit and license limit) |
| SQL: "HANA_Memory_TopConsumers" | Lists the current top memory consumers (e.g. tables and heap areas) |

SAP Note 1698281 provides a Python script that can be used to collect detailed SAP HANA memory requirements. In order to get precise data, columns are actually loaded into memory rather than only relying on estimations.

If you don't have SQL access (e.g. on the secondary site of a SAP HANA system replication environment), you can use the operating system tool hdbcons (SAP Note 2222218) and 'mm l -S' to display the allocators sorted by the inclusive memory size. Sorting by the more important exclusive size in use is not possible. Starting with SAP HANA SPS 11 you can also query this information via _SYS_SR_SITE_<site_name>. See SAP Note 1999880 ("Is it possible to monitor remote system replication sites on the primary system?") for details.

### 3. How can I collect information about the historic SAP HANA memory consumption?

SAP Note 1969700 provides the following SQL statements to collect information related to the historic SAP HANA memory allocation:

| SQL statement name | Description |
|---|---|
| SQL: "HANA_Memory_Reclaims" | Information about historic reclaim operations (i.e. defragmentations or shrinks) |
| SQL: "HANA_Memory_TopConsumersInHistory" | Lists historic top memory consumers (e.g. tables and heap areas) |
| SQL: "HANA_Memory_OutOfMemoryEvents" | Overview of out-of-memory (OOM) situations since last startup |
| SQL: "HANA_SQL_SQLCache" | Starting with SAP HANA Rev. 102.01 memory information is available in the SQL cache (if memory tracking is activated) that can be evaluated. |
| SQL: "HANA_SQL_ExpensiveStatements" | Lists memory consumption of executed SQL statements (SPS 08) Relevant output columns: MEM_USED_GB, MEM_PER_EXEC_GB Both expensive SQL statement trace and statement memory tracking needs to be activated, see "Is it possible to limit the memory that can be allocated by a single SQL statement?" in this SAP Note for more information. |

### 4. Which important memory areas exist?

The following memory areas are most important:

| Memory Area | Context | Level | Details |
|---|---|---|---|
| Physical memory | operating system | global | Total amount of memory physically available on host level (typically RAM) |
| Virtual memory | operating system | process | Total amount of memory allocated by all processes held both in physical memory and in paging area on disk |
| Resident memory | operating system | process | Total amount of memory allocated by all processes held in physical memory, large allocations are usually fine (SAP Note 2081473) |
| Allocated memory | SAP HANA | process | Total amount of memory allocated by the SAP HANA processes, limited by the configurable SAP HANA global allocation limit Less relevant for SAP HANA memory analysis because allocated, but unused memory can be re-used when required |
| Used memory | SAP HANA | process | Total amount of memory in use by the SAP HANA processes, relevant to understand SAP HANA memory footprint |
| Shared | SAP HANA | global | Memory that can be accessed by different processes, e.g.: Specific row |

| | | | |
|---|---|---|---|
| memory | | | store components (tables, catalog, free) Nameserver topology |
| Heap memory | SAP HANA | process | Memory exclusively accessible by threads of a single process (e.g. indexserver), e.g.: Column store Row store indexes Intermediate results Temporary structures SAP HANA page cache |
| Code | SAP HANA | global | Code |
| Stack | SAP HANA | process | Stack |

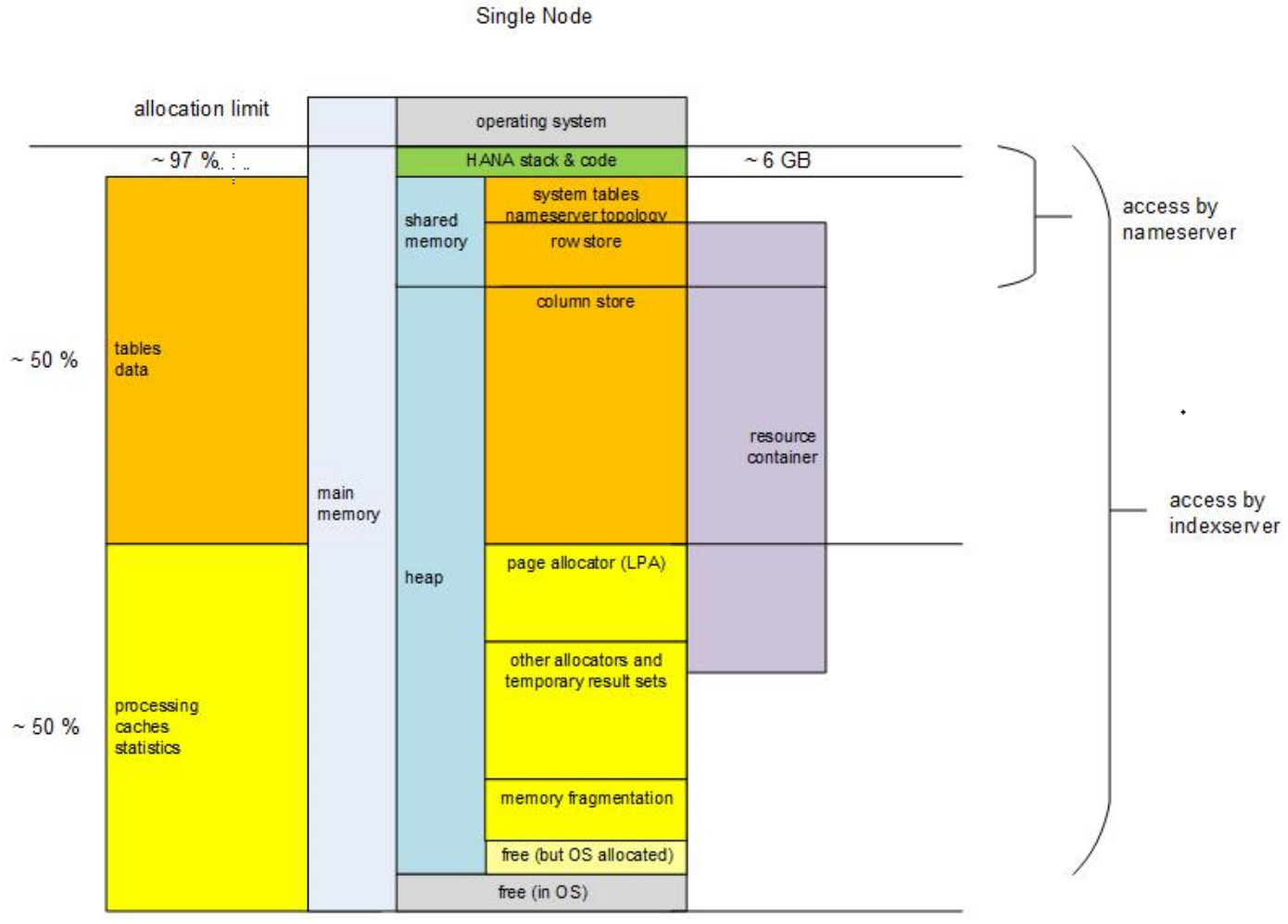
In normal SAP HANA environments no paging happens and SAP HANA is the only major memory allocator on the host. The following conditions are typically met:

- Physical memory > virtual memory
- Virtual memory = resident memory >= allocated memory
- Allocated memory = shared memory + allocated heap memory
- Used memory = shared memory + used heap memory
- Code, stack: Usually negligible sizes

For efficiency reasons SAP HANA frees allocated memory in a "lazy" way and so the allocated memory can grow up to the available memory and the global allocation limit while the used memory remains at a much lower level.

From a memory analysis perspective we can usually focus on the used memory and assume that the allocated memory is released whenever required.

The following picture illustrates the general SAP HANA memory structure:



## 5. What does SAP HANA do if memory becomes scarce?

Unlike other databases (e.g. Oracle: PGA in memory -> PSAPTEMP on disk) SAP HANA doesn't allocate disk space if certain operations require more memory than available. Instead the following actions are taken:

| Action | hdbcons | Details |
|---|---|---|
| Reclaim - return free memory to OS | | Free memory segments are returned to operating system. This is helpful in cases where a SAP HANA memory request is larger than the individually available segments in the SAP HANA heap memory. The operating system is able to perform a defragmentation and provide larger segments afterwards. This operation isn't recorded in database trace (SAP Note 2380176). You can determine reclaim activities using the "mm ipmm -d" option of hdbcons (SAP Note 2222218). The following information indicates reclaim at the specified point in time: T=2016-12-08 04:42:43.722 ...: Process <pid> requested self compaction A self compaction is a reclaim that is triggered by the process itself. |
| Reclaim - defragmentation | mm gc -f | Garbage collection is triggered so that allocated memory is defragmented and freed for re-use. It is executed automatically when not sufficient free memory is available. For a manual / explicit control of memory garbage collection see question "What is memory memory garbage collection?" below. See SAP Note SAP Note 2169283 for more information related to SAP HANA garbage collection. Releasing the memory back to the operating system requires the IPMM lock, so memory allocations can be blocked (e.g. with "ReclaimMemory" locks, see SAP Note 1999998). This operation isn't recorded in database trace (SAP Note 2380176). You can determine reclaim activities using the "mm ipmm -d" option of hdbcons (SAP Note 2222218). The following information indicates reclaims at the specified point in time: T=2016-12-08 04:42:43.722 ...: Process <pid> requested self compaction |
| Reclaim - resource container shrink | resmans | Resource container is shrunk: Non-critical heap areas are reduced (e.g. the SAP HANA page cache Pool/PersistenceManager/PersistentSpace(0)/DefaultLPA/Page or compiled L code) Column store unloads are triggered (SAP Note 2127458); this activity can significantly impact the performance. Automatic shrinks are recorded in the database trace (SAP Note 2380176) and can be found by checking for the following information: Information about shrink |
| Termination of transactions | | Transactions are terminated with error if their memory requests can no longer be fulfilled. |
| OOM dump | | An out-of-memory (OOM) dump is written (if the time defined in parameter global.ini -> [memorymanager] -> oom_dump_time_delta is exceeded since the last OOM dump) |

Memory garbage collection and shrinks are locally done by the thread that detects the need for these tasks. In the following cases a specific MemoryCompcator thread is used for that purpose (SAP Note 2114710):

- Memory garbage collection and shrink requests coming from another SAP HANA service
- Memory garbage collection triggered by explicitly configured parameters global.ini -> [memorymanager] -> gc_unused_memory_threshold_abs and global.ini -> [memorymanager] -> gc_unused_memory_threshold_rel.

*SQL: "HANA_Memory_Reclaims"* (SAP Note 1969700) can be used to display reclaim runtimes and processed memory sizes.

## 6. Which parameters can be used to limit the SAP HANA memory consumption?

The following parameters can be used to limit the overall or process-specific SAP HANA memory allocation:

| Parameter | Unit | Details |
|---|---|---|
| global.ini -> [memorymanager] -> global_allocation_limit | MB | This parameter limits the overall memory consumption of SAP HANA. The default value depends on the available physical memory and the SAP HANA revision level: SPS 06 and below: 90 % of physical memory SPS 07 and higher: 90 % of first 64 GB, 97 % of remaining physical memory |
| <service>.ini -> [memorymanager] -> allocationlimit | MB % | This parameter limits the memory consumption of the related SAP HANA process (<service>). If "%" is specified at the end of the parameter value (without preceeding blank), the value is interpreted as percentage of RAM, otherwise it is interpreted as MB. The standalone statistics server uses a value of "5%" per default. All other services including indexserver use the following allocation limit per default: Rev. <= 92: 90 % of physical memory Rev. >= 93: global_allocation_limit As an example, SAP Note 1862506 suggests an increase of the allocation limit of the standalone statistics server to "10%", "15%" or "20%" in order to come around OOM situations caused by the default 5 % limit. |

Normally there is no need to touch these settings and there are other solutions to come around memory issues.

## 7. How can I analyze problems related to the SAP HANA memory consumption?

SAP Note 1840954 describes steps to analyze and resolve SAP HANA memory issues.

SAP Note 1984422 describes how to analyze an out of memory (OOM) dump file.

SAP Note 2222718 provides a decision-tree approach for analyzing problems in the SAP HANA memory area.

The SAP HANA Troubleshooting and Peformance Analysis Guide at SAP HANA Troubleshooting and Performance Analysis Guide covers - among others - the analysis of memory related issues.

## 8. Is it possible to extend the physical memory of a SAP HANA machine?

In general the configured physical memory depends on factors like hardware, scenario and available CPUs and must not be changed. SAP Note 1903576 describes when and how you can apply for an exception.

## 9. Which options exist to reduce the risk of SAP HANA memory issues?

The following options exist to reduce the risk of SAP HANA memory issues:

| Action / Feature | Details |
|---|---|
| Cleanup of technical tables | Make sure that house-keeping is set up for technical, administration and communication tables so that they don't consume unnecessary memory. See SAP Note 2388483 for more information. |
| Archiving | Implement archiving strategies for business data. Have a look at the Information Lifecycle Management area for more details. |

| | |
|---|---|
| S/4HANA | S/4HANA significantly reduces redundancy of table data (e.g. FI: elimination of BSEG index tables like BSIS, BSID, BSAS and BSAD) and so it has a positive impact on the memory footprint. See the Simplification List for S/4HANA for further details. |
| Hybrid LOBs | Hybrid LOBs are not loaded into memory when the size exceeds a defined limit, so it is usually beneficial for memory consumption if you take advantage of this feature. SAP Note 1994962 describes how columns defined as memory LOBs can be converted to hybrid LOBs. SAP ABAP table columns with LRAW data type are mapped to either LOB or VARBINARY. As VARBINARY always has to be loaded into memory, this can have an effect on the memory utilization. See SAP Note 2220627 ("Is VARBINARY also a LOB Type?") for more information. SAP Note 2375917 describes how a VARBINARY column can be converted into a LOB in order to save memory. |
| Reduction of number of indexes | Check for indexes with high memory requirements (e.g. using SQL: "HANA_Indexes_Overview", ORDER_BY = 'SIZE' from SAP Note 1969700) and check if you can drop some of these indexes. A focus can be put in the following areas: Secondary indexes that were created in order to optimize the performance of non-HANA databases. BW: If DSOs are changed from "standard" to "write-optimized", a primary index is no longer required. BW: Check if you can flag the property “Allow duplicate records” of write-optimized DSOs because this will eliminate the need for multicolumn key indexes (/BIC/A…00KE). Check if large fulltext indexes are really required. For example, a large index REPOSRC~SRC (on a column with name $_SYS_SHADOW_DATA) may exist to support the ABAP Sourcecode Search (SAP Note 1918229) and can be removed via transaction SFW5. Dropping indexes can significantly impact performance, so you should test the effects carefully before permanently dropping indexes. |
| Transition from multi-column to single-column indexes | Multi-column indexes require much more memory than single-column indexes, because an additional internal column (concat attribute) needs to be created. Check for indexes with high memory requirements (e.g. using SQL: "HANA_Indexes_Overview", ORDER_BY = 'SIZE' from SAP Note 1969700) and check if you can redefine some multi-column indexes to single-column indexes. Often it is a good compromise to define an index only on the most selective column. Further columns like MANDT would significantly increase the memory requirements. |
| Reduction of concat attributes | Concat attributes are specific internal columns that can be created for various reasons. Some of them may no longer be required. See SAP Note 1986747 for more information. You can run SQL: "HANA_Indexes_ColumnStore_RedundantConcatAttributes" (SAP Note 1969700) in order to define redundant concat attributes - i.e. multiple concat attributes created on the identical set of columns (with the identical order) and use the generated DROP_COMMAND to drop one of these duplicates. A typical reason for this behavior for SID tables in BW environments is described in SAP Note 2376550 |
| Paged attributes | Paged attributes are columns that can be loaded into the memory piece-wise. All columns apart from primary key and internal columns can be defined as paged attributes. For more details see SAP Note 1871386. |
| Inverted hash indexes | As of SAP HANA 1.0 SPS 09 you can reduce the size of multi-column indexes using the inverted hash feature. This can reduce the size of the internal concat attribute that is required for multi-column indexes. See SAP Note 2109355 for more information. |
| Inverted individual indexes | Starting with SAP HANA 2.0 SPS 03 primary keys and unique indexes can be defined as inverted individual indexes which eliminate the need to have a potentially large concat attribute and so the index size can be significantly reduced. See SAP Note 2600076 for more details. |
| Move large tables to column store | Table data is compressed efficiently in column store, so moving tables from row store to column store usually reduced the memory allocation significantly. Furthermore table columns are only loaded into the column store memory if required and not during startup. |

| | |
|---|---|
| | Therefore you can check if large tables exist in row store that can be moved to column store. Be aware that tables with a significant amount of modifications can suffer from performance regressions if they are moved to column store. In case of SAP standard tables you should usually double-check with SAP if the move to the column store is an option. |
| Analysis of large heap areas | Some heap areas may be larger than required, e.g. due to bugs or inadequate configuration. See question "What can I do if a certain heap allocator is unusually large?" below for more details. |
| SQL statement optimization | SQL statements processing large amounts of data or accessing data inefficiently can be responsible for a significant memory growth. See SAP Note 2000002 related to SQL statement optimization. See question "Which general optimizations exist for reducing the SQL statement memory requirements?" below for more information. |
| Transactional problems | Long running transactions or idle cursors can impact the garbage collection and result in a high amount of versions or histories. See SAP Note 2169283 for more information about symptoms, analysis steps and resolutions in the area of garbage collection. |
| Fragmentation (row store) | Fragmentation effects can result in an unnecessary row store size. See SAP Note 1813245 for more information on checking the row store fragmentation and reorganizing the row store. |
| Fragmentation (heap memory) | See "Can there be fragmentation in the heap memory?" in order to check if there is an unusual high fragmentation of the heap memory (> 15 %). Open a SAP incident in case you require assistance to understand and minimize the fragmentation. |
| Large delta storage | Many records in the delta storage of tables can increase the size of the column store. See SAP Note 2057046 and make sure that delta merges are running properly. |
| Delta merge and optimize compression | Delta merges (SAP Note 2057046) and optimize compression runs (SAP Note 2112604) temporary require a much larger memory footprint, typically you have to expect that the double size of the underlying table (partition) is needed. Therefore you have to make sure that the size of the table (partitions) is sufficiently small that doubling it is possible without running into a memory bottleneck. Typically you can achieve this by proper data management (see SAP Note 2388483) and by partitioning particularly large tables (SAP Note 2044468). |
| Column store compression | See SAP Note 2112604 and make sure that the column store tables are compressed optimally. |
| Unload configuration | It is possible to influence the unload behavior so that less critical objects are unloaded first ("UNLOAD PRIORITY <level>" setting for tables) . The following parameter controls the minimum size of the SAP HANA resource container that needs to be retained (SAP Note 1993128): indexserver.ini -> [memoryobjects] -> unload_lower_bound If this size has reached the defined limit and more memory outside of the resource container is required (e.g. because of an expensive SQL statement), an out-of-memory situation is issued. It is usually not required to configure this parameter because the statement memory limit has similar effects. |
| Data aging | Data aging (SAP Note 2416490) allows to load only current data into memory while older data is kept on disk. This feature is only available for a defined set of tables. |
| Dynamic tiering | Using dynamic tiering you can mark data as hot, warm and cold. Typically only hot data resides in the SAP HANA memory. See SAP Note 2140959 for more information related to dynamic tiering. |
| Smart data access | Based on smart data access SAP HANA can retrieve data from tables in external databases (e.g. Sybase, Oracle or SAP HANA). This reduced the need to load all accessed data into SAP HANA. See SAP Note 2180119 for more information regarding smart data access. |

| | |
|---|---|
| Extension nodes | Starting with SAP HANA 1.00 SPS 12 and 2.00 SPS 01 it is possible to configure extension nodes for tables containing no hot data. By overloading the extension node it is possible to share a limited amount of memory by a high amount of tables. See SAP Note 2415279 for more information. |
| Table distribution | If some hosts in a scale-out scenario suffer from a high memory consumption you can re-locate tables or table partitions from hosts with a high memory consumption to hosts with a lower memory consumption. See section "Table Distribution in SAP HANA" of the SAP HANA Administration Guide for more information. |
| Global allocation limit | The following parameter defines the maximum overall memory size which can be allocated by the SAP HANA instance: global.ini -> [memorymanager] -> global_allocation_limit The default value depends on the available physical memory and the SAP HANA revision level: SPS 06 and below: 90 % of physical memory SPS 07 and higher: 90 % of first 64 GB, 97 % of remaining physical memory Particularly on SPS 06 and below and hosts with a lot of memory this can result in a significant amount of unused memory (e.g. SPS 06, 1 TB memory, 90 % allocation limit, up to 900 GB allocated by SAP HANA, 10 GB allocated by OS and other components -> 90 GB unused). If you observe a significant amount of permanently unused memory you can increase the global_allocation_limit parameter (e.g. to "95%" or "97%" for SPS 06 and below). Make sure that you don't increase the allocation limit to a value that results in paging. If multiple SAP HANA instances run on the same host, you have to make sure that the sum of all configured global allocation limits doesn't exceed the available memory. |
| OS configuration | Make sure that the operating system configuration is in line with the SAP recommendations. See SAP Note 2000003 ("How can the configuration and performance of the SAP HANA hardware, firmware and operating system be checked?") for more information. It is particularly important that the ulimit package isn't installed in SLES environments, because it may define address space limitations (e.g. SOFTVIRTUALLIMIT < 100 % in /etc/sysconfig/ulimit). The following command should return nothing, otherwise it has to be uninstalled: rpm -qa \| grep ulimit Make sure that no address space limitations are defined for the SAP HANA processes. You can use the following commands to determine the process ID of the indexserver via ps (<indexserver_pid>) and subsequently check for the configured address space limitations: ps -ef \| grep indexserver egrep 'Soft\|space' /proc/<indexserver_pid>/limits The correct output without limitation looks similar like the following example: Limit Soft Limit Hard Limit Units Max address space unlimited unlimited bytes See also SAP Note 1980196 that discusses OOM errors due to an inadequate setting of the Linux parameter /proc/sys/vm/max_map_count. If multiple SAP HANA instances (or other applications with high memory requirements) run on the same node, make sure that the overall assigned memory (e.g. the global allocation limits for the SAP HANA instances) doesn't exceed the available physical memory. See SAP Note 2123782 which suggests a pagepool size reduction from 16 GB to 4 GB in Lenovo / GPFS environments. Make sure that the limit for stack is not set to a high / unlimited value (SAP Note 2488924) as it can result in a significant address space consumption. |
| Strict NUMA memory binding | If the operating system issues on OOM although there is sufficient memory available, an erroneous strict NUMA memory binding of SAP HANA processes can be responsible. See SAP Note 2358255 for details. This issue is fixed with Rev. 122.02. See SAP Note 2470289 for more information related to NUMA in SAP HANA environments. |
| SAP HANA patch level | The memory allocation of certain heap areas is SAP HANA patch level dependent. Newer revision levels may include optimizations that reduce the memory allocation. Therefore it is generally useful to make sure that a reasonably new revision level is implemented. |
| Scale-out layout | Using fewer hosts with a larger amount of physical memory each will reduce the risk that specific SQL statements with a high memory requirement will result in OOM situations, because there is a larger amount of available memory on each host. So for example 2 hosts |

| | |
|---|---|
| | with 1 TB memory each would have a lower risk of OOM situations compared to 8 hosts with 256 GB each. |
| Statistics server optimizations | See SAP Note 2147247 (-> "How can the memory requirements of the statistics server be minimized?") for details. |
| BW DTP delta initialization request optimization | If you face a high memory consumption related to DTP activities in BW, you can check SAP Note 2230080 for possible optimizations. |
| Bypassing SAP HANA bugs | Make sure that you are on reasonably new SAP HANA Revision levels and avoid situations that can cause memory related issues due to SAP HANA bugs. Particularly consider the following scenarios: Impacted Revisions Details 1.00.90 - 1.00.97.03 1.00.100 - 1.00.102.00 When a column store table (partition) reaches the 2 billion record limit (SAP Note 2154870) a SAP HANA overflow bug can result in extremely high memory allocation requests like: Failed to allocate 2305843008945258496 byte. Failed to allocate 18446744073667608592 byte. As a consequence SAP HANA will run into an out-of-memory situation even if significant amounts of memory are still available. Therefore follow the general strong recommendations and take appropriate actions (e.g. data reduction or partitioning) to avoid that a table (partition) reaches the 2 billion record limit. various Check "What can I do if a certain heap allocator is unusually large?" in order to identify SAP HANA bugs that are responsible for memory leaks and other reasons of unnecessary high memory allocation. 1.00.110 - 1.00.112.05 1.00.120 - 1.00.122.01 If the row store size (shared memory) is significantly larger than the total size of row store tables, you should check if the SAP HANA bug described in SAP Note 2362759 applies (memory freed by delete operations is no longer re-used). <= 1.00.122.14 <= 2.00.012.03 2.00.020 A bug in shared memory accounting in MDC environments (SAP Note 2101244) can result in operating system related OOM situations that could have been prevented if SAP HANA had performed reclaims / shrinks. See SAP Note 2588395 for more information. <= 2.00.024.00 If the total memory size of a workload class is limited, unjustified OOMs can happen. See SAP Note 2629536 for more information. See also "Is the SAP HANA memory information always correct?" -> M_CONTEXT_MEMORY below for scenarios where a wrong implicit memory booking can result in unjustified OOM terminations. |
| Sizing review | If all above checks didn't help to reduce the OOM situations you should double-check the SAP HANA sizing. See SAP Note 2000003 ("What has to be considered for sizing SAP HANA?") for more information. |

## 10. How can I judge if the available memory is sufficient for the current system and a projected future growth?

There are some general rules of thumb available that can help to understand if the memory is properly sized in an existing system, e.g.:

- Memory size should optimally be at least two times the total size of row store and column store.
- The memory used by SAP HANA should be significantly below the SAP HANA allocation limit.

All these rules are only rough guidelines and there can always be exceptions. For example, lome large S/4HANA systems can work absolutely fine even if 65 % of the memory is populated with table data.

At this point we won't use these rules but instead describe a more detailed approach based on a real-life SAP Suite on HANA system with 4 TB of physical memory.

In a first step it is important to understand how much memory is allocated by the different main areas. This information is retrieved via *SQL: "HANA_Memory_TopConsumers" (AGGREGATE_BY = 'AREA')*:

```
-----------------------------------------------
|AREA |SIZE_GB |SIZE_PCT|CUM_SIZE_PCT|
-----------------------------------------------
|Column store| 1011.72| 60.55| 60.55|
|Heap area | 446.89| 26.74| 87.30|
|Row store | 128.77| 7.70| 95.01|
|Code | 6.62| 0.39| 95.41|
|Stack | 1.58| 0.09| 95.50|
-----------------------------------------------
```

We can see that around 1.1 TB are used by the column store, 0.1 TB is used by the row store and additional 0.4 TB are used by heap areas (that are not integral part of other areas). The total memory utilization of SAP HANA is significantly below 2 TB, so we can already conclude that there is a lot of safety margin for exceptional situations and future growth before the 4 TB memory limit is reached.

More detailed information can be determined with *SQL: "HANA_Memory_Overview"* (SAP Note 1969700). The output for the same system looks like:

```
------------------------------------------------------------------------------------------------------------
-----------------------------------------------------
|NAME |TOTAL_GB |DETAIL_GB |DETAIL2_GB |
------------------------------------------------------------------------------------------------------------
-----------------------------------------------------
|User-defined global allocation limit|not set | | |
| | | | |
|License memory limit | 4000| | |
| | | | |
|License usage | 3000| 1554 (2014/03/01-2014/03/31)| |
| | | 2873 (2014/04/01-2014/04/30)| |
| | | 2849 (2014/05/01-2014/05/31)| |
| | | 3000 (2014/06/01-2014/06/27)| |
| | | | |
|Physical memory | 4040| 4040 (hlahana21) | |
| | | | |
|HANA instance memory (allocated) | 3450| 3450 (hlahana21) | |
| | | | |
|HANA instance memory (used) | 1639| 1639 (hlahana21) | |
| | | | |
|HANA shared memory | 121| 121 (hlahana21) | |
| | | | |
|HANA heap memory (used) | 1508| 1508 (hlahana21) | 355 (Pool/NameIdMapping/RoDict) |
| | | | 192 (Pool/AttributeEngine-IndexVector-Sp-Indirect) |
| | | | 105 (Pool/AttributeEngine-IndexVector-Single) |
| | | | 102 (Pool/PersistenceManager/PersistentSpace(0)/DefaultLPA/Page)|
| | | | 85 (Pool/RowEngine/QueryExecution) |
| | | | 73 (Pool/AttributeEngine/idattribute) |
| | | | 66 (Pool/Statistics) |
| | | | 58 (Pool/AttributeEngine) |
| | | | 44 (Pool/AttributeEngine-IndexVector-SingleIndex) |
| | | | 38 (Pool/RowEngine/CpbTree) |
| | | | |
|Column store size | 1011| 1011 (hlahana21) | 315 (KONV) |
| | | | 84 (BSEG) |
| | | | 42 (ZARIXSD5) |
| | | | 36 (VBFA) |
| | | | 32 (ZARIXSD2) |
| | | | 31 (EDID4) |
| | | | 29 (BSIS) |
```

```
| | | | 28 (CDPOS) |
| | | | 25 (ZARIXMM2) |
| | | | 18 (KONP) |
| | | | |
|Row store size | 129| 129 (hlahana21) | 37 (A726) |
| | | | 30 (TST03) |
| | | | 12 (EDIDS) |
| | | | 7 (SRRELROLES) |
| | | | 5 (EDIDC) |
| | | | 4 (D010TAB) |
| | | | 4 (SWNCMONI) |
| | | | 3 (/SDF/MON) |
| | | | 3 (DD03L) |
| | | | 2 (REPOSRC) |
| | | | |
|Disk size | 1194| 1194 (global) | 320 (KONV) |
| | | | 104 (BSEG) |
| | | | 42 (ZARIXSD5) |
| | | | 36 (VBFA) |
| | | | 32 (ZARIXSD2) |
| | | | 30 (EDID4) |
| | | | 30 (TST03) |
| | | | 29 (BSIS) |
| | | | 27 (CDPOS) |
| | | | 25 (ZARIXMM2) |
-------------------------------------------------------------------------------------------------------
-----------------------------------------------------
```

The heap memory size is reported with 1508 GB which is much more than the 447 GB from further above. The reason is that in the second result list all heap areas are considered, also the ones that are the basis for the column store. This means, most of the 1508 GB heap allocation overlaps with the column store size. The shared memory size of 121 GB overlaps with the row store.

The allocated instance memory of 3450 GB is much higher than the used instance memory of 1639 GB, because SAP HANA tends to keep allocated memory allocated as long as there is no memory shortage. From a sizing perspective the used memory matters.

So also the memory overview output indicates that the used memory is significantly below 2 TB and far away from the 4 TB memory limitation.

A closer look into the top heap areas (*SQL: "HANA_Memory_TopConsumers", AREA= 'HEAP', AGGREGATE_BY = 'DETAIL'*) shows the following top allocators for the same system:

```
------------------------------------------------------------------------------- |DETAIL
|SIZE_GB
-------------------------------------------------------------------------------
|Pool/PersistenceManager/PersistentSpace(0)/DefaultLPA/Page | 105.70|
|Pool/RowEngine/QueryExecution | 84.32
|Pool/Statistics | 65.97
|Pool/JoinEvaluator/TranslationTable | 24.90| ------------------------------------------------
---------------------------------
```

The Page allocator being responsible for a memory utilization of 106 GB is a kind of file system buffer that can reduce its size without problems whenever there is a memory shortage. So we can assume that another around 80 GB could be saved if required. This means that the total required memory is 1550 GB.

**Conclusion:** Even if the used memory size doubles it is still well below the memory limit (3100 GB vs. 4000 GB) and can also handle exceptional situations (e.g. significant growth of certain heap allocators) without running into memory pressure.

It is useful to repeat this analysis from time to time and also check the historic memory utilization (*SQL: "HANA_Memory_TopConsumers_History"*) to get a good understanding of the memory requirements over time.

### 11. Is it possible to monitor the memory consumption of SQL statements?

You can activate the statement memory tracking feature by setting the following parameters:

```
global.ini -> [resource_tracking] -> enable_tracking = on global.ini -> [resource_tracking] -> memory_tracking = on
```

Changes to both parameters can be done online, no restart is required.

When memory tracking is active, the following memory information is available:

| Patch level | Table | Column |
|---|---|---|
| >= Rev. 1.00.80 | M_EXPENSIVE_STATEMENTS | MEMORY_SIZE |
| >= Rev. 1.00.94 | M_ACTIVE_STATEMENTS<br>M_PREPARED_STATEMENTS | ALLOCATED_MEMORY_SIZE USED_MEMORY_SIZE<br>AVG_EXECUTION_MEMORY_SIZE<br>MAX_EXECUTION_MEMORY_SIZE<br>MIN_EXECUTION_MEMORY_SIZE<br>TOTAL_EXECUTION_MEMORY_SIZE |
| >= Rev. 1.00.94 >= Rev. 1.00.100 | M_CONNECTION_STATISTICS<br>M_SQL_PLAN_CACHE | AVG_EXECUTION_MEMORY_SIZE<br>MAX_EXECUTION_MEMORY_SIZE<br>MIN_EXECUTION_MEMORY_SIZE<br>TOTAL_EXECUTION_MEMORY_SIZE |

Due to a bug with Rev. 1.00.90 to 1.00.96 (SAP Note 2164844) the setting will only work if additionally also the statement_memory_limit parameter (see below) is set.

Before Rev. 1.00.94 the expensive statement trace could only be triggered by runtimes of SQL statements. Starting with Rev. 1.00.94 you can use the following parameter to trigger the recording of expensive SQL statements in M_EXPENSIVE_STATEMENTS based on the memory consumption:

```
global.ini -> [expensive_statement] -> threshold_memory = <bytes>
```

### 12. Is it possible to limit the memory that can be allocated by a single SQL statement?

Starting with SPS 08 you can limit the memory consumption of single SQL statements. As a prerequisite you need to have the statement memory tracking feature enabled as described above. Additionally you have to set the following parameter in order to define the maximum permitted memory allocation per SQL statement and host:

```
global.ini -> [memorymanager] -> statement_memory_limit = <maximum_memory_allocation_in_gb>
```

For more details see SAP Note 2222250 ("How can workload management be configured for memory?").

### 13. What can I do if a certain heap allocator is unusually large?

See SAP Note 1840954 for some general advice.

The following table contains allocator-specific recommendations. Normally there is no need to perform manual analysis and optimization, so make sure that you are in a pathologic or critical situation before you

consider any changes:

| Allocator | Purpose | Analysis Steps |
| --- | --- | --- |
| AllocateOnlyAllocator-unlimited/F LA-UL<3145728,1>/MemoryMapLevel2Blocks (SAP HANA 1.0) AllocateOnlyAllocator-unlimited/F LA-UL<24592,1>/MemoryMapLevel3Nodes (SAP HANA >= 2.0) | Internal memory management | This allocator contains information for managing the SAP HANA memory. Normally no optimization is necessary. A high memory utilization or frequent memory defragmentations in the system can result in the allocation of smaller memory chunks, which will result in a larger allocator. Sizes up to 5 % of the global allocation limit are typically acceptable. Higher values are typically a consequence of a high memory pressure. In this case you should consider the following optimizations: Avoid manual memory defragmentations (hdbcons 'mm gc -f', gc_unused_memory_threshold_* parameters) Avoid resource container shrinks (hdbcons 'resman shrink', unload_upper_bound parameter) or make sure that reasonable (not too small) thresholds) are used. Avoid manual reductions of parameters async_free_target and async_free_threshold (SAP Note 2169283). Analyze and optimize the general memory situation (e.g. in terms of data volume, memory leaks, intermediate results) in order to reduce the amount of implicit memory defragmentations or resource container shrinks. If you experience a very high (and possibly rising) memory consumption due to this allocator, you can determine details with the following hdbcons commands (SAP Note 2222218): SAP HANA 1.0: mm level2map SAP HANA >= 2.0: mm pagetable Large sizes of this allocator can indicate the risk of running into address space limitations. 1 GB of allocator size represents around 170 GB of address space. See "Which indications exist that an OOM situation is triggered by the operating system?" below for more details. SAP HANA will run into address space related OOM situations at latest when the following limits are reached: Architecture Unlimited stack Limit size Intel no 768 GB Intel yes 512 GB IBM on Power (no BIGMEM) no 96 GB IBM on Power (no BIGMEM) yes 64 GB IBM on Power (BIGMEM) no 384 GB IBM on Power (BIGMEM) yes 256 GB Starting with SAP HANA 2.0 the memory management is optimized and particularly large sizes of this allocators are less likely. |

| | | |
|---|---|---|
| Pool/AdapterOperationCache | SDQ adapter operation cache | This heap allocator is used by the Smart Data Quality (SDQ) adapter operation cache. See SAP Note 2502256 for more information about SAP HANA caches in general and the adapter operation cache in particular. The cache can be disabled via: scriptserver.ini -> [adapter_operation_cache] -> enable_adapter_operation_cache = no |
| Pool/Auditing | Auditing | This allocator stores auditing related information (SAP Note 2159014). In case of large and rising sizes you can consider to disable auditing as a temporary workaround. |
| Pool/BWFlattenScenario | BW infocube conversion | This allocator is used during conversions of infocubes to HANA optimized cubes using BW_CONVERT_CLASSIC_TO_IMO_CUBE. This procedure is executed when a classic infocube is converted to an in-memory optimized infocube using transaction RSMIGRHANADB. Increased memory consumption is normal when large infocubes are converted. After the conversion of the existing infocubes is finished, executing this procedure is no longer required and the allocator size reduces. |
| Pool/AttributeEngine/Delta Pool/AttributeEngine/Delta/BtreeDictionary Pool/AttributeEngine/Delta/Cache Pool/AttributeEngine/Delta/InternalNodes Pool/AttributeEngine/Delta/LeafNodes Pool/ColumnStore/Delta/BtreeDictionary Pool/ColumnStore/Delta/Btreeindex Pool/ColumnStore/Delta/Cache Pool/ColumnStore/Delta/InternalNodes Pool/ColumnStore/Delta/LeafNodes Pool/ColumnStoreTables/Delta/BtreeDictionary Pool/ColumnStoreTables/Delta/Btreeindex Pool/ColumnStoreTables/Delta/Cache Pool/ColumnStoreTables/Delta/InternalNodes Pool/ColumnStoreTables/Delta/LeafNodes | Delta storage components | See SAP Note 2057046 and make sure that delta merges are properly configured and executed, so that the delta storage size of the tables remains on acceptable levels. |
| Pool/AttributeEngine Pool/AttributeEngine/idattribute Pool/AttributeEngine-IndexVector-BlockIndex Pool/AttributeEngine-IndexVector-BTreeIndex Pool/AttributeEngine-IndexVector- | Column store components | These allocators are responsible for parts of the column store. Their memory allocation will implicitly reduce if you reduce the amount of table data in column store (archiving, cleanup, reduction of indexes, ...) |

| | | |
|---|---|---|
| Single Pool/AttributeEngine-IndexVector-<br>SingleIndex Pool/AttributeEngine-<br>IndexVector-<br>Sp-Cluster Pool/AttributeEngine-<br>IndexVector-<br>Sp-Indirect Pool/AttributeEngine-<br>IndexVector-<br>Sp-Prefix Pool/AttributeEngine-<br>IndexVector-<br>Sp-Rle Pool/AttributeEngine-IndexVector-<br>Sp-Sparse<br>Pool/ColumnStore/Main/Compressed/<br>Cluster<br>Pool/ColumnStore/Main/Compressed/<br>Indirect<br>Pool/ColumnStore/Main/Compressed/<br>Prefix<br>Pool/ColumnStore/Main/Compressed/<br>Rle Pool/ColumnStore/Main/Compressed/<br>Sparse<br>Pool/ColumnStore/Main/Dictionary/<br>RoDict Pool/ColumnStore/Main/Dictionary/<br>ValueDict<br>Pool/ColumnStore/Main/Index/Block<br>Pool/ColumnStore/Main/Index/Singl<br>e Pool/ColumnStore/Main/PagedUncomp<br>ressed Pool/ColumnStore/Main/Rowid<br>Pool/ColumnStore/Main/Text/DocObj<br>ects<br>Pool/ColumnStore/Main/Uncompresse<br>d Pool/ColumnStoreTables/Main/Compr<br>essed/Cluster<br>Pool/ColumnStoreTables/Main/Compr<br>essed/Indirect<br>Pool/ColumnStoreTables/Main/Compr<br>essed/Prefix<br>Pool/ColumnStoreTables/Main/Compr<br>essed/Rle<br>Pool/ColumnStoreTables/Main/Compr<br>essed/Sparse<br>Pool/ColumnStoreTables/Main/Dicti<br>onary/RoDict<br>Pool/ColumnStoreTables/Main/Dicti<br>onary/ValueDict<br>Pool/ColumnStoreTables/Main/Index<br>/Block<br>Pool/ColumnStoreTables/Main/Index<br>/Single<br>Pool/ColumnStoreTables/Main/Paged<br>Uncompressed<br>Pool/ColumnStoreTables/Main/Rowid<br>Pool/ColumnStoreTables/Main/Text/<br>DocObjects | | |

| | | |
|---|---|---|
| Pool/ColumnStoreTables/Main/Uncompressed Pool/NameIdMapping/RoDict | | |
| Pool/ColumnStore/Main/Rowid/build-reverse-index Pool/ColumnStoreTables/Main/Rowid/build-reverse-index | Temporary structure when creating reverse index on $rowid$ column | This allocator is used during specific operations like optimize compression runs (SAP Note 2112604) when a reverse index is created on the $rowid$ column. |
| Pool/AttributeEngine/Transient Pool/AttributeEngine/Transient/updateContainerConcat | Transient column store information | These allocators contain temporary column store data. They can grow significantly in case of index creation (SAP Note 2160391). The behavior is improved with with SAP HANA 1.00.122.11 SAP HANA 2.0. As a workaround you can consider creating indexes at a time when the underlying tables are empty or filled to a minor extent. |
| Pool/BitVector | Basic data structure (e.g. temporary query results, columnar data, transactional info of column tables) | Can be linked to problems with garbage collection in column store, see "Which options exist to reduce the risk of SAP HANA memory issues?" -> "Transactional problems" for details. |
| Pool/CacheMgr/CE_ScenarioModelCache | Calculation engine model cache | This heap allocator is used for the calculation engine model cache (SAP Note 2502256). |
| Pool/CacheMgr/CS_QueryResultCache [Realtime] Pool/CacheMgr/CS_QueryResultCache [TimeControlled] | Query result cache | These heap allocators are linked to the query result cache (SAP Note 2014148). |
| Pool/CacheMgr/CS_StatisticsCache | Column store statistics cache | This heap allocator is used for the column store statistics cache (SAP Note 2502256). |
| Pool/CacheMgr/DataStatisticsAdviserCache | Data statistics adviser cache | This heap allocator is used for the data adviser statistics cache (SAP Note 2502256). |
| Pool/commlibDefAllocator Pool/ncCommLibDefAllocator | Network communication support objects | This allocator can grow in case a large number of network channels exist (due to inter-node or inter-service communication), see SAP Note 2222200 for more information related to SAP HANA network. A memory leak issue was fixed with SAP HANA 1.00.122.05. Another memory leak is fixed with SAP HANA 2.00.021. |
| Pool/Crypto | Encryption related data structures | This allocator can grow with SAP HANA 2.00.020 and 2.00.021 due to a memory leak related to hash functions (e.g. HASH_SHA256). You need to restart SAP HANA in order to reclaim the allocated memory. The related call stack module (that can be checked via an hdbcons allocator block list as described in SAP Note 2222218) is |

| | | |
|---|---|---|
| | | Crypto::Provider::CommonCryptoProvider::initHash. |
| Pool/CSPlanExecutor/PlanExecution | Intermediate data structures | See question "Which general optimizations exist for reducing the SQL statement memory requirements?" below in order to make sure that SQL statements are executed as memory-efficient as possible. This allocator is used during plan execution of a database request as a fallback allocator used in cases where specific statement allocators aren't available. |
| Pool/CSRowLocking | Column store row locking | This allocator is typically large if many record locks exist. You can check the current record locks via SQL: "HANA_Locks_Transactional_Current " (SAP Note 1969700). The allocator is purged asynchronously during delta merge (SAP Note 2057046) and column unload (SAP Note 2127458). Only an unload guarantees that all entries for the related table are completely purged. There is no possibility to map allocations to specific tables. In systems with mass modifications it is normal that this allocator can grow and remain at a certain size although there are no current record locks. In general this does neither indicate a memory leak nor a bottleneck. In case of permanent sizes of more than 50 GB a more detailed analysis is useful. |
| Pool/CS_TableSearch | Query optimizer related data structures | See question "Which general optimizations exist for reducing the SQL statement memory requirements?" below in order to make sure that SQL statements are executed as memory-efficient as possible. |
| Pool/DeletedPageList | Recording of DELETE operations in row store | This allocator is typically quite small, but in context of the problem described in SAP Note 2253017 (SAP HANA Rev. 100 - 102.02) it can already become critical in case of small sizes >= 2 MB. |
| Pool/DocidValueArray | Set of rowids and related values in context of join engine | See question "Which general optimizations exist for reducing the SQL statement memory requirements?" below in order to make sure that join SQL statements are executed as memory-efficient as possible. |
| Pool/DPServerFramework Pool/DPServerStatsRequestIfacerAllocator | Data provisioning server memory | These allocators are used by the data provisioning server (dpserver) that is used in smart data integration (SDI) scenarios (SAP Note 2400022). The following known reasons for large and rising allocator sizes exist: SAP Note 2542700: Memory leak accessing remote cluster tables (SAP HANA <= 1.00.122.12, <= 2.00.012.02, <= 2.00.021) SAP Note 2643641: |

| | | |
|---|---|---|
| | | Risk of increased memory requirements in context of streaming |
| Pool/DSO/DSORead<br>Pool/DSO/DSOUpdate | DSO activation / rollback | These allocators temporarily store data during DSO operations in BW like DSO_ACTIVATE_PERSISTED, DSO_ROLLBACK_PERSISTED, DSO_ACTIVATE_CHANGES and DSO_ROLLBACK_CHANGES. Once these activities are finished, the majority of the allocated memory is released. |
| Pool/DynamicCachedView<br>Pool/DynamicCachedView/ViewMatching | Dynamic result cache information | These allocators contain information related to the dynamic result cache (SAP Note 2506811). |
| Pool/entityCache | MDX entity cache | This allocator contains MDX entity cache information. You can use SQL: "HANA_Memory_Caches_Overview" and SQL: "HANA_Memory_Caches_Entries" (SAP Note 1969700) with CACHE_NAME = 'MdxEntityCache' to display and understand details about the current utilization of the entity cache. The life time of entries in the entity cache doesn't depend on the execution of the underlying SQL statements, they are maintained independently. The entity cache is part of the SAP HANA resource container that is shrunk whenever the memory gets scarce. It is unloaded with a higher priority than columns so that a large cache size isn't necessarily critical. See SAP Note 2502256 for more information related to SAP HANA caches. |
| Pool/ESX | ESX runtime data | The Extended SQL Executor (SQL) uses Pool/ESX for storing runtime data. SAP Note 2597818 describes a SAP HANA memory leak with Revisions 2.00.000 to 2.00.023 that can result in a growth of this heap allocator in context of PlanViz executions (SAP Note 2073964). |
| Pool/ExecutorPlanExecution | Intermediate result sets | See question "Which general optimizations exist for reducing the SQL statement memory requirements?" below in order to make sure that join SQL statements are executed as memory-efficient as possible. |
| Pool/FemsCompression/CompositeFemsCompression | FEMS compression | The field element selection (FEMS) compression is used for BW queries with execution modes 2 and 3 in order to reduce the amount of data transferred to the BW OLAP engine within SAP HANA. In some cases FEMS can result in increased memory requirements. See BW on HANA and the Query Execution Mode for more information related to BW query execution modes. As a |

| | | |
|---|---|---|
| | | local workaround you can check if executing the query in question in execution mode 0 is an acceptable alternative. Also execution mode 2 instead of 3 is worth a try, because the underlying FEMS activities are different and may not run the same issues. As a global workaround you can disable FEMS compression in method _GET_TREX_REQ_FLAGS_READ of class CL_RSDRV_TREX_API_STMT by commenting the following line with a leading '*' (see pilot SAP Note 1828751): r_trex_req_flags = r_trex_req_flags + 33554432. As this will lead to disadvantages in other areas (e.g. increasing amount of transmitted data), you should undo this change once you have understood and fixed the reason for the high FEMS related memory consumption. |
| Pool/Filter | intermediate result sets | See question "Which general optimizations exist for reducing the SQL statement memory requirements?" below in order to make sure that join SQL statements are executed as memory-efficient as possible. This allocator is used in different contexts, e.g.: Pruning (join engine, OLAP engine, TableUpdate, TRexApiSearch) Hierarchy filter analysis In general the memory should be released once the related database request is finished. |
| Pool/FRSWLockAllocator | Read write locks | Starting with SAP HANA 1.00.122.13 and 2.00.010 read write locks are no longer stored in the Pool/Statistics allocator, instead they are maintained in the dedicated Pool/FRSWLockAllocator ("fast read slow write lock allocator"). In case of a large size you can check for read write lock details via SQL: "HANA_Locks_Internal_LockWaits_Ov erview" (LOCK_TYPE = 'READWRITELOCK') available via SAP Note 1969700. With SAP HANA 2.0 the space consumption of this allocator is reduced compared to SAP HANA 1.0. The following known issues with a high number of recorded locks exist: SAP HANA <= 1.00.122.17, <= 2.0 SPS 00: PlanInfoLock SAP HANA <= 1.00.122.17, <= 2.0 SPS 00: PreferredRoutingLocations See SAP Note 1999998 for more information related to SAP HANA locks. |
| Pool/hierarchyBlob | Hierarchy cache, MDX hierarchy cache | This allocator contains hierarchy cache and MDX hierarchy cache information that is populated when you query SAP HANA views with hierarchies. You can use SQL: |

| | | |
|---|---|---|
| | | "HANA_Memory_Caches_Overview" and SQL: "HANA_Memory_Caches_Entries" (SAP Note 1969700) with CACHE_NAME = '%HierarchyCache' to display and understand details about the current utilization of the hierarchy cache. The life time of entries in the hierarchy cache doesn't depend on the execution of the underlying SQL statements, they are maintained independently. The hierarchy cache is part of the SAP HANA resource container that is shrunk whenever the memory gets scarce. It is unloaded with a higher priority than columns so that a large cache size isn't necessarily critical. If hierarchies / caching isn't required, you can disable it by setting cache=false in the hierarchy definitions, 'Drill Down Enablement' = ' ' (instead of 'Drilldown') in SAP HANA Studio or globally by disabling it with the following parameter: indexserver.ini -> [cache] -> hierarchies_transactional_cache_e nabled = 'false' See SAP Note 2502256 for more information related to SAP HANA caches. |
| Pool/IndexRebuildAllocator | Memory area for row store index rebuilds | This issue can happen with SAP HANA 1.0 SPS 07 and SPS 08. See SAP Note 2005478 and set the following parameter as a workaround in order to disable row store index rebuilds during startup: indexserver.ini -> [row_engine] -> use_jobex_index_rebuild = false |
| Pool/IndexVector Pool/IndexVectorAligned | Temporary index vector structures | This allocator is used in different contexts like table optimizations, column load, column write, binary import or index creation. A temporary large value is typically caused by delta merges (SAP Note 2057046) of tables with paged attributes (SAP Note 1871386), e.g. in the context of data aging (SAP Note 2416490). |
| Pool/itab Pool/itab/VectorColumn | Column store (intermediate) search results | See question "Which general optimizations exist for reducing the SQL statement memory requirements?" below in order to make sure that SQL statements are executed as memory-efficient as possible. If no explanation for a large and rising Pool/itab allocator is found, an itab leak trace can be activated as described in SAP Note 2074981 (SAP internal). |
| Pool/JERequestHandler | Temporary structure during translation table creation | This allocator is required in certain scenarios when translation tables are created to support join operations. See SAP Note 1998599 for more information related to translation tables. |
| Pool/JoinEvaluator | Global join | See question "Which general optimizations |

| | | |
|---|---|---|
| | engine allocator | exist for reducing the SQL statement memory requirements?" below in order to make sure that SQL statements are executed as memory-efficient as possible. This is the main join engine allocator that should normally not be used for significant memory allocations. Instead sub-allocators are used. If you see a high allocation, please check if also some sub-allocators are significantly filled. If yes, proceed with the analysis based on the Pool/JoinEvaluator/* sub allocators as described below. SAP Note 2370588 describes a problem that results in an increased size for Pool/JoinEvaluator, but at the same time also the sub-allocator Pool/JoinEvaluator/JECalculate/Re sults is extremely large. A large size of Pool/JoinEvaluator has been observed with SAP HANA Rev. 122.05 in combination with fast data access (FDA), so you can consider deactivating FDA for FOR ALL ENTRIES as described in SAP Note 2399993 via rsdb/prefer_join_with_fda and dbs/hdb/prefer_join_with_fda = 0. |
| Pool/JoinEvaluator/DictsAndDocs | Join engine dictionaries | See question "Which general optimizations exist for reducing the SQL statement memory requirements?" below in order to make sure that SQL statements are executed as memory-efficient as possible. This allocator is usually linked to SAP HANA SQL statement processing in call stack modules like AttributeEngine::AttributeApi::je GetDictAndDocs and JoinEvaluator::JEDistinctAttribut e::getDictAndDocs. Among others the following scenarios can be responsible for a significant growth of this allocator: Check if inefficient joins of partitioned tables are responsible and reduce or optimize partitioning. Check for COUNT DISTINCT operations on large (partitioned) tables and columns with many distinct values (SAP Note 2000002 -> "What are typical approaches to tune expensive SQL statements?" -> "High runtime of COUNT DISTINCT"). This allocator may also grow when an index is created because the join engine takes over some data processing. |
| Pool/JoinEvaluator/JECalculate Pool/JoinEvaluator/JECalculate/Tm pResults Pool/JoinEvaluator/JECreateNTuple Pool/JoinEvaluator/JEPreAggregate | Join engine intermediate data structures | See question "Which general optimizations exist for reducing the SQL statement memory requirements?" below in order to make sure that SQL statements are executed as memory-efficient as possible. If you can't directly |

| | | |
|---|---|---|
| Pool/JoinEvaluator/JEStep1<br>Pool/JoinEvaluator/JEStep2<br>Pool/JoinEvaluator/NTuple | | identify the SQL statements responsible for the memory growth, you can use SQL: "HANA_Threads_ThreadSamples_FilterAndAggregation" (THREAD_DETAIL = '%(JE%', AGGREGATE_BY = 'HASH, THREAD_DETAIL') available via SAP Note 1969700 to check for SQL statements with a significant processing time in related join engine functions. Consider setting the hint USE_OLAP_PLAN (SAP Note 2142945) for testing purposes in order to check if a switch from join engine to OLAP engine works and results in a reduced memory consumption. Huge allocations in Pool/JoinEvaluator/JECreateNTuple in combination with anti joins (e.g. EXCEPT) and call stacks in JoinEvaluator::LoopJob::findJoinPairsTL_native can be caused by a SAP HANA bug that is fixed with Rev. 1.00.122.12 and 2.00.010. With SAP HANA >= 2.0 SPS 02 the fix is enabled per default. With SAP HANA <= 2.0 SPS 01 the fix is disabled per default and can be activated with hint CONSERVATIVE_CS_ANTI_JOIN_ESTIMATION or globally with the following parameter: indexserver.ini -> [sql] -> conservative_cs_anti_join_estimation_enabled = true As a workaround the NO_GROUPING_SIMPLIFICATION hint (SAP Note 2142945) can be used. If triggered by BW / MDX, you can also disable the RSADMIN parameter MDX_F4_USE_SQL (SAP Note 1865554). |
| Pool/JoinEvaluator/JEPlanData/deserialized | Join engine intermediate data structures involving inter-node communication | See question "Which general optimizations exist for reducing the SQL statement memory requirements?" below in order to make sure that SQL statements are executed as memory-efficient as possible. Check if the distribution of involved tables across nodes is already optimal or if you can adjust it so that less inter-node data transfer is required. See SAP Note 2081591 for more information about SAP HANA table distribution. Consider setting the hint USE_OLAP_PLAN (SAP Note 2142945) for testing purposes in order to check if a switch from join engine to OLAP engine works and results in a reduced memory consumption. |
| Pool/JoinEvaluator/JEAssembleResults Pool/JoinEvaluator/JEAssembleResults/Results Pool/JoinEvaluator/JECalculate/Re | Join engine results | See question "Which general optimizations exist for reducing the SQL statement memory requirements?" below in order to make sure that SQL statements are executed as memory- |

| | | |
|---|---|---|
| sults Pool/JoinEvaluator/JERequestedAttributes/Results | | efficient as possible. If you can't directly identify the SQL statements responsible for the memory growth, you can use SQL: "HANA_Threads_ThreadSamples_FilterAndAggregation" (THREAD_DETAIL = '%(JE%', AGGREGATE_BY = 'HASH, THREAD_DETAIL') available via SAP Note 1969700 to check for SQL statements with a significant processing time in related join engine functions. This allocator can grow considerably when late materialization isn't used. For some reasons (e.g. bugs described in SAP Note 1975448) the following parameters may be increased, resulting in higher memory requirements: indexserver.ini -> [search] -> late_materialization_threshold indexserver.ini -> [search] -> late_materialization_threshold_for_insert Unset these parameters as soon as you have another solution in place (e.g. a revision level with included bug fix). Consider setting the hint USE_OLAP_PLAN (SAP Note 2142945) for testing purposes in order to check if a switch from join engine to OLAP engine works and results in a reduced memory consumption. Other reasons for a high memory consumption are: SAP Note 2174236 (bug in SAP HANA Rev. 91, fixed with Rev. 92) SAP Note 2260972 (inappropriate implementation of statistics server procedures) SAP Note 2370588 (inappropriate coding of S/4HANA migration routines) Increased allocation due to missing calc view unfolding in context of BETWEEN filter with decimal notation (fixed with >= 1.00.122.15, >= 2.00.012.04 and 2.00.024) |
| Pool/JoinEvaluator/PlanDataAttrVals/Deserialized | Join engine results | See question "Which general optimizations exist for reducing the SQL statement memory requirements?" below in order to make sure that SQL statements are executed as memory-efficient as possible. This allocator is used when join engine results have to be sent from one node to another in scale-out scenarios. |
| Pool/JoinEvaluator/TranslationTable | Join column mapping | Translation tables are required to map value IDs of join column values. SAP Note 1998599 describes how they can be configured in order to optimize memory consumption. In certain scenarios a significant memory requirement is linked to caching of translation tables related to joins with temporary tables. As of SAP HANA Rev. 102.02 translation tables related to temporary table joins are no longer kept. With |

| | | |
|---|---|---|
| | | SAP HANA SPS 09 and Rev. 97.02 and higher you can set the following parameter: indexserver.ini -> [joins] -> cache_temp_translation_tables = 'false' See SAP Note 2217936 for more information. |
| Pool/JoinEvaluator/ValueList | Intermediate join engine value list | See question "Which general optimizations exist for reducing the SQL statement memory requirements?" below in order to make sure that SQL statements are executed as memory-efficient as possible. An increased size during an S/4HANA migration can be a consequence of the issue described in SAP Note 2370588. |
| Pool/L/jit/MetaData Pool/L/llang/Debuggee | Intermediate Llang structures (for compiled programs / for interpreting and debugging) | These heap allocators contain L related information. With SAP HANA <= 2.00.023 the Pool/L/llang/Debuggee allocator is not part of the SAP HANA resource container and so it can't be shrinked in case of low memory. This scenario can become critical in context of the SAP HANA Execution Engine (HEX) activation (SAP Note 2570371) with SAP HANA 2.0 SPS 02 when many parsed queries are still contained in the SQL cache and so the allocator size can be high. Starting with SAP HANA 2.00.024 this problem is fixed and the allocator will be part of the resource container so that it can be shrinked when memory is required. As a workaround for SAP HANA <= 2.00.023 you can set the following SAP HANA parameters: indexserver.ini -> [execution] -> compilation_strategy = always indexserver.ini -> [execution] -> asynchronous_compilation = false Clearing the SQL cache (SAP Note 2124112) can be an immediate action to reduce the allocator size with SAP HANA <= 2.00.023: ALTER SYSTEM CLEAR SQL PLAN CACHE Be aware that clearing the SQL cache will result in increased parsing requirements and so it should only be performed in exceptional situations. |
| Pool/L/llang/Runtime/Local | Intermediate Llang script results | See question "Which general optimizations exist for reducing the SQL statement memory requirements?" below in order to make sure that SQL statements are executed as memory-efficient as possible. Llang queries may be manually created or are results of FOX, SQLScript or HEX (SAP Note 2570371). The allocator can grow if a large amount of strings or CLOB values is processed. The name of the Llang program can be found in the call stack, e.g. _SYS_PLE:20160126114423_4338930:TMPDATA in the following case: 20: 0x00007fdd4b55b8b1 in |

| | | |
|---|---|---|
| | | ljit::dynamic/_split_main_0+0x387 0 at fox/cen_"SAPSR3"."_SYS_PLE:20160126114423_4338930:TMPDATA". cv053_fox_LLangView.56A7F0213FB8EFC7E10000000AAA0052:0 (<unknown>) If required, you can activate a trace for the ljit component on debug level (see SAP Note 2119087): indexserver.ini -> [trace] -> ljit = 'debug' |
| Pool/LVCAllocator/LVCContainerDir Pool/LVCAllocator/LVCContainerDir /LVCContainer_<id> Pool/LVCAllocator/LVCObjectPageDir Pool/LVCAllocator/LVC_ObjectPageDir | liveCache data | These allocators hold the actual liveCache data and so their sizes should correspond to the amount of liveCache data. See SAP Note 2593571 for more information related to liveCache. |
| Pool/malloc/hdbnameserver | Temporary nameserver data structures | This allocator is used for temporary nameserver data structures that are e.g. used in context of Python activities triggered by actions like a full system info dump (SAP Note 2573880) or a performance trace (SAP Note 2520774). |
| Pool/malloc/libdbrsa16_r.so | Allocations of RSA library for Sybase IQ | The libdbrsa16_r.so library is used for RSA functionalities in context of Sybase IQ remote accesses. In SAP HANA contexts these accesses are usually related to smart data access (SDA, SAP Note 2180119). |
| Pool/malloc/libc.so.6 | Linux libc allocations | This allocator is used when memory allocation is done by the Linux libc.so library, e.g. in the context of file system accesses like __alloc_dir and System:UX:opendir. If you see this allocator in the context of an out-of-memory situation, you can assume that it is only a victim and not responsible for OOM. If for example the compileserver issues an OOM when allocating memory for Pool/malloc/libc.so.6, you should check the indexserver at first, because very likely it has consumed all available memory before. |
| Pool/malloc/libhdbbasement.so | Column store data structures | A large and growing size can be caused by the following reasons: A memory leak in SAP HANA <= 1.00.122.06 can result in growth of this allocator related to module TrexThreads::InheritableLocalStorage::cloneMap. This problem is fixed with SAP HANA 1.00.122.07. This can also happen in consistency check contexts (SAP Note 2547516). Using the function profiler with SAP HANA <= 1.00.122.16, <= 2.00.024.01 and 2.00.030 can result in a growing allocator size (SAP Note 2637828). |
| Pool/malloc/libhdbcalcengine.so | Calculation | See question "Which general optimizations |

| | | |
|---|---|---|
| Pool/malloc/libhdbcalcengineapi.so | engine intermediate results | exist for reducing the SQL statement memory requirements?" below in order to make sure that SQL statements are executed as memory-efficient as possible. The following individual reasons can be responsible for increased allocator sizes: If you observe a growth of this allocator in combination with calculation engine processing (e.g. TREXviaDBSL, call stack modules like TrexCalculationEngine::Optimizer::optimizeHierarchyJoinOverMultiprovider and TrexCalculationEngine::CombineNonRootAggrOverMpRule::applyAggrOverHierachyJoin), you face a SAP HANA bug that is fixed with SAP HANA Rev. 97.02 and 102. SAP Note 2374935 describes a memory leak in the context of calculation scenario modeler objects that is fixed with SAP HANA Rev. 112.07 and 122.04. |
| Pool/malloc/libhdbcalcenginepops.so | Intermediate results during calculation engine plan operation processing | See question "Which general optimizations exist for reducing the SQL statement memory requirements?" below in order to make sure that SQL statements are executed as memory-efficient as possible. In most cases cePopInternalJoin will be the main contributor for the allocator size. As a workaround you can use calculation view unfolding, e.g. via CALC_VIEW_UNFOLDING hint (SAP Note 2142945). |
| Pool/malloc/libhdbcsapi.so | Column store API (search) and intermediate results | See question "Which general optimizations exist for reducing the SQL statement memory requirements?" below in order to make sure that SQL statements are executed as memory-efficient as possible. Additionally you can check for the following specific constellations: If you observe a growth of this allocator in combination with calculation engine processing (e.g. TREXviaDBSL, call stack modules like TrexCalculationEngine::Optimizer::optimizeHierarchyJoinOverMultiprovider and TrexCalculationEngine::CombineNonRootAggrOverMpRule::applyAggrOverHierachyJoin), you face a SAP HANA bug that is fixed with SAP HANA Rev. 1.00.97.02 and 1.00.102. With SAP HANA <= 1.0 SPS 08 a large size of this allocator can be caused by the creation of join statistics. These statistics are dynamically created during the first execution of a specific join after a restart of SAP HANA. If the same kind of join is |

| | | |
|---|---|---|
| | | concurrently started in many different transactions, the efforts and memory requirements are also multiplied, because each transaction calculates the join statistics individually. As a workaround you can execute a critical join individually before running it with a higher parallelism. As of SAP HANA 1.0 SPS 09 the join statistics creation is improved and the problem no longer happens. This allocator can also grow when tables with large delta storages are accessed. Call stacks like TRexAPI::DeltaIndexManager::docId Search are indicators for this scenario. In this case you have to make sure that a reasonable merge strategy is implemented (SAP Note 2057046). Up to SAP HANA 1.0 SPS 12 this allocator is also used when the query result cache is configured (indexserver.ini -> [cache] -> query_result_cache = enabled; SAP Note 2014148). With newer patch levels dedicated result cache allocators are used instead. Large allocations can also be linked to queries with expensive fuzzy / text searches (call stack modules like ltt_adp::vector, TRexAPI::FreeStyleExecutor::addTo ken, TRexAPI::FreeStyleExecutor::creat eAllAlternatives, TRexAPI::FreeStyleExecutor::build FSQuery). Large allocations from module OlapEngine::BwPopSearch::setQuery EntryInList in context of BW multiproviders (or composite providers) and FEMS can be caused by missing filter pushdown or problems with a convex hull optimization (SAP HANA >= 1.00.122.10). See SAP Note 2517443 and also consider to set indexserver.ini -> [calc_engine] -> optimize_convex_hull_through_mp = 0 for testing purposes. With SAP HANA <= 1.00.122.15, <= 2.00.012.04 and <= 2.00.024 a memory leak in context of guided navigation searches (e.g. Enterprise Search) can result in a growing allocator size (SAP Note 2601475). |
| Pool/malloc/libhdbcs.so | Column store components | If you see particularly high values for this allocator, check the following typical reasons: Large indexes being created on a partitioned column store table can consume significant amounts of memory in these allocators. Check if you can avoid creating indexes on particularly large, partitioned column store tables. If the problem happens during a DMO activity, see SAP Notes 2257362 and 2578336 and make sure that critical indexes are created |

| | | |
|---|---|---|
| | | before the table data is loaded. Starting with SAP HANA 1.00.122.11 and 2.00.012.01 the memory footprint of index creations is optimized. Pool/malloc/libhdbcs.so may also grow during merges of large tables (see SAP Note 2057046) COUNT DISTINCT operations on large tables and columns with many distinct values can also be responsible for a growth of this allocator. |
| Pool/malloc/libhdbcsmd.so | Transient metadata | This allocator contains transient metadata like the last delta merge time or the number of values in main and delta storage. Additionally it is used by different SAP HANA engines for specific reasons. In order to understand the origination of the space consumption, you can run the hdbcons blocklist option (hdbcons 'mm bl -t Pool/malloc/libhdbcsmd.so'). |
| Pool/malloc/libhdbcsstore.so | Column store persistence objects | This allocator contains administrative column store information (like parts of the row lock information and transaction handling) and may grow in case of many locks or blocked garbage collection. If much memory is allocated by ptime::LinkHash / TrexStore::LockMapEntry*, it can be caused by an infrequent row lock link hashmap garbage collection. As a workaround you can trigger this garbage collection by unloading and reloading tables with a high INSERT / DELETE load. The problem is fixed with SAP HANA Revisions 102.04 and 111. |
| Pool/malloc/libhdbcstypes.so | Column store data types, hybrid LOB information | This allocator contains information about column store data types including hybrid LOB information like memory LOB values or disk LOB references. As of SAP HANA SPS 10 it is common to see sizes between 10 and 50 GB for larger databases, depending on the amount of hybrid LOB values existing in the system. This allocated can be treated similarly like the Pool/ColumnStoreTables allocators described above: Its size is closely linked to the column store table sizes (with a focus on hybrid LOB columns), and so it can mainly be reduced by reducing data stored in hybrid LOB columns. See SAP Note 2220627 for more information related to SAP HANA LOBs. |
| Pool/malloc/libhdbcswrapper.so | (Intermediate) results | See question "Which general optimizations exist for reducing the SQL statement memory requirements?" below in order to make sure that SQL statements are executed as memory-efficient as possible. The following scenarios can lead to an increased memory footprint of Pool/malloc/libhdbcswrapper: SAP HANA <= |

| | | |
|---|---|---|
| | | 1.00.122.16: Memory leak in context of mixed inverted index joins (SAP Note 2624305) |
| Pool/malloc/libhdbevaluator.so | Intermediate results | See question "Which general optimizations exist for reducing the SQL statement memory requirements?" below in order to make sure that SQL statements are executed as memory-efficient as possible. Database requests responsible for a growth of this allocator typically show evaluator specific modules like Evaluator::ThreeCode::run in their call stacks. |
| Pool/malloc/libhdbitab.so | Intermediate results | See Pool/itab for more information. In general Pool/malloc/libhdbitab.so should be small. If it is large and growing, a memory leak can be responsible with Rev. <= 97.02 and Rev. 100 to 102.00. |
| Pool/malloc/libhdbmetadataobject.so | Metadata | This allocator is linked to the SAP HANA metadata cache. A large and rising size can be a consequence of a bug that is fixed with SAP HANA >= 1.00.122.13. As a workaround clearing the SQL cache with "ALTER SYSTEM CLEAR SQL PLAN CACHE" can help to release no longer required entries and reduce the allocator size. |
| Pool/malloc/libhdbolap.so | Intermediate OLAP engine results | See question "Which general optimizations exist for reducing the SQL statement memory requirements?" below in order to make sure that SQL statements are executed as memory-efficient as possible. SAP Note 2373932 describes a memory leak in this allocator that is fixed with SAP HANA Rev. 112.07 and 120. |
| Pool/malloc/libhdbpartitioning.so | Intermediate results in context of partitioned tables | This generic allocator can grow in case of accesses to partitioned tables. If you experience temporary or permanent large sizes, open a SAP incident for further analysis. |
| Pool/malloc/libhdbpythonbase.so | Python initialization and execution | This allocator is related to a wrapper used for initializing and executing Python functions. If it is too large it may be due to the pythontrace was enabled for too long. See KBA 2519536 - Nameserver ran out of memory with Pool/malloc/libhdbpythonbase.so as top allocator |
| Pool/malloc/libhdbrskernel.so | Row store components | This allocator contains all dynamic row store memory allocations which aren't assigned to more specific allocators. With newer revisions the utilization of this allocator should reduce. You can use the options "mm bl" of hdbcons and additionally create an allocator stack strace if required (SAP Note 2222218) to determine the top consumers inside the |

| | | |
|---|---|---|
| | | allocator. The following individual reasons for an increased size exist: Top consumer SAP Note Details ptime::Proc_insert_parallel::execute If most space is allocated by ptime::Proc_insert_parallel::execute, it can be caused by a memory leak bug which is fixed with SPS 08. ptime::RowInsertReproducibleJobNode::preprocess 2253121 With SAP HANA Revisions between 90 and 102.02 this allocator can grow due to a bug that can be bypassed by setting the following parameter as a workaround: indexserver.ini -> [row_engine] -> dynamic_parallel_insert_max_workers = '1' ptime::codegen_qp2so::gen_code 2275252 With SAP HANA Revisions 100 to 102.01 and call stacks containing ptime::codegen_qp2so::gen_code you can suffer from the bug described in SAP Note 2275252. ltt::string_base<char, ltt::char_traits<char> >::enlarge_ ptime::traceThreadLockInfo ptime::ServiceThreadSamplerThread::run 2114710 If thread sample details are collected and very large SQL statements are running, the allocator can grow significantly, because every second a new copies are created and stored in Pool/malloc/libhdbrskernel.so. In order to reduce the allocation, you should search for extremely large SQL statements and avoid or reduce them as much as possible. As a temporary workaround you can also disable the collection of thread sample details: global.ini -> [resource_tracking] -> service_thread_sampling_monitor_thread_detail_enabled = false Be aware that this change impacts the supportability of the system and so it shouldn't be implemented on a permanent basis. This problem is fixed with Rev. 1.00.122.06 and 2.00.001, then only the first 256 characters will be collected. See also SAP Note 2000002 ("Are there known issues with particularly large SQL statement texts?") and consider a reduction of the maximum statement length limit. If you experience a large and rising size that can't be explained, open a SAP incident for clarification. |
| Pool/malloc/libhdbtableconsistencycheck.so | Table consistency check | This allocator is related to the consistency check procedure CHECK_TABLE_CONSISTENCY (see SAP Note 1977584). You can limit the number of |

| | | |
|---|---|---|
| | | concurrent executions on different tables or run it at times with less concurrent workload in order to reduce the risk of critical memory allocations. |
| Pool/malloc/libsapcrypto.so | Encryption related data structures | This allocator can grow with SAP HANA 2.00.020 to 2.00.021 due to a memory leak related to hash functions (e.g. HASH_SHA256). You need to restart SAP HANA in order to reclaim the allocated memory. A fix is available with SAP HANA 2.00.022. |
| Pool/mds | (intermediate) result sets of InA / MDS queries | See question "Which general optimizations exist for reducing the SQL statement memory requirements?" below in order to make sure that database requests are executed as memory-efficient as possible. The memory is only allocated while an InA / MDS query is executed. You can reduce the memory footprint by reducing the amount of processed and returned data. See SAP Note 2670064 for more information related to MDS. |
| Pool/mdx | MDX query allocations | As of SAP HANA SPS 09 several reasons for a high memory allocation of Pool/mdx are fixed. |
| Pool/Metadata/MetadataCache/MetadataGlobalCacheSlot | Metadata cache | The metadata cache allocator was introduced with SAP HANA SPS 12 and is used to store metadata locally that otherwise has to be retrieved from a remote SAP HANA node. It can grow significantly if many DDL operations are executed, because DDL operations invalidate existing cache entries. Reasons for increased sizes are: Blocked garbage collection (SAP Note 2169283) A problem exists in SAP HANA Rev. 120 to 122.03 that can result in increased metadata cache sizes. This problem is fixed with Rev. 122.04 (re-use of previous cache entries if possible). Before SAP HANA SPS 12 the metadata information was stored in temporary row tables and so the allocator Pool/RowEngine/RSTempPage was used. As a workaround in case of large metadata cache sizes you can clear the it manually: ALTER SYSTEM CLEAR METADATA CACHE This command will clear the cache on the SAP HANA node where you are currently logged on. Starting with SAP HANA 1.00.122.13 this command will clear the metadata cache on all SAP HANA nodes and you can add "AT '<host>:<port>'" if you want to clear only the cache for one specific host and service. |
| Pool/Metadata/SessionLocalItabCon | Temporary | This heap allocator exists with SAP HANA |

| | | |
|---|---|---|
| tainer | table information | SPS 12 and higher and is used to store the following information: Session local data and session local metadata of session local temporary tables Session local data of global temporary tables If you face a high size of this allocator you can check M_TEMPORARY_TABLES at first (e.g. using SQL: "HANA_Tables_Temporary_Tables" available via SAP Note 1969700). Reasons for increased sizes are: Unnecessary high amount of temporary tables On Rev. 1.00.120 to 1.00.122.05 and 2.00.000 the session local metadata consumes larger amounts of memory than required. This is fixed with Rev. 1.00.122.06 and 2.00.001. See SAP Note 2418950 for more details. |
| Pool/NetworkChannelCompletionHandler | Network channel completion interface | This allocator holds network channel related information. A high number of channels can increase the size of this allocator. The following options exist to optimize the allocator size: See SAP Notes 2222200 and 2382421 and make sure that a proper SAP HANA network configuration is in place. Avoid an unnecessary high amount of partitions that can increase the number of required network channels (SAP Note 2044468). |
| Pool/OptimizeCompression/<schema>:<table> Pool/OptimizeCompression/<schema>:_SYS_SPLIT_<table>~<partition> | Compression optimization | Allocators starting with Pool/OptimizeCompression are used during compression optimizations of tables. See SAP Note 2112604 and make sure that compressions area configured in a reasonable way. |
| Pool/parallel Pool/parallel/aggregates Pool/parallel/align Pool/parallel/compactcol Pool/parallel/ihm Pool/parallel/pop Pool/parallel/temp_aggregates Pool/parallel/temp_dimensions Pool/parallel/temp_other | OLAP aggregation results | See question "Which general optimizations exist for reducing the SQL statement memory requirements?" below in order to make sure that SQL statements are executed as memory-efficient as possible. If a high memory consumption is caused by the F4 value help mode D in BW environments (7.30 - 7.40), implement the BW correction available via SAP Note 2097025. Consider setting the hint NO_USE_OLAP_PLAN (SAP Note 2142945) for testing purposes in order to check if a switch from OLAP engine to join engine works and results in a reduced memory consumption. If the issue appears with a BW query, check if the problem improves using a different BW query execution mode. See BW on HANA and the Query Execution Mode for more information related to BW query execution modes. If the issues is linked to SAP ABAP queries using fast data access (FDA), you can |

| | | |
|---|---|---|
| | | consider deactivating it as described in SAP Note 2399993. If you face a high memory consumption related to DTP activities in BW, you can check SAP Note 2230080 for possible optimizations. |
| Pool/PersistenceManager/Backup | Sorted list of page numbers for data backups | This allocator is populated at the beginning of a data backup in order to have a sorted list of page numbers for backup streaming. Once the data backup is finished, the space is released. The size of the allocator can be particularly large in case of large databases and / or a high amount of disk LOBs (SAP Note 2220627). |
| Pool/PersistenceManager/Container FileIDMapping | LOB container mapping | This allocator maps LOB containers to the persistence files. If it is particularly large, the following reasons are possible: A large amount of LOB data (can e.g. be checked via SQL: "HANA_Tables_LargestTables" -> LOB_GB in SAP Note 1969700) Unnecessarily high amount of LOB containers, improved with Revision 94 |
| Pool/PersistenceManager/DisasterRecoveryPrimary | Asynchronous system replication buffer | The main contributor to the allocator is usually the asynchronous system replication buffer, so it only has a significant size of asynchronous system replication is used and it closely depends on the value of the related parameter: <service>.ini -> [system_replication] -> logshipping_async_buffer_size = <size_in_byte> If you have to increase this buffer, you should only do it for the services with a high redo log generation, typically the indexserver (<service>.ini = indexserver.ini). Setting this parameter in global.ini technically also works, but as a consequence the increased space is allocated multiple times (for all different services), and so memory is wasted. See SAP Note 1999880 for more information related to SAP HANA system replication. |
| Pool/PersistenceManager/DisasterRecoverySecondary | System replication related allocations on secondary site | The size of this cache is mainly linked to the setting of the following system replication parameter on secondary system replication site (SAP Note 1999880): global.ini -> [system_replication] -> logshipping_replay_push_persisten t_segment_count The default value of 20 relates to a size of 1 GB. See SAP Note 2409671 for more information related to this setting. |
| Pool/PersistenceManager/DisasterRecoverySecondary/ReplayLogCache | System replication log replay cache | This cache is used in system replication environments (SAP Note 1999880) with a log replay related operation mode like logreplay or |

| | | |
|---|---|---|
| | | logreplay_readaccess. It improves log replay performance by avoiding disk I/O. Per default 4 GB are used for the indexserver and 1 GB for other services. Normally no change to the default is required. In special situations (e.g. as workaround in SAP Note 2405763) the size can be adjusted with the following parameter: <service>.ini -> [system_replication] -> logshipping_replay_logbuffer_cach e_size = <size_in_byte> |
| Pool/PersistenceManager/LOBContai nerDirectory | Hybrid LOB directory | This allocator contains information about hybrid LOB values stored on disk (see SAP Note 2220627). Its size depends mainly on the amount of hybrid LOB values stored on disk. It can grow in case of problems with LOB garbage collection. See SAP Note 2169283 for more information related to garbage collection. |
| Pool/PersistenceManager/LogRecove ry | Log recovery | This allocator is used to buffer up to four log segments in memory during recovery. The configured log segment sizes can be checked with SQL: "HANA_Logs_LogBuffers" (SAP Note 1969700). In case of a 1 GB log segment size you have to expect a memory allocation of 4 GB during recovery. |
| Pool/PersistenceManager/Persisten tSpace/DefaultLPA/LOBPage | Disk LOB caching | While disk LOB pages (SAP Note 2220627) are cached in the generic allocator Pool/PersistenceManager/Persisten tSpace/DefaultLPA/Page with SAP HANA <= 2.0 SPS 02, they use the dedicated LOB allocator Pool/PersistenceManager/Persisten tSpace/DefaultLPA/LOBPage with SAP HANA >= 2.0 SPS 03. Populating and releasing this allocator follows the same rules like described for Pool/PersistenceManager/Persisten tSpace/DefaultLPA/Page. Starting with SAP HANA 2.0 SPS 04 you can configure the allocator size limit that will trigger a cleanup, preventing further growth: <service>.ini -> [persistence] -> lob_page_trigger_cleanup_threshol d = <size_in_byte> |
| Pool/PersistenceManager/Persisten tSpace(0)/DefaultLPA/Page Pool/PersistenceManager/Persisten tSpace/DefaultLPA/Page | SAP HANA page cache | The SAP HANA page cache stores blocks retrieved from disk similar to a file system cache. This can e.g. speed up the access to hybrid LOBs (SAP Note 1994962). You can check for the content of the allocator in terms of page types by executing "pageaccess a" with hdbcons (SAP Note 2222218). Due to a bug the size of this cache can be unnecessary large with Revisions 110 to 122.05. In order to avoid automatic reclaims with impact on the |

| | | |
|---|---|---|
| | | running system you can consider the following proactive measures (see SAP Note 2301382): Rev. 110 - 122.01: regular "resman shrink" scheduling Rev. 122.02 - 122.05: unload_upper_bound configuration The page allocator size can be significant after a recovery, e.g. during a near zero downtime upgrade using SAP HANA system replication or a takeover (SAP Note 1999880). See SAP Note 2427897 for more details. With SAP HANA 1.0 SPS 12 >= 1.00.122.06 it is recommended to disable caching for main (global.ini -> [persistence] -> internal_caching_for_main = false, SAP Note 2600030) in order to reduce the utilization of the page cache. The pages cache also buffers history files and so garbage collection issues resulting in a higher amount of history files can implicitly result in an allocator growth. Related allocator stack modules can be e.g. DataAccess::GarbageCollectorJob:: run. See SAP Note 2169283 for more information related to SAP HANA garbage collection. Delta merges (SAP Note 2057046) temporarily write the data of the new main storage into the page allocator, so during delta merges of large tables the allocator size can significantly grow. The allocated space will be freed when the delta merge is finished. You can consider partitioning large tables (SAP Note 2044468) in order to reduce the temporary space overhead. If LOB garbage collection doesn't take place properly (SAP Note 2169283), LOB related pages aren't purged in time and so there can be an increased growth and size of the page allocator. If the above scenarios don't apply and you see unloads or OOMs at a time where this allocator is still large, it is likely that the disk I/O peformance is not able to keep up with the data changes. In this case you should check your I/O stack for bottlenecks. See SAP Note 1999930 for more information. The page cache also contains the paged attributes cache (SAP Note 1871386). If you use paged attributes, you should check the following details: Check the size parameters described in SAP Note 2111649 and consider that the configured size contributes to the page cache size. Due to the bug described in SAP Note 2497016 it can happen that inverted index related pages are pinned in memory and more space is used for the paged attributes cache |

| | | |
|---|---|---|
| | | than intended (SAP HANA <= 1.00.122.09, <= 2.00.002.00, <= 2.00.011). |
| Pool/PersistenceManager/PersistentSpace(0)/DefaultLPA/ShadowPage Pool/PersistenceManager/PersistentSpace/DefaultLPA/ShadowPage | I/O flush shadow pages | In some scenarios a copy of a page has to be created before the flush thread can write it down to disk: Critical savepoint phase Encrypted page Row store page The page is deallocated as soon as the I/O write was successfully finished and acknowledged. A large size of this allocator can indicate a high I/O write volume or that the I/O stack isn't able to keep up with the flush thread activities. See SAP Note 1999930 for more information regarding I/O analysis. |
| Pool/PersistenceManager/PersistentSpace(0)/PageChunk Pool/PersistenceManager/PersistentSpace/PageChunk | Clustering of small pages for write | This allocator is used to combine several smaller pages in chunks before writing them to disk. Once the write has finished, the allocated space is released. A temporary large allocator size is usually a consequence of disk I/O bottlenecks. See SAP Note 1999930 and eliminate bottlenecks in the I/O stack. |
| Pool/PersistenceManager/PersistentSpace(0)/RowStoreLPA Pool/PersistenceManager/PersistentSpace/RowStoreLPA | Row store control blocks | This allocator contains row store control blocks and can grow significantly in case of a large row store. See SAP Note 2222277 and make sure to keep the row store at a reasonable size, e.g. via cleanup (SAP Note 2388483) or defragmentation (SAP Note 1813245). |
| Pool/PersistenceManager/PersistentSpace(0)/RowStoreLPA/RowStoreSegment Pool/PersistenceManager/PersistentSpace/RowStoreLPA/RowStoreSegment | Row store cache (system replication) | This allocator caches row store blocks on the secondary site of a system replication scenario, so that the row store can be created efficiently during failover. Its size is related to the row store size on the primary system. If the secondary site runs into OOM because of this allocator, you have the following options: Primary system: Move large row store table to column store Primary system: Reduce data volume in large row store tables (e.g. via SAP Note 2388483) Secondary system: Increase the global allocation limit sufficiently |
| Pool/PersistenceManager/PersistentSpace(0)/RowStoreLPA/Superblock Pool/PersistenceManager/PersistentSpace/RowStoreLPA/Superblock | Row store superblocks | This allocator is used during SAP HANA startup to store the row store superblocks (including row store data) in heap while populating the row store shared memory. After startup its size is typically 0, but on secondary system replication sites it can remain with a large size for a longer time (with short term disposition). See SAP Note 2222277 and make sure to keep the row store at a reasonable size, e.g. via cleanup (SAP Note 2388483) or defragmentation (SAP Note 1813245). If the large size of the allocator causes trouble (e.g. OOM during startup), you can temporarily |

| | | |
|---|---|---|
| | | disable the optimized row store load with the following parameter (SAP Note 2612205): indexserver.ini -> [persistence] -> optimized_rowstore_load = false |
| Pool/PersistenceManager/PersistentSpace(0)/StaticLPA/Page<br>Pool/PersistenceManager/PersistentSpace/StaticLPA/Page | liveCache pages | This area contains the page cache related to liveCache (if operated as part of SAP HANA). Up to SPS 08 these pages aren't swappable. Starting with SPS 09 the space is reclaimed automatically by SAP HANA whenever memory is required, so a large size is not critical. See SAP Note 2593571 for more information related to liveCache. |
| Pool/PersistenceManager/PersistentSpace/TempLPA/Page | Temporary table data | This heap allocator stores data of temporary tables. It can significantly grow during repartitioning activities (SAP Note 2044468). |
| Pool/PersistenceManager/UndoDirectory | Undo and cleanup file directory | This allocator contains undo and cleanup file information and can grow significantly if persistence garbage collection is blocked. See SAP Note 2169283 for more information related to garbage collection and take appropriate actions to resolve garbage collection issues. |
| Pool/PersistenceManager/UnifiedTable container<br>Pool/PersistenceManager/UnifiedTableContainer | L2 delta and paged attribute information | This allocator contains persistence information related to the new delta mechanism used as of SPS 09 (L2 delta) and paged attributes (SAP Note 1871386). Up to SPS 08 delta logs were stored in virtual files instead. The actual delta area in column store remains untouched from this allocator. See SAP Note 2057046 and make sure that delta merges are properly configured and executed, so that the delta storage size of the tables remains on acceptable levels. The allocator can grow in cases when persistence garbage collection is blocked (SAP Note 2169283). |
| Pool/PersistenceManager/VirtualFile entry TID map | Transient mapping for VirtualFile overwrite optimization | This allocator typically grows in the context of disk LOBs (SAP Note 2220627): With SAP HANA SPS <= 08 it can consume significant amounts of space if a lot of small disk LOBs are inserted It can also grow temporarily if tables with many disk LOBs are converted (e.g. for adusting the LOB MEMORY THRESHOLD limit). |
| Pool/PlanningEngine/Compile | Planning engine compilation structures | If the allocator size is large in context of planning engine activities, you can check if dropping no longer required planning sessions can help to reduce the allocations (SAP Note 2169283 -> "How can garbage collection be triggered manually?" -> "Planning engine garbage collection"). |

| | | |
|---|---|---|
| Pool/PlanningEngine/Fox | Dictionaries for FOX formula executions | FOX formula executions by the planning engine may require temporary helper structures that are allocated in Pool/PlanningEngine/Fox. They are dropped after the FOX planning function is finished. If a significant memory allocation - often in combination with Pool/itab - is seen, there may be a loop in the FOX script that has to be corrected. Additionally you can check if dropping no longer required planning sessions can help to reduce the allocations (SAP Note 2169283 -> "How can garbage collection be triggered manually?" -> "Planning engine garbage collection"). |
| Pool/PlanningEngine/LookupDict | Master data lookup dictionary of planning engine | You can use SQL: "HANA_Heap_PlanningEngine" (OBJECT_TYPE = 'LOOKUP DICTIONARY') available via SAP Note 1969700 in order to check for the main contributors and the creation times. After a restart of SAP HANA this allocator is empty and re-populated on demand. You can use SQL: "HANA_Heap_PlanningEngine_Cleanup " (SAP Note 1969700) in order to drop no longer required runtime objects. Starting with SPS 10 SAP HANA automatically takes care for the cleanup. If these steps don't help you can check if dropping no longer required planning sessions can help to reduce the allocations (SAP Note 2169283 -> "How can garbage collection be triggered manually?" -> "Planning engine garbage collection"). |
| Pool/planviz/column store/plans Pool/planviz/column store/plans/ParentCycleDetector Pool/planviz/column store/PlanVizContext Pool/planviz/column store/PlanVizContext/JsonAllocato r Pool/planviz/common/final results Pool/planviz/common/strings Pool/planviz/sql layer/PlanVizContext Pool/planviz/sql layer/PlanVizContext/PlanVizParam s | PlanViz details | These allocators are used by PlanViz and the plan trace (see SAP Note 2119087). In order to keep their sizes at a reasonable level you should use the plan trace only as restricted as possible (in terms of time and traced statements). SAP Note Impacted Revisions Details 2405237 <= 112.06 120 - 122.02 Due to a bug with SAP HANA SPS 10 to SPS 12 it can happen that these allocators continue to grow even after you have disabled the trace. In this case you have to restart in order to clear the allocators and avoid further growth. <= 122.06 A similar bug (continous growth of allocators after tracing) still exists up to Rev. 122.06. It is fixed with Rev. 122.07. Even with higher SAP HANA Revisions these allocators can grow although the plan trace was deactivated, so in production systems you should make sure that plan trace isn't used. |
| Pool/QueryMediator | Processing of | See question "Which general optimizations |

| | | |
|---|---|---|
| | complex filters | exist for reducing the SQL statement memory requirements?" below in order to make sure that SQL statements are executed as memory-efficient as possible. Related queries typically have query mediator related modules in their call stacks, e.g.: QueryMediator::FilterProcessor::a ddFilterAsExpression QueryMediator::FilterTranslation: :SearchOperation Starting with SAP HANA SPS 10 an optimization is implemented so that the problem is fixed in many cases. As a workaround you can check if specifying the hint NO_CS_ITAB_IN_SUBQUERY helps to reduce the memory consumption. See SAP Note 2142945 for more information related to SAP HANA hints. |
| Pool/ResourceContainer Pool/ResourceContainer/ResourceHe ader | Metadata for memory objects | A large size of these allocators is typically caused by a high number of memory objects. You can use SQL: "HANA_Memory_MemoryObjects" (SAP Note 1969700) or directly query M_MEMORY_OBJECTS in order to check for the number of existing memory objects. Pool/ResourceContainer/ResourceHe ader contains resource headers that are never destroyed, so it will not reduce in size over time until SAP HANA is restarted. |
| Pool/ResultCache(for cached view) | Static result cache information | This allocator stores static result cache information information with SAP HANA SPS 12 and higher. The static result cache is available as of SAP HANA SPS 11. See SAP Note 2336344 for more information related to the SAP HANA static result cache cache. |
| Pool/RowEngine/Communication | TCP/IP communication channel management | See SAP Note 2222200 and try to reduce the amount of connections and inter-node / inter-service communications in order to reduce the size of this allocator. |
| Pool/RowEngine/CpbTree Pool/RowStoreTables/CpbTree | Row store indexes | Check via SQL: "HANA_RowStore_TotalIndexSize" (SAP Note 1969700) if the size of the heap allocator is in line with the size of the row store indexes. If it is significantly larger, most likely a memory leak exists that can only be cleaned up by restarting SAP HANA. Upgrade to at least revision 83 in order to eliminate known memory leaks. Due to a bug with Rev. <= 85.03 and Rev. 90 to 94 index garbage collection is not necessarily triggered in time and so this allocator can unnecessarily grow. With Rev. 85.04 and Rev. 95 a fix is delivered. |

| | | |
|---|---|---|
| | | See SAP Note 2169283 for more information related to garbage collection. If the allocator size is in line with the index sizes, check if there are large tables with indexes in row store that can be cleaned (e.g. via SAP Note 2388483) or moved to column store. Check via SQL: "HANA_Indexes_Overview" (ORDER_BY = 'SIZE', SAP Note 1969700) if there are large indexes created on row store tables that are not required and can be dropped. |
| Pool/RowEngine/GlobalHeap | Global, unspecific row engine data areas | This allocator is an unspecific allocator for row engine memory. As all significant memory allocation should be assigned to dedicated allocators, Pool/RowEngine/GlobalHeap shouldn't allocate too much memory. Reasons for increased sizes are: SAP Note Impacted Revisions Details < 90 Processing of a very high number of rows with STRING_AGG function can result in an increased allocator size. 2371445 <= 122.02 A memory leak can be responsible for a rising allocator size. If you face another situation with a significant memory allocation in Pool/RowEngine/GlobalHeap, you can open a SAP incident for clarification. |
| Pool/RowEngine/IndexRebuild | Row store index rebuild structures | This heap allocator is used when row store indexes are rebuilt, typically during / after SAP HANA restarts (SAP Note 2177064). |
| Pool/RowEngine/LOB | LOB data processed by database requests | See question "Which general optimizations exist for reducing the SQL statement memory requirements?" below in order to make sure that SQL statements are executed as memory-efficient as possible. This allocator stores small LOB values (default: <= 1024 byte) that are processed by database requests. Larger LOB values are stored in Pool/RowEngine/QueryExecution. If the Pool/RowEngine/LOB allocator is large, you should check for SQL statements requesting a large amount of records with LOB columns. |
| Pool/RowEngine/LockTable<br>Pool/RowStoreTables/LockTable | Row store lock and version information | A large size of this allocator can indicate a high number of transactional locks (SAP Note 1999998) or garbage collection issues (SAP Note 2169283). Additionally you can check the following known SAP HANA bugs resulting in increased sizes of this allocator: SAP Note Impacted Revisions Details 2391552 110 - 122.04 Missing cleanup of Pool/RowStoreTables/LockTable during garbage collection |
| Pool/RowEngine/MonitorView | Monitoring view | This heap allocator contains information of in- |

| | | |
|---|---|---|
| | information | memory monitoring views (M_* views). You can display the largest areas within Pool/RowEngine/MonitorView using "mm bl" with hdbcons (SAP Note 2222218) as described further below. In general you have to make sure that less data is collected in the critical monitoring views (e.g. by reducing the trace level). Known issues are: Monitor views Impacted revisions SAP Note Details M_CS_ALL_COLUMNS M_CS_COLUMNS M_CS_TABLES M_FUZZY_SEARCH_INDEXES M_TABLES >= 102.00 >= 110 120 - 121 2343177 Accesses to these monitoring views can result in memory leaks if certain conditions like scale-out and a high number of records are fulfilled. M_CS_LOADS M_CS_UNLOADS 120 - 121 2340582 The load and unload trace can be responsible for a memory leak. M_EXPENSIVE_STATEMENTS 70 - 85.00 2112732 If most space is allocated by "ptime::ExpensiveStatementMonitor ::create_objects_ringBuffer" at ExpensiveStatementsMonitor.cc the problem is caused by the expensive statement trace. You can reduce the allocation by increasing the trace limit or deactivating the expensive statements trace (SAP Note 2180165). In SPS 07 and SPS 08 a memory leak exists which can be eliminated by deactivating in-memory tracing using the following parameter: global.ini -> [expensive_statement] -> use_in_memory_tracing = false M_SQL_PLAN_CACHE_FOR_STATISTICSSERVER_RESET_ < 100 2186299 The SQL cache history collection of the embedded statistics server can result in a high memory demand. M_TABLE_LOB_FILES Queries on can consume significant amount of this memory in ptime::TableLobFilesMonitor::createLobEntry if many hybrid LOBs exist. Starting with SAP HANA SPS 12 you can consider using M_TABLE_LOB_STATISTICS as a light-weight variant for M_TABLE_LOB_FILES that doesn't show the high memory requirements. |
| Pool/RowEngine/QueryCompilation | Compilation memory | This allocator is required during parsing of database queries. Large sizes can be caused by complex SQL statements. Additionally you can check the following known SAP HANA issues resulting in increased sizes of this allocator: SAP Note Details 2124112 For parsing particularly large SQL statements it |

| | | |
|---|---|---|
| | | can be required to increase the value of this parameter: indexserver.ini -> [sql] -> default_segment_size = <segment_size_in_byte> As a consequence the size of the Pool/RowEngine/QueryCompilation parameter can increase. 2453348 The size of this allocator can grow in the context of an activated plan trace. A memory leak in context of call stack module AnalyticalAuthorization::FilterPr ovider::getFilterAsQoStructure is a consequence of a SAP HANA bug fixed with Rev. 1.00.122.09. A growth of this allocator was observed in the context of terminations that show up in the database trace (SAP Note 2380176) in the following way: Error during Plan execution of model... This behavior can be considered as a memory leak bug. You can analyze and reduce the terminations as a workaround until a fix is available. 2124112 Reducing the complexity of parsing and estimations ("Which problems and solutions exist in the area of parsing?" -> "High sampling overhead") can have a positive impact on the size of Pool/RowEngine/QueryCompilation. |
| Pool/RowEngine/QueryExecution Pool/RowEngine/QueryExecution/Sea rchAlloc | Row engine results | See question "Which general optimizations exist for reducing the SQL statement memory requirements?" below in order to make sure that SQL statements are executed as memory-efficient as possible. Additionally you can check the following known SAP HANA bugs resulting in increased sizes of this allocator: SAP Note Impacted Revisions Details 2000792 67 - 69.00 70 ORDER BY with parallelized sub plan 2271235 102.01 - 102.04 110 Batch INSERTs on row store table 2527251 1.00.112.07 1.00.122.06 - 1.00.122.08 2.00.000 Calculation view with analytic privilege check |
| Pool/RowEngine/RowTableManager/MV CCManager/MVCCAllocator | Row store MVCC management during recovery | This heap allocator is used for row store MVCC management during recovery operations (e.g. on secondary system replication sites and during database recoveries). It is available starting with SAP HANA 2.0 and it can increase in case of garbage collection issues (SAP Note 2169283). The following SAP HANA bugs exist that can result in an increased allocator size: SAP Note Impacted Revisions Details 2573738 2.00.021 - 2.00.022 Missing row store garbage collection on secondary system replication site |

| | | |
|---|---|---|
| | | (SAP Note 1999880) with operation mode logreplay |
| Pool/RowEngine/RSTempPage | Temporary row store tables | This allocator holds data related to temporary tables and NO LOGGING in row store. Check why many or large temporary row store tables exist and try to reduce it. Make sure that sessions are closed whenever possible, because this will drop related temporary tables. See SAP Note 2000003 ("What kind of temporary and non-persisted tables can be created with SAP HANA?") for more information related to temporary and NO LOGGING tables. Additionally you can check the following known SAP HANA bugs resulting in increased sizes of this allocator: SAP Note Impacted Revisions Details 2368929 <= 112.06 Memory leak if temporary row tables without variable length columns are used 2402318 120 - 122.02 Memory leak when temporary row tables are dropped |
| Pool/RowEngine/Session | Session management | Check if there is an unusual high number of open connections and eliminate the root cause. |
| Pool/RowEngine/SQLPlan | SQL cache | The SQL cache can be configured unnecessarily large because underlying issues like a lack of bind variables or varying IN LIST sizes are not recognized. See SAP Note 2124112 and make sure that the SQL cache is not configured larger than required. Be aware that the heap allocator size can be up to three times larger than the size (in byte) configured with the following SAP HANA parameter: <service>.ini -> [sql] -> plan_cache_size Reason: In addition to the SQL plan cache itself, this allocator includes all other miscellaneous memory allocations such as data structures for managing SQL plan cache, monitoring view data and optimizer allocations (SAP Note 2502256). Due to a SAP HANA bug on SAP HANA <= 1.00.122.13, <= 2.00.012.03 and <= 2.00.022 certain internal SQL cache statistics weren't considered for the plan cache size calculation and so the heap allocator could grow more than expected. |
| Pool/RowEngine/TableRuntimeData | Table runtime data | With SAP HANA Rev. <= 97.01 no proper cleanup happens when a temporary table is dropped, this bug is fixed as of Rev. 97.02. |
| Pool/RowEngine/Version Pool/RowStoreTables/Version | Row store version space | A high number of versions may need to be preserved for read consistency (MVCC) reasons in case of a long running transaction. This increases the size of this allocator. See |

| | | |
|---|---|---|
| | | "Which options exist to reduce the risk of SAP HANA memory issues?" -> "Transactional problems" in this SAP Note for more detailed recommendations. |
| Pool/RowEngine/ViewCache | Static result cache information | This allocator stores static result cache information with SAP HANA SPS 11. The static result cache is available as of SAP HANA SPS 11. See SAP Note 2336344 for more information related to the SAP HANA static result cache. |
| Pool/RowTableUpdateAllocator | Row table update information | This allocator is used in context of updating row store tables. Due to a SAP HANA bug it can happen that memory isn't released in time. As a workaround you can clear the SQL cache: ALTER SYSTEM CLEAR SQL PLAN CACHE The bug is fixed with SAP HANA >= 122.10, 2.00.002.01 and 2.00.010. |
| Pool/SearchAPI Pool/SearchAPI/Itab Search | Intermediate results | See question "Which general optimizations exist for reducing the SQL statement memory requirements?" below in order to make sure that SQL statements are executed as memory-efficient as possible. This allocator is more intensively used with SAP HANA Rev. 122.05. It works in a similar way like Pool/itab, so you can check that allocator for more information. Among others it is used by the hierarchy cache, see allocator Pool/hierarchyBlob for more details. |
| Pool/Search/PreparedQuery | Prepared searches | This allocator is used for searches related to prepared statements. |
| Pool/SerializedObject | Fulltext index data structures | You can run SQL: "HANA_Indexes_Overview" (INDEX_TYPE = 'FULLTEXT', ORDER_BY = 'SIZE') available via SAP Note 1969700 to display the existing fulltext indexes sorted by size. Check if particularly large fulltext indexes are really required. For example, a large index REPOSRC~SRC may exist to support the ABAP Sourcecode Search (SAP Note 1918229) and can be removed via transaction SFW5. |
| Pool/SingleValueCacheBuilder | Single value cache | This heap allocator is used for the single value cache (SAP Note 2502256). |
| Pool/spatialcs | Spatial data | This allocator is linked to the Spatials option (SAP Note 2091935) and it is typically required for spatial joins, spatial clustering and for providing metadata for geometry attributes. A memory leak can be responsible for a permanent growth on SAP HANA Revisions between 100 and 122.02. A fix is available with Revision 122.03. |

| | | |
|---|---|---|
| Pool/SQLScript/Execution | SQL Script runtime information | Check for design problems in the used SQL Script procedures. If you face a high memory consumption with Rev. 100 or 101, a bug can be responsible (SAP Note 2312948) in context of XS engine calls. Upgrade to Rev. 102 or higher in order to fix it. |
| Pool/Statistics | Internal statistical information | You can display the largest areas within Pool/Statistics using "mm bl" with hdbcons (SAP Note 2222218). Example 1: (most space consumed by allocators) 44 GB: MemoryManager::PoolAllocator::Poo lAllocator (libhdbbasis.so) 14 GB: MemoryManager::MemoryCounter::Mem oryCounter (libhdbbasis.so) 12 GB: Execution::ContextAllocator::init ImplicitStatementMemoryBooking (libhdbbasis.so) 12 GB: ltt::allocator_statistics::setCom positeLimit (libhdbbasis.so) Example 2: (most space consumed by read write locks) 14 GB: Synchronization::ReadWriteLock::R eadWriteLock (libhdbbasis.so) 3 GB: Synchronization::FastReadSlowWrit eLock::allocateReaderItems (libhdbbasis.so) The size of the - usually dominant - modules mentioned in "Example 1" above mainly depends on the following factors: Number of records in M_CONTEXT_MEMORY (which is closely linked to the number of SQL connections) Number of records in M_HEAP_MEMORY Number of (logical) CPUs Activation of special features like memory tracking or statement memory limit Example: Constellation Pool/Statistics size Rough calculation formula 578 CPUs 4.5 million entries in M_CONTEXT_MEMORY memory tracking and statement memory limit 300 GB 578 * 4500000 * 96 * 1.3 (statement memory limit factor) 180 CPUs 3.6 million entries in M_CONTEXT_MEMORY memory tracking and statement memory limit 75 GB 180 * 3600000 * 96 * 1.3 (statement memory limit factor) The factor of 96 byte in the calculation is a worst case estimation - depending on different factors it can vary between 64 and 96 byte in different environments. If there are many records in M_HEAP_MEMORY (> 100000), you can check for the most frequent heap allocators using SQL: "HANA_Memory_TopConsumers" (AREA = 'HEAP', ORDER_BY = 'NUM') of SAP Note 1969700. The following individual reasons for a |

| | | |
|---|---|---|
| | | high Pool/Statistics allocation exist: The amount of entries in M_CONTEXT_MEMORY among others depends on the amount of database requests cached on client side. In ABAP environments this is controlled by parameter dbs/hdb/stmt_cache_size (default: 1000 statements per connection). You can reduce it in order to minimize the M_CONTEXT_MEMORY entries and the Pool/Statistics size (SAP Note 2532199). In a real-life scenario the size of Pool/Statistics reduced by factor 4 after having reduced the value from 1000 to 100. Be aware that a reduction of this setting can increase the parsing activities, so you should monitor the performance effects. See SAP Note 2124112 ("Can there be also statement caches on client side?") for more information related to the ABAP client statement cache. A very high number of entries for Pool/RowEngine/QueryCompilation/SqlOptAlloc can be caused by a memory leak in Revisions up to 83. Large allocations for Synchronization::ReadWriteLock::ReadWriteLock can be caused by a memory leak with Rev. 100 to 101 which is fixed as of Rev. 102. With SAP HANA Rev. <= 97.01 no proper cleanup happens when a temporary table is dropped, this bug is fixed as of Rev. 97.02. SAP HANA Revisions 100 - 102.03 and 110 - 112 can suffer from a memory leak in the context of XS engine calls (SAP Note 2313013) Starting with SAP HANA 1.00.122.13 and 2.00.010 read write locks are tracked in the dedicated allocator Pool/FRSWLockAllocator and no longer in Pool/Statistics. See "Pool/FRSWLockAllocator" for known issues in context of read write lock statistics. |
| Pool/StatisticsServer/ThreadManager/Stats::Thread_<num><br>Pool/StatisticsServer/JobManager/Stats::Thread_<num><br>Pool/StatisticsServer/JobManager/WriteLastValuesJob<br>Pool/StatisticsServer/LastValuesHolder | Standalone statistics server | These allocators can become quite large if the standalone statistics server is used and a significant amount of monitoring data is available (e.g. large SQL plan cache, many connections). In order to optimize these allocators please proceed as described at "Which options exist to reduce the risk of SAP HANA memory issues?" -> "Statisticsserver optimizations" above. |
| Pool/StringContainer | Storage of (uncompressed) strings during column store | See question "Which general optimizations exist for reducing the SQL statement memory requirements?" below in order to make sure that SQL statements are executed as memory- |

| | | |
|---|---|---|
| | activities | efficient as possible. A temporary increase of Pool/StringContainer is possible during processing of large amounts of data, e.g.: Data load Index creation (improved with SAP HANA >= 1.00.122.11 and >= 2.00.012.01) Merge When the utilization reduces again after the large operation, it is normally not critical. |
| Pool/ChannelUtils/SynchronousPool CopyHandler | Multistream channel copy | This allocator is used in of backup and restore. The allocated size is linked to the buffer size and the I/O stream parallelism: Number of parallel I/O streams (default: 1): global.ini -> [backup] -> parallel_data_backup_backint_chan nels Buffer size (default: 512 MB): global.ini -> [backup] -> data_backup_buffer_size The following rules for the allocator size apply (with #services = number of SAP HANA services that are backed up): File backup: 2 * data_backup_buffer_size * #services Backint backup: (1 + parallel_data_backup_backint_chan nels) * data_backup_buffer_size * #services Attention: Reducing the buffer size can have a negative impact on backup runtimes. |
| Pool/TableConsistencyCheck | Table consistency check | This allocator is related to the consistency check procedure CHECK_TABLE_CONSISTENCY (see SAP Note 1977584). You can limit the number of concurrent executions on different tables or run it at times with less concurrent workload in order to reduce the risk of critical memory allocations. |
| Pool/Text/AEText Pool/Text/AEText/phrase_index Pool/Text/AEText/split_document_i ndex Pool/Text/AEText/split_positional _index Pool/Text/AEText/termset_containe r Pool/Text/AEText/text_property_in dex | Fulltext index data structures | These allocators store specific parts of the data of fulltext indexes, so their size mainly depends on the size of existing fulltext indexes. See SAP Note 2160391 for SAP HANA indexes in general and fulltext indexes in particular. |
| Pool/TransientMetadataAlloc | Transient metadata | This allocator stores temporary metadata information (object definitions; local on transaction / session level or global). The life time of some data is linked to the SQL cache, so you should check if this cache is defined larger than required (see SAP Note 2124112). The following individual reason afor a large allocator exist: SAP HANA Revisions 100 to 102.04 and 110 to 112.01 can suffer from a memory leak bug when a procedure is called (SAP Note 2312994). |

| | | |
|---|---|---|
| Pool/UdivListMgr/UdivListContainer | MVCC management | This allocator is responsible for managing multi-version concurrency control (MVCC), so the visibility of rows in different transactions. In order to check for problems like long-running transactions you can proceed as described in "Which options exist to optimize the SAP HANA memory utilization?" -> "Transactional problems". |
| Pool/ValueArray<br>Pool/ValueArrayColumnDeserialize | Join engine results | See question "Which general optimizations exist for reducing the SQL statement memory requirements?" below in order to make sure that SQL statements are executed as memory-efficient as possible. These allocators are closely linked to Pool/JoinEvaluator/JERequestedAttributes/Results: Pool/ValueArray was a previous name that is no longer used with current Revisions. Pool/ValueArrayColumnDeserialize is used when join engine results have to be sent from one node to another in scale-out scenarios. |
| Pool/XDictData | Intermediate OLAP engine dictionary data | See question "Which general optimizations exist for reducing the SQL statement memory requirements?" below in order to make sure that SQL statements are executed as memory-efficient as possible. This allocator stores dictionary data during query processing in the OLAP engine. It can grow significantly if columns with many distinct values are processed and filters don't exist or aren't pushed down. |
| Pool/XSEngine/XSJobScheduler/_<application><br>Pool/XSEngine/XSJobScheduler/_<application>/_<job_name> | XS engine jobs | These heap allocators contain information related to XS engine jobs. Every job creates an own instantiation of the allocator, so the number of allocator instantiations can be particularly high. The size is at the same time typically at a reasonable level. Starting with SAP HANA 1.00.122.16, 2.00.012.05 and 2.00.024 jobs with the same name use the same allocator and so the amount of allocator instantiations can be significantly lower. There is no way to purge old jobs from the allocator. After a SAP HANA restart the allocator starts again from scratch. |
| StackAllocator | Thread stacks | This allocator contains thread stack data. Large sizes are typically caused by a large number of threads or be large settings of the following stack size parameters: global.ini -> [threads] -> default_stack_size_kb global.ini -> [threads] -> worker_stack_size_kb These parameters should normally remain on default |

| | | |
|---|---|---|
| | | values, adjustments are only required in specific situations (e.g. SAP Note 1847202). |
| VirtualAlloc | SAP HANA external memory management | This allocator is related to memory allocations outside of the standard SAP HANA memory management, e.g. related to Java Virtual Machine (JVM). |

### 14. How can I identify how a particular heap allocator is populated?

You can use the tool hdbcons on operating system level in order to understand better how a heap allocator is filled (SAP Note 2222218). Typical commands are:

| Command | Example | Purpose |
|---|---|---|
| help mm | | Overview of all memory management (mm) related command options |
| mm bl -t <allocator> | mm bl -t Pool/Statistics | Show top memory contributors ("block list") in <allocator> sorted by used size descending |
| mm cg -o <file>.dot <allocator> | mm cg -o callgraph.dot Pool/Statistics | Generate output file with allocator stack trace information for <allocator> |
| mm f <allocator> as | mm f Pool/Statistics as | Activation of allocator call stack trace for <allocator> Particularly useful in case of suspected memory leaks so that you can understand from which modules the memory allocations are mainly performed Can result in significant overhead and should only be activated for limited times |
| mm f <allocator> as -d | mm f Pool/Statistics as -d | Deactivation of allocator call stack trace for <allocator> |
| mm ru | mm ru | Reset all previous measurements ("reset usage") |
| mm top -l <num> <allocator> | mm top -l 20 Pool/Statistics | Generate report with top <num> call stacks recorded for <allocator> |
| pageaccessa | | Provide breakdown of Pool/PersistenceManager/PersistentSpace(0)/DefaultLPA/Page content based on page type, e.g.: ConvIdxPage 256k Temp : 1 (262144 Byte) ConvLeafPage 256k Temp : 130 (34078720 Byte) TidCidMappingPage 256k Short : 1 (262144 Byte) FileIDMappingPage 256k Temp : 172 (45088768 Byte) FileIDMappingPage 256k Short : 2 (524288 Byte) ContainerDirectoryPage 256k Short : 35 (9175040 Byte) ContainerDirectoryPage 256k Long : 2 (524288 Byte) ContainerNameDirectoryPage 256k Long : 1 (262144 Byte) UndoFilePage 64k Short : 707 (46333952 Byte) VirtualFilePage 4k InternalShort : 134 (548864 Byte) VirtualFilePage 16k InternalShort : 57 (933888 Byte) VirtualFilePage 64k InternalShort : 325 (21299200 Byte) VirtualFilePage 256k InternalShort : 196 (51380224 Byte) VirtualFilePage 1M InternalShort : 552 (578813952 Byte) VirtualFilePage 4M InternalShort : 2832 (11878268928 Byte) VirtualFilePage 16M InternalShort : 9458 (158678908928 Byte) VarSizeEntryBasePage 256k Short : 809 (212074496 Byte) ... |

Example 1 (check for top memory contributors in allocator):

```
mm bl -t Pool/RowEngine/MonitorView
```

```
1256625660800 (392695519 blocks) in use at: Dumping frame 0x00007fcc318ad0d0:
1: 0x00007fcc318ad0d0 in ptime::ExpensiveStatementsMonitor::create_objects_ringBuffer at ExpensiveStatementsMonitor.cc

1479495168b (963213 blocks) in use at: Dumping frame 0x00007fcc32736290:
1: 0x00007fcc32736290 in ptime::Statement::Statement at handle_ref.hpp

6918144b (9008 blocks) in use at: Dumping frame 0x00007fcc32cb3070:
1: 0x00007fcc32cb3070 in ptime::Transaction::postcommit() at handle_ref.hpp

4249600b (3320 blocks) in use at: Dumping frame 0x00007fcc326d4de0:
1: 0x00007fcc326d4de0 in ptime::Connection::prepareStatistics_() at handle_ref.hpp
```

This output indicates that more than 100 GB of allocator Pool/RowEngine/MonitorView is consumed by the ExpensiveStatementsMonitor and so optimizations like adjustments to the expensive statements trace or implementing a bugfix to resolve a memory leak problem can be considered.

Example 2 (create an allocator call stack trace and extract top 5 call stacks)

```
mm ru mm f Pool/Statistics as -- Now wait until a representative amount of allocations is
captured mm top -l 5 Pool/Statistics mm ru mm f Pool/Statistics as -d
```

### 15. How often are OOM dumps written?

In case of OOM situations SAP HANA may write a dump, e.g.:

- <service>_<host>.<port>.**rtedump**.<timestamp>.**oom**.trc
- <service>_<host>.<port>.**rtedump**.<timestamp>.**after_oom_cleanup**.trc
- <service>_<host>.<port>.**rtedump**.<timestamp>.**compositelimit_oom**.trc
- <service>_<host>.<port>.**rtedump**.<timestamp>.**oom_memory_release**.trc

For more details about the different dump types see SAP Note 2000003 ("Which types of dumps can be created in SAP HANA environments?").

Not every OOM termination results in an OOM dump because in case of a memory bottleneck many different transactions can run into an OOM error within a short time frame. Per default a SAP HANA service only creates an OOM dump if the last dump was written at least one day ago. This behaviour can sometimes be of disadvantage when two individual OOM situations should be analyzed that happened within less than 24 hours.

In special cases you can reduce the minimum time between two OOM dumps using the following SAP HANA parameter:

```
global.ini -> [memorymanager] -> oom_dump_time_delta = <min_seconds_between_oom_dumps>
```

If you set the parameter for example to 7200, the minimum interval between two OOM dumps will be two hours (7200 seconds).

### 16. Where can I find more information regarding SAP HANA memory consumption?

The document SAP HANA Memory Usage Explained provides a good overview of different types of memory in SAP HANA environments.

### 17. How can the resident memory be smaller than the allocated memory?

Normally the allocated memory should be fully contained in the resident memory, nevertheless there are a few exceptions:

- If parts of the virtual memory are paged out to disk, the resident memory can be smaller than the allocated memory.
- There are technical constellations where parts of the heap memory and the row store shared memory

are marked as used, but not as resident.

## 18. What are typical reasons for significant size differences in memory vs. on disk?

The allocation of tables in memory and on disk may significantly differ for the following reasons:

| Reason | Symptom | Details |
|---|---|---|
| No logging tables | Memory > disk | Tables created with the NO LOGGING option are not persisted to disk. See SAP Note 2000003 ("What kind of temporary and non-persisted tables can be created with SAP HANA?") for more information. |
| Temporary tables | Memory > disk | Tables created with the TEMPORARY option are not persisted to disk. See SAP Note 2000003 ("What kind of temporary and non-persisted tables can be created with SAP HANA?") for more information. |
| Single column and row store indexes | Memory > disk | Single column indexes and row store indexes aren't persisted to disk. See SAP Note 2160391 ("Are indexes persisted to disk?") for more information. |
| Logically moved tables | Memory > disk | If tables are moved logically, their disk footprint can be significantly smaller than the size in memory. See SAP Note 1994408 for more information. |
| Hybrid LOBs | Disk > memory | Large hybrid LOB values (SAP Note 1994962) are not loaded into memory, so the disk size of tables is larger than the memory size. |
| Partially loaded tables | Disk > memory | If columns of a table are only partially loaded into the memory, the disk size is larger than the current memory size. You can use SQL: "HANA_Tables_LargestTables" (SAP Note 1969700) to check disk size, potential maximum memory size and current memory size. |
| Data fragmentation | Disk > memory | A fragmented data area can significantly increase the disk requirements. You can use SQL: "HANA_Disks_Overview" (SAP Note 1969700) to check for the amount of fragmentation on disk side. |
| L2 delta migration | Disk > memory | When upgrading from SAP HANA <= SPS 08 to SAP HANA >= SPS 09 an L2 delta migration takes place that can temporarily increase the disk space requirements significantly. See SAP Note 2349081 for more information. |
| Large MVCC size | Disk > memory | MVCC information (SAP Note 2169283) can allocate additional space on disk. SAP Note 2146989 discusses a specific MVCC issue in context of upgrades to SAP HANA 1.0 SPS 09. You can use SQL: "HANA_Tables_DiskSize_UnifiedTabl es" (SAP Note 1969700) in order to check for table disk sizes including MVCC space. |
| Activities with heavy data movement (table redistribution, migration, data load, delta merge or optimize compression of large tables) | Disk > memory | Processing a larger amount of data can result in an temporary increase of disk space requirements for various reasons (shadow pages, snapshots, uncompressed columns, interim tables, ...). For that reason the Storage Whitepaper available via SAP Note 1900823 recommends to make sure that the double data size should be used during operations like table redistribution or migration import. |
| Database snapshots | Disk > memory | Database snapshots can result in significantly increased disk space requirements, because the before image of modified blocks needs to be stored in addition to the normal data blocks. Therefore you should make sure that old snapshots are deleted. SQL: "HANA_IO_Snapshots" (SAP |

| | | |
|---|---|---|
| | | Note 1969700) can be used to check for old snapshots. See SAP Note 2100009 for more information related to savepoints and snapshots. You can use the hdbcons command "snapshot a <snapshot_id>" (SAP Note 2222218) to find out how much disk space is allocated due to a snapshot. In the output you can find the relevant size information: dropping this snapshot would free <num_pages> pages with total size of <size_MB> MB See SAP Note 2100009 ("What are reasons for snapshots being retained for a long time?") for typical situations when snapshots exist for a long time. |
| Low savepoint frequency | Disk > memory | Normal savepoints work in a similar way like database snapshots and all pages referenced by one savepoint are kept until the next savepoint succeeds. If a page is changed, both the former version and the new version needs to be stored in parallel. Normally this doesn't result in a significantly increased disk space, but in case of a low savepoint frequency (e.g. due to a very long running savepoint or due to a high setting of parameter global.ini -> [persistence] -> savepoint_interval_s) or in case of a high change load the persistence overhead can be significant. See SAP Note 2100009 for more information related to savepoints and snapshots. In this scenario you can observe a rising amount of shadow pages and check ID 383 ("Max. size of shadow pages (GB, last day)") of the SAP HANA Mini Checks (SAP Note 1999993) can be reported as potentially critical. |
| Garbage collection blocked | Disk > memory | Blocked persistence garbage collection can result in a significant increase of disk space. SAP Note 2169283 describes how to analyze issues with garbage collection. |
| Large DELETE / TRUNCATE | Disk > memory | As described in SAP Note 2014987 the disk size can remain at a high level after having performed a large DELETE or TRUNCATE operation. The amount of allocated disk space can be 16 MB * <num_columns> * <num_partitions> in the worst case. Proceed as described in SAP Note 2014987 in order to reduce the allocated disk size. |
| Orphan disk LOBs | Disk > memory | Orphan disk LOBs can be responsible for a significant space allocation on disk that isn't reflected in the memory. See SAP Note 2220627 ("Can there be orphan disk LOBs?") for more information related to orphan disk LOBs. |
| LOB fragmentation | Backup > disk | Although this kind of space overhead doesn't properly fit here (because disk LOBs are never loaded into memory), it should be mentioned for completeness purposes. LOBs are allocated with fix page sizes (>= 4 KB) and so there can be significant unused space, particularly if you have many small LOB values smaller than 4 KB. See SAP Note 2220627 for more information related to LOBs. You can use SQL: "HANA_LOBs_LOBFiles" (SAP Note 1969700) in order to check for allocated LOB space (PHYS_SIZE_MB) and used LOB space (BIN_SIZE_MB). Due to the fact that the full pages are backed up, the backup size can be significantly larger than the used disk size in some cases. |

### 19. Which general optimizations exist for reducing the SQL statement memory requirements?

The following heap allocators are used in context of processing database requests (e.g. for intermediate result sets and structures) and usually their life time ends when the database request is finished:

- Pool/AttributeEngine/Transient
- Pool/AttributeEngine/Transient/updateContainerConcat
- Pool/CSPlanExecutor/PlanExecution

- Pool/DocidValueArray
- Pool/ExecutorPlanExecution
- Pool/Filter
- Pool/itab
- Pool/itab/VectorColumn
- Pool/JoinEvaluator
- Pool/JoinEvaluator/DictsAndDocs
- Pool/JoinEvaluator/JEAssembleResults
- Pool/JoinEvaluator/JEAssembleResults/Results
- Pool/JoinEvaluator/JECalculate
- Pool/JoinEvaluator/JECalculate/TmpResults
- Pool/JoinEvaluator/JECalculate/Results
- Pool/JoinEvaluator/JECreateNTuple
- Pool/JoinEvaluator/JEPlanData/deserialized
- Pool/JoinEvaluator/JEPreAggregate
- Pool/JoinEvaluator/JERequestedAttributes/Results
- Pool/JoinEvaluator/JEStep1
- Pool/JoinEvaluator/JEStep2
- Pool/JoinEvaluator/NTuple
- Pool/JoinEvaluator/PlanDataAttrVals/Deserialized
- Pool/JoinEvaluator/ValueList
- Pool/malloc/libhdbcalcengine.so
- Pool/malloc/libhdbcalcengineapi.so
- Pool/malloc/libhdbcalcenginepops.so
- Pool/malloc/libhdbcswrapper.so
- Pool/malloc/libhdbevaluator.so
- Pool/malloc/libhdbolap.so
- Pool/mds
- Pool/parallel/aggregates
- Pool/parallel/align
- Pool/parallel/compactcol
- Pool/parallel/ihm
- Pool/parallel/pop
- Pool/parallel/temp_aggregates
- Pool/parallel/temp_dimensions
- Pool/parallel/temp_other
- Pool/RowEngine/LOB
- Pool/RowEngine/MonitorView
- Pool/RowEngine/QueryExecution
- Pool/RowEngine/QueryExecution/SearchAlloc
- Pool/SearchAPI
- Pool/SearchAPI/Itab Search
- Pool/StringContainer
- Pool/ValueArray
- Pool/XDictData

To a certain extent this specific allocator class can also be identified in monitoring view M_HEAP_MEMORY with COMPONENT = 'Statement Execution & Intermediate Results', but the assignment to this class is not always 100 % precise.

The following general rules can help to reduce memory requirements of SQL statements during execution:

| Rule | Details |
|---|---|

| | |
|---|---|
| As few rows as possible | Use as many restrictions as possible so that the amount of fetched records is as small as possible. |
| As few columns as possible | Select as few columns as possible. Avoid "SELECT *" whenever possible. |
| Avoid UNION ALL, UNION, INTERSECT, EXCEPT | These operations can't be handled by the column engine and so optimizations like late materialization (SAP Note 1975448) are not possible. As a consequence the memory requirements can significantly increase. Therefore you should use alternative whenever possible (e.g. OR instead of UNION or UNION ALL). |
| BW: Configure safety belt | If BW queries read a large amount of data, check if it is possible to configure the query safety belt as described in SAP Note 1127156. |
| Homogeneous user for composite provider / stacked calculation view on top of scripted calculation view | If the user of a composite provider / stacked calculation view and of an inner scripted calculation view is different, predicate pushdown may be impacted and so a high memory consumption related to intermediate result set allocators is possible. Either make sure that the owner is identical or define the scripted calculation view in "Invoker" mode. |

If the memory consumption of these allocators remains at levels that can hardly be explained by executions of database requests, you can consider the following technical SAP HANA root causes:

| Scenario | Details |
|---|---|
| Memory leak | If you see a steady size increase, it can be caused by a memory leak, e.g.: SAP Note 2062555 (join operation in the subquery of an UPDATE statement, fixed with Rev. 1.00.83) SAP Note 2088349 (querying calculation views with currency conversion, fixed with Rev. 1.00.84) If you suspect a memory leak that is not documented, yet, open a SAP incident on component HAN-DB in order to request a more detailed analysis. |
| SAP HANA internal reference still open | Normally the statement specific heap allocators should be relased as soon as the database request ends. Due to SAP HANA bugs it can happen that the cleanup isn't performed if certain references (like temporary tables) still exist. You can check if you suffer from this scenario by clearing the SQL cache globally or some suspicious entries individually. See SAP Note 2124112 ("How can entries in the SQL cache be invalidated or reparsed manually?") for more information. Attention: Clearing the SQL cache results in additional parsing requirements and so temporary performance regressions are possible. The following already known scenarios exist: SAP Note 2312976 (DML operations, problem exists for Rev. 1.00.100 - 1.00.102.06 and 1.00.110 - 1.00.112.01) SAP Note 2312983 (memory leak in Pool/parallel/aggregates when querying on distributed environment with SAP HANA 1.00.100 - 1.00.102.06 and 1.00.110 - 1.00.112.02) SAP Note 2533352 (no proper cleanup after execution, fixed with SAP HANA >= 1.00.122.13, >= 2.00.012.02 and >= 2.00.021) SAP Note 2535110 (Memory Leak on Pool/parallel/compactcol and Pool/parallel/aggregates or Pool/itab with SAP HANA <= 1.00.122.12, <= 2.00.012.01 and 2.00.020) If the size of statement allocators reduces significantly after clearing the SQL cache, you can use this approach as a workaround and additionally open a SAP incident on component HAN-DB in order to request a fix for this behavior. |
| No cleanup in context of terminations | The following scenarios can be responsible for an incomplete cleanup in case of terminations: SAP HANA Revisions <= 1.00.122.12, <= 2.00.002.02, <= 2.00.012.01 and 2.00.020 can suffer from an increase in Pool/itab in context of INSERT abortions (SAP Note 2535110). With SAP HANA <= 1.00.122.15 the smart data access (SDA, SAP Note 2180119) related internal procedure SDA_SELECT_AS_ITAB_DEV isn't OOM safe and so memory can remain allocated after an out-of-memory situation. With SAP HANA <= 1.00.122.15, <= 2.00.012.04 and <= 2.00.24 an OOM termination can result in an incomplete cleanup of memory allocations of distributed queries (SAP Note 2612022). |

| | |
|---|---|
| BW temporary tables | In BW environments the high utilization can be linked to temporary objects. In this case you can run report RSDDTMPTAB_DELETE to drop these temporary objects in order to check if it has a positive impact on the Pool/itab size (SAP Note 2352541). Be aware that running this report can result in terminations of currently running reports. |
| Planning engine | If the allocator size is large in context of planning engine activities, you can check if dropping no longer required planning sessions can help to reduce the allocations (SAP Note 2169283 -> "How can garbage collection be triggered manually?" -> "Planning engine garbage collection"). SAP Note 2583148 describes a problem with missing garbage collection in context of the TMA application. |
| MDX | If you execute MDX queries (e.g. using SAP HANA Studio), make sure that you explicitly close MDX requests (MDX CLOSE REQUEST <guid>) when you no longer need them. A COMMIT will not automatically close the requests. If you suspect orphan MDX queries (e.g. because MDX CLOSE REQUEST wasn't executed), you can check for MDX related temporary tables in M_TEMPORARY_TABLES (MDX_..._<guid>). By dropping these tables (DROP TABLE _SYS_BIC.MDX_..._<guid>) also the related internal tables should be dropped. Only drop these tables if you are sure that they are no longer required. |
| BPC queries with MDX | If BPC reports are executed on the system, the results may not be closed properly in context of ENABLE_HANA_MDX = 'X' (SAP Note 2108247). |
| Smart data access | If you use smart data access (SAP Note 2180119) with Rev. <= 1.00.85.02 or Rev. 1.00.90 - 1.00.91, a SAP HANA bug can be responsible for growing Pool/itab requirements. Upgrade to a more recent SAP HANA Revision in order to resolve the problem. See SAP Note 2242507 for more information. |

### 20. How can the tables with the highest memory consumption be determined?

You can use *SQL: "HANA_Tables_LargestTables"* (SAP Note 1969700) in order to check for the largest tables in memory. The following ORDER_BY settings are possible:

| ORDER_BY | Details |
|---|---|
| MAX_MEM | The tables (including indexes and LOBs) with the highest possible maximum memory consumption are shown. The maximum memory information is independent of the currently loaded columns and so it provides a general overview independent of the current load state. |
| CURRENT_MEM | The tables with the highest current memory consumption (including indexes and LOBs) are displayed. |
| TABLE_MEM | The tables with the highest current memory consumption (excluding indexes and LOBs) are displayed. |
| INDEX_MEM | The tables with the highest index memory consumption are displayed. |

Be aware that there are situations where the maximum memory information (M_CS_TABLES.ESTIMATED_MAX_MEMORY_SIZE_IN_TOTAL) is not filled properly, particularly after DDL operations with SPS 08 and below. If you have doubts you can user ORDER_BY = 'TOTAL_DISK' to display the tables with the highest disk space consumption.

### 21. How much swap space should be configured for SAP HANA hosts?

It is recommended to configure a small swap space in order to avoid performance regressions at times of high memory utilization on operating system side. Instead it is usually better if activities are terminated with "out of memory" errors. This makes sure that the overall system is still usable and only certain requests are

terminated. A good value for the swap space is 2 GB (see e.g. SAP Note 1944799 for SLES environments).

## 22. What is memory garbage collection?

Memory garbage collection and defragmentation is done in order to release no longer used memory. It is not required to perform this task manually as SAP HANA will automatically take care for this activity whenever required. In exceptional cases you can trigger / configure memory garbage collection manually:

| Command / Setting | SAP Note | Details |
|---|---|---|
| hdbcons 'mm gc -f' | 2222218 | This command triggers an immediate garbage collection. Defragmentation will happen as much as possible. Attention: Executing this command has potentially critical side-effects like a temporary blockage of business operations, a reduction of address space or - in the long run - increased memory fragmentation. Therefore it must only be executed when advised by SAP support. |
| global.ini -> [memorymanager] -> gc_unused_memory_threshold_abs<br>global.ini -> [memorymanager] -> gc_unused_memory_threshold_rel | 2169283 | These parameters trigger a garbage collection when both the absolute and relative value is exceeded. As soon as one of the configured limits is reached, memory garbage collection stops. Attention: Setting these parameters can result in frequently recurring memory defragmentation activities and related performance regressions. If at all, you should set these parameters only temporarily (e.g. for a few minutes) during a less critical time frame. Unsetting the parameters will not stop the initial defragmentation. |

**Attention:** Setting these parameters can cause significant performance issues, so they shouldn't be used unless explicitly requested by SAP support.

The following problems are possible when triggering manual memory garbage collection:

| Risk | Details |
|---|---|
| OOM situations | Each memory garbage collection has an impact on the virtual address space utilization and so the risk of out-of-memory terminations because of address space limitations increases. See "Which indications exist that an OOM situation is triggered by the operating system?" for more information. |
| Performance regressions | At runtime of a memory garbage collection SAP HANA internal lock contention can result in reduced performance and increased resource consumption. In busy systems contention and spin locks on operating system side are possible when releasing memory back to the operating system. This scenario results in increased system CPU consumption and page faults. As a workaround the following parameter can be set in order to execute the defragmentation sequentially: indexserver.ini -> [memorymanager] -> disabled_parallel_tasks = poolgarbagecollection |

## 23. Why do I get an OOM although the SAP HANA allocation limits aren't reached?

The following reasons can be responsible for OOM situations although neither the global nor the process specific allocation limits aren't reached:

| Reason | Details |
|---|---|
| Operating system memory exhausted | Check if the available memory is exhausted on operating system side, e.g. because of external software allocating a lot of memory, large caches or another SAP HANA instance. Make sure that in the future there is always enough physical memory available to host the complete SAP HANA allocation limit. See "Which indications exist that an OOM situation is triggered by the operating system?" below for more details. |
| Small temporary process allocation limit | Based on the defined allocation limits SAP HANA and the current service memory allocations the temporary process allocation limit (TPAL) may be significantly smaller than the defined allocation limit. As a consequence OOMs are possible although the configured allocation limits aren't reached. SAP Note 2133638 describes a related startup issue that can happen as of Rev. 90. |
| Statement memory limit reached | OOM dumps with "compositelimit" in their names are no global memory shortages. Instead they are linked to a defined statement memory limit. See "Is it possible to limit the memory that can be allocated by a single SQL statement?" above for more details. |

### 24. How can I involve SAP to perform a detailed memory check?

A detailed SAP HANA memory check and further general health checks and performance optimiaztions are performed as part of the SAP HANA Technical Performance Optimization Service (TPO). See SAP Note 2177604 for more information.

### 25. Why is the allocated memory in some heap allocators very large?

The column EXCLUSIVE_ALLOCATED_SIZE in monitoring view M_HEAP_MEMORY (respectively HOST_HEAP_ALLOCATORS) contains the sum of all allocations in this heap allocator since the last startup. Normally also a lot of deallocations happen, so the EXCLUSIVE_ALLOCATED_SIZE can be much higher than the currently allocated size. For example, if over time 100 MB are allocated and deallocated 10 times, the actual allocated size is 0, but EXCLUSIVE_ALLOCATED_SIZE would show 1 GB (10 * 100 MB).

If the overall allocated memory is much higher than the overall used memory, the difference is usually free for reuse, so no longer heap allocator specific. Therefore the EXCLUSIVE_ALLOCATED_SIZE information can only be used to understand which heap allocators have the highest "throughput" in terms of memory allocations, but it is not helpful to understand the current memory situation.

### 26. Why does PlanViz show a high "Memory Allocated" figure?

If you observe a high "Memory Allocated" figure in PlanViz (SAP Note 2073964) that may significantly exceed the configured statement_memory_limit setting, this is typically caused by the same reason like discussed in the previous question: PlanViz summarizes the overall memory allocation irrespectively of intermittent deallocations. As a consequence the recorded allocated memory can be much higher than maximum memory allocation at a specific point in time.

See SAP Note 2302903 for more information.

### 27. Why does the delta storage allocate more memory with SAP HANA SPS >= 09?

With SAP HANA SPS 09 the delta storage was significantly adjusted. As a consequence the minimum memory footprint of the delta storage of a loaded empty column increased from around 2 KB to more than 8

KB. Having many empty tables with many columns this can increase the overall delta storage size by 10 GB and more. This is an expected behavior that can't be changed.

## 28. Are there any special memory considerations for multitenant databases?

In multitenant database container (MDC) scenarios (SAP Note 2101244) you should make sure that individual containers don't consume excessive amounts of memory, impacting other containers or the system database. On tenant level the memory can be controlled by the service specific parameter global.ini -> [memorymanager] -> allocationlimit in the best way. Optimally the sum of all tenant allocation limits sums up to the global allocation limit, but it is also possible to exceed it.

Example:

- Global allocation limit: 1000 GB
- Tenant service allocation limits: 500 GB, 400 GB, 300 GB

If only a single tenant reaches its allocation limit while the others are well below, the global allocation limit isn't exceeded. Only when several tenants approach their individual allocation limit, the global allocation limit can become a real limit and result in OOMs in all tenants.

Furthermore the following special MDC memory parameters exist:

| Parameter | Unit | Default | Validity | Details |
|---|---|---|---|---|
| global.ini -> [multidb] -> systemdb_reserved_memory | MB | 0 | >= SPS 12 | This parameter allows you to configure a minimal amount of memory (in MB) to be exclusively used by the MDC system database. |

## 29. Which errors indicate memory issues on SAP HANA client side?

Normally memory issues are more likely on SAP HANA server side, but in some scenarios also the SAP HANA client can run into a memory bottleneck. In this case you can see terminations with the following client error:

```
SQL error -9300: no more memory SQL error -10760: Memory allocation failed
```

In SAP ABAP client environments you may find short dumps like DBSQL_ALLOCATION_FAILED, DBSQL_DBSL_NO_MEMORY or DBSQL_NO_PERM_MM_MEMORY for similar reasons.

If you experience these errors, there is usually something wrong with the general memory configuration on client side (operating system or client product like SAP ABAP), e.g. wrong ulimit settings.

## 30. Can there be fragmentation in the heap memory?

Yes, heap memory can fragment to a certain extent. When an out-of-memory situation happens and the allocated memory is still higher than the used memory, the difference is caused by heap memory fragmentation. You can find related fragmentation information in the out-of-memory dump (SAP Note 1984422), e.g.:

```
Total allocated memory= 760083382272b (707.88gb) Total used memory = 665270861313b (619.58gb)
Heap memory fragmentation: 12
```

In general a heap memory fragmentation up to 15 % can be considered as acceptable.

A particularly high, non-reclaimable fragmentation can be a consequence of underlying limitations /

configuration issues, e.g. an inadequate setting of /proc/sys/vm/max_map_count. See "Which indications exist that an OOM situation is triggered by the operating system?" for more information.

Be aware that the calculation of the memory fragmentation in trace files can show misleading high values in case of large memory allocation requests, e.g.:

```
Failed to allocate 2565818396904 byte.

...

Heap memory fragmentation: 58% (this value may be high if defragmentation does not help solving
the current memory request)
```

This combination (high 2.4 TB allocation request, high 58 % fragmentation) typically indicates that the high fragmentation value isn't reliable and should be ignored at this point. It is more important to understand and resolve the high memory allocation request.

If you want to check for the current heap memory fragmentation, you can use *SQL: "HANA_Memory_ProcessMemory"* (SAP Note 1969700).

Example:

```
-------------------------------------------------------------------------------------------------
-------
|HOST |PORT |PAL_GB
|ALLOC_GB|HEAP_USED_GB|FREE_GB|FRAG_GB|ALLOC_PCT|HEAP_USED_PCT|FREE_PCT|FRAG_PCT|
-------------------------------------------------------------------------------------------------
-------
|saphana|30003| 176.55| 176.14| 155.34| 0.00| 20.80| 99.77| 87.98| 0.00| 11.78|
-------------------------------------------------------------------------------------------------
-------
```

Effects of different cleanup options on these numbers:

- Internal ad-hoc defragmentation or manual "hdbcons 'mm gc'": Reduction of FRAG_GB, increase of FREE_GB
- Reclaim defragmentation or manual "hdbcons 'mm gc -f'": Minimization of FREE_GB and FRAG_GB
- Reclaim shrink or manual "hdbcons 'resman s'": Reduction of HEAP_USED_GB

Before an OOM is triggered, SAP HANA will always reduce fragmentation as much as possible. It is also possible - but usually not required - to trigger the defragmentation manually as described in "What is memory garbage collection?" above.

## 31. Which indications exist that an OOM situation is triggered by the operating system?

The following indications exist that an out-of-memory situation is triggered by the operating system and not by SAP HANA:

| Symptom | Detail |
|---|---|
| <service>_<host>.<port>.rtedump.<timestamp>.oom_memory_release.trc dump | This type of SAP HANA dump is only generated in combination with operating system related OOM situations. |
| [MEMORY_OOM] section in OOM dump: Sum of AB (allocated byte) significantly smaller than GLOBAL_MAX_ALLOCATION_LIMIT "--- precharge ok ---" entries in "Out of memory reasons" overview "Could not return | If the sum of allocated memory is smaller than the SAP HANA global allocation limit (and the amount of requested memory is not extraordinary large), the OOM is |

| | |
|---|---|
| <bytes>b to operating system. This is a configuration problem of your operating system: Please increase /proc/sys/vm/max_map_count" Other information in OOM dump: Rather small value for /proc/sys/vm/max_map_count (SAP Note 1980196) Value smaller than 100 for SOFTVIRTUALLIMIT in /etc/sysconfig/ulimit in combination with an installed ulimit.rpm package ("rpm -qa \| grep ulimit") | normally triggered from outside of SAP HANA. In this case you may also see "--- precharge ok ---" information in the OOM dump. Reasons can be: Ulimit memory limitation (e.g. due to installed ulimit.rpm package or because of explicit configuration) Inadequate /proc/sys/vm/max_map_count setting (SAP Note 1980196) High ulimit setting for stack (SAP Note 2488924) Insufficient physical memory (e.g. due to inadequate SAP HANA memory settings or external software consuming a lot of memory) Address space limit reached (Intel: 128 TB, Power: 16 TB, Power with bigmem: 64 TB); make sure that bigmem flavor is used with Power on SLES 11.x; on SLES >= 12 bigmem is already default |
| /var/log/messages contains messages like: <process> invoked oom-killer Out of memory: Kill process <pid> (hdbindexserver) score <score> or sacrifice child | This OOM killer functionality of Linux is used whenever it runs short on physical memory. In this case processes are terminated in order to reclaim memory. |

If you face these symptoms, you can proceed as described in question "Which options exist to reduce the risk of SAP HANA memory issues?" -> "OS configuration" and "Strict NUMA memory binding" above.

### 32. What is the SAP HANA resource container?

The SAP HANA resource container consists of the row store and heap allocators with information that may be re-used like:

- SAP HANA page cache (Pool/PersistenceManager/PersistentSpace/DefaultLPA/Page)
- Column store tables

It doesn't cover heap areas that can't be re-used - particulary related to SQL statement data processing, e.g.:

- Pool/itab
- Pool/JoinEvaluator/JEAssembleResults
- Pool/parallel/aggregates
- Pool/RowEngine/MonitorView
- Pool/TableConsistencyCheck

There is no easy approach to identify allocators assigned to the resource container.

You can use *SQL: "HANA_Memory_MemoryObjects"* (SAP Note 1969700) in order to check for the current population of the resource container. The hdbcons command "resman info" (SAP Note 2222218) provides general information related to the current resource container state.

When additional memory is required and not available, SAP HANA can shrink the resource container (e.g. by reduction of certain heap allocators or unloading columns). In this case the database trace (SAP Note 2380176) will contain an entry like the following:

```
Information about shrink at <date> <time> Local: Reason for shrink: Precharge for big block
```

`allocation.`

The hdbcons command "resman shrink", as e.g. suggested in SAP Note 2301382, only works on the resource container, external allocators can't be shrunk with this command.

### 33. How can the types in M_MEMORY_OBJECTS be mapped to allocators?

The object types in monitoring view M_MEMORY_OBJECTS use an individual naming convention. The most important object types can be mapped in the following way:

| Type | Allocators / Memory | Details |
|---|---|---|
| AttributeEngine/AttributeValueContainerElement | Pool/AttributeEngine* Pool/ColumnStoreTables* | Column store tables |
| Cache/Hierarchy | Pool/hierarchyBlob | Hierarchy cache |
| Persistency/Pages/Default | Pool/PersistenceManager/PersistentSpace(0)/DefaultLPA/Page Pool/PersistenceManager/PersistentSpace/DefaultLPA/Page | SAP HANA page cache |
| Persistency/Pages/RowStore | Shared Memory (allocators Pool/RowStoreTables/* aren't persisted) | Row store |

Starting with SAP HANA 2.0 SPS 01 the mapping can be retrieved from monitoring view M_MEMORY_OBJECT_DISPOSITIONS.CATEGORY.

### 34. In which order are objects unloaded from the resource container?

The unload order of objects in the resource container depends on disposition and unload priority (SAP Note 2127458) settings. A rough mapping is shown in the following table, in general one object type can have portions assigned to different dispositions:

| Disposition | Related objects | Parameter | Default |
|---|---|---|---|
| early unload | columns of tables with unload priorities 6 to 9 | global.ini -> [memoryobjects] -> disposition_weight_early_unload | 100 |
| paged attribute | paged attributes (SAP Note 1871386) | global.ini -> [memoryobjects] -> disposition_paged_attribute | 300 |
| (internal) short term | SAP HANA page cache Hierarchy cache | global.ini -> [memoryobjects] -> disposition_weight_short_term | 300 |
| lob read lob read small lob write lob write small | disk LOBs | indexserver.ini -> [persistence] -> disposition_lob_read indexserver.ini -> [persistence] -> disposition_lob_read_small indexserver.ini -> [persistence] -> disposition_lob_write indexserver.ini -> [persistence] -> disposition_lob_write_small | 300 |
| mid term | | global.ini -> [memoryobjects] -> disposition_weight_mid_term | 900 |

| | | | |
|---|---|---|---|
| long term | columns of tables with unload priorities 1 to 5 | global.ini -> [memoryobjects] -> disposition_weight_long_term | 2700 |
| non swappable | columns of tables with unload priority 0 row store | | 0 |

The disposition weight is divided by the time since the last access of a resource and resources with the smaller resulting values are unloaded first.

Example:

- Column with unload priority 5 last touched 10 hours ago -> disposition result value (based on hours) = 2700 / 10 = 270
- Page in page cache last touched 1 hour ago -> disposition result value (based on hours) = 300 / 1 = 300
- The column has the lower result value (270 vs. 300) and so it is unloaded earlier than the page of the page cache.

In general it is not required to adjust the disposition parameters because the weight factors provide a reasonable overall unload priority, except in a few scenarios:

- In case of critical bugs in the context of unloads it can be useful to increase disposition_weight_long_term and disposition_wait_early_unload (e.g. by factor 10 to 100) in order to make sure that the page cache is unloaded with a higher priority than usual and column unloads are the last resort in case of memory shortage.
- The same applies when you want to minimize column store unloads (e.g. in order to avoid unnecessary reloads or alerts). Be aware that column store unloads can be considered as harmless when only tables with unload priority >= 6 or rarely accessed tables are unloaded. In this case it is neither required nor recommended to adjust the default settings.

Due to a bug with SAP HANA <= 122.03 it can happen that column unloads happen in an undesired order and critical columns are unloaded earlier than intended (SAP Note 2458491). In this case you can manually unload non-critical columns as a workaround (SAP Note 2127458).

You can use *SQL: "HANA_Memory_Objects_Dispositions"* (SAP Note 1969700) in order to check for current disposition / objects / allocators mappings in a system.

Example:

```
-------------------------------------------------------------------------------------------------
--------------------
|OBJECT_TYPE |DISPOSITION |OBJECT_COUNT|OBJECT_SIZE_GB|SIZE_PER_OBJECT_KB|
-------------------------------------------------------------------------------------------------
--------------------
|AttributeEngine/AttributeValueContainerElement |LONG_TERM | 1283759| 3442.49| 2811.83|
|Cache/HierarchyCache |SHORT_TERM | 7209| 499.31| 72627.05|
|Persistency/Pages/Default |INTERNAL_SHORT_TERM| 194839| 161.37| 868.47|
|Persistency/Pages/RowStore |NON_SWAPPABLE | 7364608| 116.92| 16.64|
|Persistency/Pages/Default |SHORT_TERM | 1400790| 102.45| 76.69|
|Cache/MdxHierarchyCache |SHORT_TERM | 1225| 40.59| 34744.45|
|AttributeEngine/AttributeValueContainerElement |NON_SWAPPABLE | 1295833| 8.95| 7.24|
|Persistency/Pages/Default |LONG_TERM | 236301| 5.27| 23.42|
|Persistency/Container/VirtualFile |SHORT_TERM | 3505507| 3.13| 0.93|
|Persistency/Pages/Default |TEMPORARY | 3983| 2.65| 699.15|
|Persistency/Pages/Converter/Default |TEMPORARY | 5667| 1.39| 258.69|
-------------------------------------------------------------------------------------------------
```

---

### 35. Is the SAP HANA memory information always correct?

In general you can rely on the SAP HANA memory information, but the following exceptions exist:

| Area | SAP Note | Details |
|---|---|---|
| M_CONTEXT_MEMORY | | Memory information for granular units like connection and SQL statement are tracked in M_CONTEXT_MEMORY. It can be evaluated via SQL: "HANA_Memory_ContextMemory" (SAP Note 1969700). This information tells you how much statement execution specific memory is currently allocated. This information is usually precise and it is used as basis of memory features like the statement memory limit. The following exceptions exist: SAP Note 2593571 (SAP HANA <= 1.00.122.13, <= 2.00.012.02, <= 2.00.021): Wrong implicit memory booking behavior in context of liveCache procedure calls SAP Note 2603589 (SAP HANA <= 1.00.122.13, <= 2.00.012.02, <= 2.00.022): Allocations in orawstream::reserve are not properly deallocated from context memory. SAP Note 2584388 (SAP HANA <= 1.00.122.14): SQL cache related memory allocations may be accounted for the context memory and so statistics server calls (SAP Note 2147247) can show a high context memory size although at the same time the actual intermediate memory allocation is rather small. SAP Note 2628153 (SAP HANA 1.00.122.16): A wrong memory accounting can result in rising context memory values. SAP Note 2669798 (SAP HANA <= 1.00.122.17): Wrong memory accounting in context of MDS (SAP Note 2670064) SAP HANA <= 2.00.024.02, 2.00.030: Wrong accounting in config::IniParser::parse In the worst case the wrong context memory allocation can result in statement memory limit terminations. As a workaround you can set the statement_memory_limit parameter sufficiently high to make sure that it isn't reached by the erroneous context memory value. Existing increased bookings can be cleaned by terminating the related connection (SAP Note 2092196). In case of statistics server sessions a restart is possible based on the description in SAP Note 2584388. |
| M_EXPENSIVE_STATEMENTS.MEMORY_SIZE | 2180165 | With SAP HANA <= 1.00.122.13, <= 2.00.012.01, <= 2.00.020 the memory value is |

| | | |
|---|---|---|
| | | incomplete and it is not reliable. With later Revisions it represents the highest memory utilization in an involved service. |
| M_SQL_PLAN_CACHE.TOTAL_EXECUTION_MEMORY_SIZE | 2124112 | With SAP HANA <= 1.00.122.14 the memory consumption of the final close operation is captured, not the peak memory consumption of the actual execution. Starting with SAP HANA 1.00.122.15 the peak memory consumption is properly recorded. |

### 36. How can I get an overview of all recent OOM situations?

Trace files may not cover all OOM situations because a trace is only written after the configured oom_dump_time_delta (default: 1 day) is exceeded. Instead you can find an overview of OOM situations in monitoring view M_OUT_OF_MEMORY_EVENTS (SAP HANA 1.0 >= SPS 12) or alternatively via *SQL: "HANA_Memory_OutOfMemoryEvents"* (SAP Note 1969700).

Example:

```
-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------
|OOM_TIME |HOST |PORT |CONN_ID |STATEMENT_HASH |MEM_REQ_GB|MEM_USED_GB|MEM_LIMIT_GB|EVENT_REASON
|TRACEFILE_NAME|
-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------
|2018/01/21 14:56:37|saphana6|30003| 509002|8c9a904596ef7297c18047ae899593d4| 7.28| 199.35|
200.00|GENERIC_COMPOSITE_LIMIT| |
|2018/01/21 14:56:38|saphana6|30003| 509002|8c9a904596ef7297c18047ae899593d4| 7.27| 199.38|
200.00|GENERIC_COMPOSITE_LIMIT| |
|2018/01/21 14:56:40|saphana6|30003| 509002|8c9a904596ef7297c18047ae899593d4| 7.26| 199.45|
200.00|GENERIC_COMPOSITE_LIMIT| |
|2018/01/21 14:56:43|saphana6|30003| 509002|8c9a904596ef7297c18047ae899593d4| 7.27| 199.57|
200.00|GENERIC_COMPOSITE_LIMIT| |
|2018/01/21 14:56:44|saphana6|30003| 509002|8c9a904596ef7297c18047ae899593d4| 0.50| 199.96|
200.00|GENERIC_COMPOSITE_LIMIT| |
|2018/01/22 14:14:19|saphana5|30003| 408089|ea8afb5aed39f133e5f593dfaed1828b| 0.00| 200.00|
200.00|GENERIC_COMPOSITE_LIMIT| |
|2018/01/23 17:28:35|saphana6|30003| 508413|0450975123f2a81eb26a1ebc06f819cf| 3.21| 197.64|
200.00|GENERIC_COMPOSITE_LIMIT| |
|2018/01/24 11:37:24|saphana6|30003| 416809|d589b47003b8db3caf9425ebfaf5b72e| 11.06| 189.43|
200.00|GENERIC_COMPOSITE_LIMIT| |
-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------
```

### 37. Is SAP HANA aware about dynamic memory changes?

If you adjust the amount of physical memory while SAP HANA is up and running, SAP HANA won't automatically consider the new size. To avoid issues you can manually adjust memory related parameters like global_allocation_limit and allocationlimit or synchronize memory adjustments with times of SAP HANA restarts.

### 38. Are all SAP HANA services part of the memory management?

No, not all SAP HANA services (SAP Note 2477204) are covered by the memory management. Exceptions are:

- daemon

- esserver
- etsserver
- rdsyncserver
- streamingserver
- xscontrol
- xsexecagent
- xsuaaserver

As a consequence values in memory related monitoring views may be missing or having unexpected values (e.g. -1 for the process allocation limit).

### Keywords

SAP HANA memory heap allocator table row column store oom out of memory

## Products

SAP HANA, platform edition all versions

## This document refers to

| SAP Note/KBA | Title |
|---|---|
| 2670064 | FAQ: SAP HANA Multi-Dimensional Services (MDS) |
| 2600076 | FAQ: SAP HANA Inverted Individual Indexes |
| 2600030 | Parameter Recommendations in SAP HANA Environments |
| 2593571 | FAQ: SAP HANA Integrated liveCache |
| 2573880 | FAQ: SAP HANA Full System Info Dump |
| 2570371 | FAQ: SAP HANA Execution Engine (HEX) |
| 2520774 | FAQ: SAP HANA Performance Trace |
| 2506811 | FAQ: SAP HANA Dynamic Result Cache |
| 2502256 | FAQ: SAP HANA Caches |
| 2477204 | FAQ: SAP HANA Services and Ports |
| 2470289 | FAQ: SAP HANA Non-Uniform Memory Access (NUMA) |
| 2467292 | memAllocSystemPages failed with rc=12 - Cannot allocate memory |
| 2453348 | Out of Memory Occured with Large Pool/planviz/ and Pool/RowEngine/QueryCompilation |

| | |
|---|---|
| 2416490 | FAQ: SAP HANA Data Aging in SAP S/4HANA |
| 2400022 | FAQ: SAP HANA Smart Data Integration (SDI) |
| 2399993 | FAQ: SAP HANA Fast Data Access (FDA) |
| 2388483 | How-To: Data Management for Technical Tables |
| 2380176 | FAQ: SAP HANA Database Trace |
| 2375917 | How-To: Converting SAP HANA VARBINARY columns to LOB |
| 2370588 | S/4 migration job causes an Out Of Memory during the MUJ step on a HANA System |
| 2349081 | Datavolume increase following an upgrade to SPS09 or higher |
| 2336344 | FAQ: SAP HANA Static Result Cache |
| 2302903 | HANA PlanViz "Memory Allocated" figure is higher than the statement memory limit |
| 2242507 | HANA out of memory problem while using Smart Data Access |
| 2222718 | Troubleshooting HANA High Memory Consumption - Guided Answers |
| 2222277 | FAQ: SAP HANA Column Store and Row Store |
| 2222250 | FAQ: SAP HANA Workload Management |
| 2222218 | FAQ: SAP HANA Database Server Management Console (hdbcons) |
| 2222200 | FAQ: SAP HANA Network |
| 2220627 | FAQ: SAP HANA LOBs |
| 2180165 | FAQ: SAP HANA Expensive Statements Trace |
| 2180119 | FAQ: SAP HANA Smart Data Access |
| 2177604 | FAQ: SAP HANA Technical Performance Optimization Service |
| 2169283 | FAQ: SAP HANA Garbage Collection |
| 2160391 | FAQ: SAP HANA Indexes |
| 2159014 | FAQ: SAP HANA Security |
| 2154870 | How-To: Understanding and defining SAP HANA Limitations |
| 2147247 | FAQ: SAP HANA Statistics Server |
| 2143679 | How-To: Removing Primary Keys of SAP HANA Statistics Server Histories |
| 2142945 | FAQ: SAP HANA Hints |
| 2127458 | FAQ: SAP HANA Loads and Unloads |
| 2124112 | FAQ: SAP HANA Parsing |

| | |
|---|---|
| 2122650 | Hana Server Crashes with 'Composite limit violation (OUT OF MEMORY) occurred' in SPS 08 |
| 2119087 | How-To: Configuring SAP HANA Traces |
| 2116157 | FAQ: SAP HANA Consistency Checks and Corruptions |
| 2114710 | FAQ: SAP HANA Threads and Thread Samples |
| 2112604 | FAQ: SAP HANA Compression |
| 2109355 | How-To: Configuring SAP HANA Inverted Hash Indexes |
| 2101244 | FAQ: SAP HANA Multitenant Database Containers (MDC) |
| 2100009 | FAQ: SAP HANA Savepoints |
| 2092196 | How-To: Terminating Sessions in SAP HANA |
| 2088971 | How-To: Controlling the Amount of Records in SAP HANA Monitoring Views |
| 2081869 | How to handle HANA Alert 64: 'Total memory usage of table-based audit log' |
| 2081591 | FAQ: SAP HANA Table Distribution |
| 2081473 | HANA Resident Memory : High Memory Usage |
| 2073964 | Create & Export PlanViz in HANA Studio |
| 2057046 | FAQ: SAP HANA Delta Merges |
| 2050579 | How to handle HANA Alert 68: 'total memory usage of row store' |
| 2044468 | FAQ: SAP HANA Partitioning |
| 2000003 | FAQ: SAP HANA |
| 2000002 | FAQ: SAP HANA SQL Optimization |
| 1999998 | FAQ: SAP HANA Lock Analysis |
| 1999993 | How-To: Interpreting SAP HANA Mini Check Results |
| 1999930 | FAQ: SAP HANA I/O Analysis |
| 1999880 | FAQ: SAP HANA System Replication |
| 1998599 | How-To: Analyzing high SAP HANA Memory Consumption due to Translation Tables |
| 1984422 | How-To: Analyzing SAP HANA Out-of-memory (OOM) Dumps |
| 1977269 | How to handle HANA Alert 45: 'Check memory usage of main storage of column-store tables' |
| 1977268 | How to handle HANA Alert 40: 'Total memory usage of column-store tables' |
| 1977207 | How to handle HANA Alert 55: Columnstore unloads |
| 1977101 | How to handle HANA Alert 12: 'Memory usage of name server' |

| | |
|---|---|
| 1900257 | How to handle HANA Alert 43: 'Memory Usage of Services' |
| 1899511 | How to handle HANA Alert 44 'Licensed Memory Usage' |
| 1898317 | How to handle HANA Alert 1: ‘Host physical memory usage’ |
| 1862506 | HANA: Statisticsserver runs out of memory (OOM) as of SPS05 |
| 1847202 | Error "400 Bad Request" when executing EPM Add-in report with a big amount of dimension members to be retrieved - BPC NW |
| 2669798 | Query Execution Leads to an Out of Memory Situation |
| 2643641 | DPServer Memory Utilization |
| 2637828 | Memory Leak on Pool/malloc/libhdbbasement.so When Collecting Performance Trace/Planviz/Plan Trace with Function Profiler |
| 2629536 | Unexpected Composite OOM Errors Caused by Setting Total Statement Memory Limit |
| 2628153 | Unexpected Composite Out of Memory Event Occurs Frequently |
| 2624305 | Potential Memory Leakage on Pool/malloc/libhdbcswrapper.so |
| 2612205 | HANA Indexserver Cannot Load Row Store Tables Because of OOM |
| 2612022 | Increased Memory Allocator Size After Distributed Query Execution Failed due to OOM |
| 2603589 | Composite OOM in orawstream::reserve |
| 2601475 | Memory Leak in Pool/malloc/libhdbcsapi.so When Running Enterprise Search Queries |
| 2599658 | Increased Version Count or Data Volume Size and Memory Consumption Increase due to Dangling Transtoken |
| 2597818 | Memory Leak in Pool/ESX When Using PlanViz Execution |
| 2588395 | Erroneous Accounting of Shared Memory in Multitenant Database Container Systems Running in High Isolation Level on Linux |
| 2584388 | High Memory Usage in Allocator Connection/XXXXXX/Statement/YYYYYYYY/IMPLICIT by User _SYS_STATISTICS |
| 2583148 | Higher garbage memory build up in SAP HANA due to TMA application |
| 2573738 | Rowstore Versions are not Collected on System Replication Target Site When Using Operation Mode Logreplay |
| 2547516 | Consistency Check Execution Causes Growth of Pool/malloc/libhdbbasement.so |
| 2542700 | DPServer memory utilization continues to climb when processing cluster tables |
| 2535110 | Memory Leak on Pool/parallel/compactcol and Pool/parallel/aggregates or Pool/itab |
| 2533352 | Memory Leak on "Pool/JoinEvaluator/JERequestedAttributes/Results" |
| 2532199 | Optimization of the HANA Memory Allocator Pool/Statistics Usage |

| | |
|---|---|
| 2527251 | Memory Leak in Pool/RowEngine/QueryCompilation |
| 2517443 | Filter push down missing for TREXviaDBSL calls on Hana native calculation view when FEMS are used |
| 2497016 | Pages Belonging to Cold Partitions Created With Paged Attribute Are Not Unloaded by The Resource Manager if They Are Pinned by an Inverted Index |
| 2488924 | Linux: Recommended values for maximum stack size of processes |
| 2458491 | Unloads of Recently Columns Despite Older Columns Could be Evicted |
| 2415279 | How-To: Configuring SAP HANA for the SAP HANA Extension Node |
| 2405763 | SAP HANA DB: Log Replay on HSR Secondary Site Hangs |
| 2371445 | SAP HANA SPS 12 Database Maintenance Revision 122.03 |
| 2312983 | Memory leak in Pool/parallel/aggregates when querying on distributed environment |
| 2146989 | SAP HANA: High Number of Persistent Pages of Type UnifiedTableMVCC |
| 1993128 | SAP HANA: column store table unloads and unloading behavior of Memory Objects Container |
| 1980765 | Operations with columns containing only one value may lead to wrong data |
| 1969700 | SQL Statement Collection for SAP HANA |
| 1900823 | SAP HANA Storage Connector API |
| 1871386 | SAP HANA: Paged Attributes |
| 1865554 | MDX: Access type F4 help / improved error update |
| 1813245 | SAP HANA DB: Row store reorganization |
| | SAP HANA Troubleshooting and Performance Analysis Guide |
| | SAP HANA Administration Guide |
| | ABAP Sourcecode Search |
| | Simplification List for SAP S/4HANA |
| | Information Lifecycle Management |