Article <u>Murray Oldfield</u> · Apr 8, 2016 17m read

InterSystems Data Platforms and Performance Part 4 - Looking at Memory

This post will guide you through the process of sizing shared memory requirements for database applications running on InterSystems data platforms. It will cover key aspects such as global and routine buffers, gmheap, and locksize, providing you with a comprehensive understanding. Additionally, it will offer performance tips for configuring servers and virtualizing IRIS applications. Please note that when I refer to IRIS, I include all the data platforms (Ensemble, HealthShare, iKnow, Caché, and IRIS).

A list of other posts in this series is here

When I first started working with Caché, most customer operating systems were 32-bit, and memory for an IRIS application was limited and expensive. Commonly deployed Intel servers had only a few cores, and the only way to scale up was to go with big iron servers or use ECP to scale out horizontally. Now, even basic production-grade servers have multiple processors, dozens of cores, and minimum memory is hundreds of GB or TB. For most database installations, ECP is forgotten, and we can now scale application transaction rates massively on a single server.

A key feature of IRIS is the way we use data in shared memory usually referred to as database cache or global buffers. The short story is that if you can right size and allocate 'more' memory to global buffers you will usually improve system performance - data in memory is much faster to access than data on disk. Back in the day, when 32-bit systems ruled, the answer to the question how much memory should I allocate to global buffers? It was a simple - as much as possible! There wasn't that much available anyway, so sums were done diligently to calculate OS requirements, the number of and size of OS and IRIS processes and real memory used by each to find the remainder to allocate as large a global buffer as possible.

The tide has turned

If you are running your application on a current-generation server, you can allocate huge amounts of memory to an IRIS instance, and a laissez-faire attitude often applies because memory is now "cheap" and plentiful. However, the tide has turned again, and pretty much all but the very largest systems I see deployed now are virtualized. So, while 'monster' VMs can have large memory footprints if needed, the focus still comes back to the right sizing systems. To make the most of server consolidation, capacity planning is required to make the best use of available host memory.

What uses memory?

Generally, there are four main consumers of memory on an IRIS database server:

- Operating System, including filesystem cache.
- If installed, other non-IRIS applications.
- IRIS processes.
- IRIS shared memory (includes global and routine buffers and GMHEAP).

At a high level, the amount of physical memory required is simply added up by adding up the requirements of each of the items on the list. All of the above use real memory, but they can also use virtual memory. A key part of capacity planning is to size a system so that there is enough physical memory so that paging does not occur or is

minimized, or at least minimize or eliminate hard page faults where memory has to be brought back from disk.

In this post I will focus on sizing IRIS shared memory and some general rules for optimising memory performance. The operating system and kernel requirements vary by operating system but will be several GB in most cases. File system cache varies and is will be whatever is available after the other items on the list take their allocation.

IRIS is mostly processes - if you look at the operating system statistics while your application is running you will see cache processes (e.g. iris or iris.exe). So a simple way to observe what your application memory requirements are is to look at the operating system metrics. For example with vmstat or ps on Linux or Windows process explorer and total the amount of real memory in use, extrapolating for growth and peak requirements. Be aware that some metrics report virtual memory which includes shared memory, so be careful to gather real memory requirements.

Sizing Global buffers - A simplified way

One of the capacity planning goals for a high transaction database is to size global buffers so that as much of the application database working set is in memory as possible. This will minimise read IOPS and generally improve the application's performance. We also need to strike a balance so that other memory users, such as the operating system and IRIS process, are not paged out and there is enough memory for the filesystem cache.

I showed an example of what can happen if reads from disk are excessive in <u>Part 2 of this series</u>. In that case, high reads were caused by a bad report or query, but the same effect can be seen if global buffers are too small, forcing the application to be constantly reading data blocks from disk. As a sidebar, it's also worth noting that the landscape for storage is always changing - storage is getting faster and faster with advances in SSDs and NVMe, but data in memory close to the running processes is still best.

Of course, every application is different, so it's important to say, "Your mileage may vary" but there are some general rules which will get you started on the road to capacity planning shared memory for your application. After that you can tune for your specific requirements.

Where to start?

Unfortunately, there is no magic answer. However, as I discussed in previous posts, a good practice is to size the system CPU capacity so that for a required peak transaction rate, the CPU will be approximately 80% utilized at peak processing times, leaving 20% headroom for short-term growth or unexpected spikes in activity.

For example, when I am sizing TrakCare systems I know CPU requirements for a known transaction rate from benchmarking and reviewing customer site metrics, and I can use a broad rule of thumb for Intel processor-based servers:

Rule of thumb: Physical memory is sized at n GB per CPU core for servers running IRIS.

• For example, for TrakCare database servers, a starting point of n is 8 GB. But this can vary, and servers may be right-sized after the application has been running for a while -- you must monitor your systems continuously and do a formal performance review, for example, every six or 12 months.

Rule of thumb: Allocate n% of memory to IRIS global buffers.

- For small to medium TrakCare systems, n% is 60%, leaving 40% of memory for the operating system, filesystem cache, and IRIS processes. You may vary this, say to 50%, if you need a lot of filesystem cache or have a lot of processes. Or make it a higher percentage as you use very large memory configurations on large systems.
- This rule of thumb assumes only one IRIS instance on the server.

For example, if the application needs 10 CPU cores, the VM would have 80 GB of memory, 48 GB for global buffers, and 32 GB for everything else.

Memory sizing rules apply to physical or virtualized systems, so the same 1 vCPU: 8 GB memory ratio applies to TrakCare VMs.

Tuning global buffers

There are a few items to observe to see how effective your sizing is. You can observe free memory outside IRIS with operating system tools. Set up as per your best calculations, then observe memory usage over time, and if there is always free memory, the system can be reconfigured to increase global buffers or to right-size a VM.

Another key indicator of good global buffer sizing is having read IOPS as low as possible, which means IRIS cache efficiency will be high. You can observe the impact of different global buffer sizes on PhyRds and RdRatio with mgstat; an example of looking at these metrics is in Part 2 of this series. Unless you have your entire database in memory, there will always be some reads from disk; the aim is simply to keep reads as low as possible.

Remember your hardware food groups and get the balance right. More memory for global buffers will lower read IOPS but possibly increase CPU utilization because your system can now do more work in a shorter time. Lowering IOPS is pretty much always a good thing, and your users will be happier with faster response times.

See the section below for applying your requirements to physical memory configuration.

For virtual servers, plan not to ever oversubscribe your production VM memory. This is especially true for IRIS shared memory; more on this below.

Is your application's sweet spot 8GB of physical memory per CPU core? I can't say, but see if a similar method works for your application, whether 4GB or 10GB per core. If you have found another method for sizing global buffers, please leave a comment below.

Monitoring Global Buffer usage

The IRIS utility ^GLOBUFF displays statistics about what your global buffers are doing at any point in time. For example to display the top 25 by percentage:

do display^GLOBUFF(25)

For example, output could look like this:

Total buffers: 2560000 Buffers in use: 2559981 PPG buffers: 1121 (0.044%)

Item	Global	Database	Percentage (Count)
1	MyGlobal	BUILD-MYDB1	29.283 (749651)
2	MyGlobal2	BUILD-MYDB2	23.925 (612478)
3	CacheTemp.xxData	CACHETEMP	19.974 (511335)
4	RTx	BUILD-MYDB2	10.364 (265309)
5	TMP.CachedObjectD	CACHETEMP	2.268 (58073)
б	TMP	CACHETEMP	2.152 (55102)
7	RFRED	BUILD-RB	2.087 (53428)
8	PANOTFRED	BUILD-MYDB2	1.993 (51024)
9	PAPi	BUILD-MYDB2	1.770 (45310)
10	HIT	BUILD-MYDB2	1.396 (35727)
11	AHOMER	BUILD-MYDB1	1.287 (32946)
12	IN	BUILD-DATA	0.803 (20550)
13	HIS	BUILD-DATA	0.732 (18729)
14	FIRST	BUILD-MYDB1	0.561 (14362)
15	GAMEi	BUILD-DATA	0.264 (6748)

InterSystems Data Platforms and Performance Part 4 - Looking at Memory Published on InterSystems Developer Community (https://community.intersystems.com)

16	OF	BUILD-DATA	0.161	(4111)
17	HISLast	BUILD-FROGS	0.102	(2616)
18	%Season	CACHE	0.101	(2588)
19	WooHoo	BUILD-DATA	0.101	(2573)
20	BLAHi	BUILD-GECKOS	0.091	(2329)
21	CTPCP	BUILD-DATA	0.059	(1505)
22	BLAHi	BUILD-DATA	0.049	(1259)
23	Unknown	CACHETEMP	0.048	(1222)
24	COD	BUILD-DATA	0.047	(1192)
25	TMP.CachedObjectI	CACHETEMP	0.032	(808)

This could be useful in several ways, for example, to see how much of your working set is kept in memory. If you find this utility is useful please make a comment below to enlighten other community users on why it helped you.

Sizing Routine Buffers

Routines your application is running, including compiled classes, are stored in routine buffers. The goal of sizing shared memory for routine buffers is for all your routine code to be loaded and stay resident in routine buffers. Like global buffers, it is expensive and inefficient to read routines off disk. The maximum size of routine buffers is 1023 MB. As a rule you want more routine buffers than you need as there is always a big performance gain to have routines cached.

Routine buffers are made up of different sizes. By default, IRIS determines the number of buffers for each size; at install time, the defaults for 2016.1 are 4, 16 and 64 KB. It is possible to change the allocation of memory for different sizes; however, to start your capacity planning, it is recommended to stay with IRIS defaults unless you have a special reason for changing. For more information, see routines in the <u>IRIS documentation</u> " config " appendix of the IRIS Parameter File Reference and Memory and Startup Settings in the " Configuring IRIS " chapter of the IRIS System Administration Guide.

As your application runs, routines are loaded off disk and stored in the smallest buffer the routine will fit. For example, if a routine is 3 KB, it will ideally be stored in a 4 KB buffer. If no 4 KB buffers are available, a larger one will be used. A routine larger than 32 KB will use as many 64 KB routine buffers as needed.

Checking Routine Buffer Use

mgstat metric RouLas

One way to understand if the routine buffer is large enough is the mgstat metric RouLas (routine loads and saves). A RouLas is a fetch from or save to disk. A high number of routine loads/saves may show up as a performance problem; in that case, you can improve performance by increasing the number of routine buffers.

cstat

If you have increased routine buffers to the maximum of 1023 MB and still find high RouLas a more detailed examination is available so you can see what routines are in buffers and how much is used with cstat command.

ccontrol stat cache -R1

This will produce a listing of routine metrics including a list of routine buffers and all the routines in cache. For example a partial listing of a default IRIS install is:

Number of rtn buf: 4 KB-> 9600, 16 KB-> 7200, 64 KB-> 2400, gmaxrouvec (cache rtns/proc): 4 KB-> 276, 16 KB-> 276, 64 KB-> 276, gmaxinitalrouvec: 4 KB-> 276, 16 KB-> 276, 64 KB-> 276, Dumping Routine Buffer Pool Currently Inuse size sys sfn inuse old type hash buf rcrc rtime rver rctentry rouname 22: 8937 4096 0 1 1 0 D 6adcb49e 56e34d34 53 dcc5d477 %CSP.UI.Po rtal.ECP.0 36: 9374 4096 1 1 0 5c384cae 56e34d88 13 908224b5 0 М %SYSTEM.Wo rkMgr.1 37: 9375 4096 0 1 1 0 D a4d44485 56e34d88 22 91404e82 %SYSTEM.Wo rkMgr.0 44: 9455 4096 0 1 0 D 9976745d 56e34ca0 57 9699a880 SYS.Monito 0 r.Health.x 2691:16802 16384 0 7 0 da8d596f 0 Ρ 56e34c80 27 383da785 START etc etc

"rtns/proc" on the 2nd line above is saying that 276 routines can be cached at each buffer size as default.

Using this information another approach to sizing routine buffers is to run your application and list the running routines with cstat -R1. You could then calculate the routine sizes in use, for example put this list in excel, sort by size and see exactly what routines are in use. If your are not using all buffers of each size then you have enough routine buffers, or if you are using all of each size then you need to increase routine buffers or can be more direct about configuring the number of each bucket size.

Lock table size

The locksiz configuration parameter is the size (in bytes) of memory allocated for managing locks for concurrency control to prevent different processes from changing a specific element of data at the same time. Internally, the inmemory lock table contains the current locks, along with information about the processes that hold those locks.

Since memory used to allocate locks is taken from GMHEAP, you cannot use more memory for locks than exists in GMHEAP. If you increase the size of locksiz, increase the size of GMHEAP to match as per the formula in the GMHEAP section below. Information about application use of the lock table can be monitored using the system management portal (SMP), or more directly with the API:

```
set x=##class(SYS.Lock).GetLockSpaceInfo().
```

This API returns three values: "Available Space, Usable Space, Used Space". Check Usable space and Used Space to roughly calculate suitable values (some lock space is reserved for lock structure). Further information is available in <u>IRIS documentation</u>.

Note: If you edit the locksiz setting, changes take place immediately.

GMHEAP

The GMHEAP (the Generic Memory Heap) configuration parameter is defined as: Size (in kilobytes) of the generic memory heap for IRIS. This is the allocation from which the Lock table, the NLS tables, and the PID table are also allocated.

Note: Changing GMHEAP requires a IRIS restart.

To assist you in sizing for your application information about GMHEAP usage can be checked using the API:

%SYSTEM.Config.SharedMemoryHeap

This API also provides the ability to get available generic memory heap and recommends GMHEAP parameters for configuration. For example, the DisplayUsage method displays all memory used by each of the system components and the amount of available heap memory. Further information is available in the <u>IRIS documentation</u>.

```
write $system.Config.SharedMemoryHeap.DisplayUsage()
```

The RecommendedSize method can give you an idea of GMHEAP usage and recommendations at any point in time. However, you will need to run this multiple times to build up a baseline and recommendations for your system.

```
write $system.Config.SharedMemoryHeap.RecommendedSize()
```

Rule of thumb: Once again your application mileage will vary, but somewhere to start your sizing could be one of the following:

```
(Minimum 128MB) or (64 MB * number of cores) or (2x locksiz) or whichever is larger.
```

Remember GMHEAP must be sized to include the lock table.

Large/Huge pages

The short story is that huge pages on Linux have a positive effect on increasing system performance. However, the benefits will only be known if you test your application with and without huge pages. The benefits of huge pages for IRIS database servers are more than just performance -- which may only be 40% improvement at best. There are other reasons to use huge pages; When IRIS uses huge pages for shared memory, you guarantee that the memory is available for shared memory and not fragmented.

Note: By default, when huge/large pages are configured, InterSystems IRIS attempts to utilize them on startup. If there is not enough space, InterSystems IRIS reverts to standard pages. However, you can use the memlock parameter to control this behavior and fail at startup if huge/large page allocation fails.

As a sidebar for TrakCare, we do not automatically specify huge pages for non-production servers/VMs with small memory footprints (for example less than 8GB) or utility servers (for example print servers) running IRIS because allocating memory for huge pages may end up orphaning memory, or sometimes a bad calculation that undersizes huge pages means IRIS starts not using huge pages which is even worse. As per our docs, remember that when using huge pages to configure and start IRIS without huge pages, look at the total shared memory at startup and then use that to calculate huge pages. <u>Configuring Huge and Large Pages</u>

Danger! Windows Large Pages and Shared Memory

IRIS uses shared memory on all platforms and versions, and it's a great performance booster, including on Windows, where it is always used. However, there are particular issues unique to Windows that you need to be aware of.

When IRIS starts, it allocates a single, large chunk of shared memory to be used for database cache (global buffers), routine cache (routine buffers), the shared memory heap, journal buffers, and other control structures. On

IRIS startup, shared memory can be allocated using small or large pages. On Windows 2008 R2 and later, IRIS uses large pages by default; however, if a system has been running for a long time, due to fragmentation, contiguous memory may not be able to be allocated at IRIS startup, and IRIS can instead start using small pages.

Unexpectedly starting IRIS with small pages can cause it to start with less shared memory than defined in the configuration, or it may take a long time to start or fail to start. I have seen this happen on sites with a failover cluster where the backup server has not been used as a database server for a long time.

Tip: One mitigation strategy is periodically rebooting the offline Windows cluster server. Another is to use Linux.

Physical Memory

The best configuration for the processor dictates physical memory. A bad memory configuration can significantly impact performance.

Intel Memory configuration best practice

This information applies to Intel processors only. Please confirm with vendors what rules apply to other processors.

Factors that determine optimal DIMM performance include:

- DIMM type
- DIMM rank
- Clock speed
- Position to the processor (closest/furthest)
- Number of memory channels
- Desired redundancy features.

For example, on Nehalem and Westmere servers (Xeon 5500 and 5600) there are three memory channels per processor and memory should be installed in sets of three per processor. For current processors (for example, E5-2600), there are four memory channels per processor, so memory should be installed in sets of four per processor.

When there are unbalanced memory configurations — where memory is not installed in sets of three/four or memory DIMMS are different sizes, unbalanced memory can impose a 23% memory performance penalty.

Remember that one of the features of IRIS is in-memory data processing, so getting the best performance from memory is important. It is also worth noting that for maximum bandwidth servers should be configured for the fastest memory speed. For Xeon processors maximum memory performance is only supported at up to 2 DIMMs per channel, so the maximum memory configurations for common servers with 2 CPUs is dictated by factors including CPU frequency and DIMM size (8GB, 16GB, etc).

Rules of thumb:

- Use a balanced platform configuration: populate the same number of DIMMs for each channel and each socket
- Use identical DIMM types throughout the platform: same size, speed, and number of ranks.
- For physical servers, round up the total physical memory in a host server to the natural break points—64GB, 128GB, and so on—based on these Intel processor best practices.

VMware Virtualisation considerations

I will follow up in future with another post with more guidelines for when IRIS is virtualized. However the following key rule should be considered for memory allocation:

Rule: Set VMware memory reservation on production systems.

As we have seen above when IRIS starts, it allocates a single, large chunk of shared memory to be used for global and routine buffers, GMHEAP, journal buffers, and other control structures.

You want to avoid any swapping for shared memory so set your production database VMs memory reservation to at least the size of IRIS shared memory plus memory for IRIS processes and operating system and kernel services. If in doubt reserve the full production database VMs memory.

As a rule if you mix production and non-production servers on the same systems do not set memory reservations on non-production systems. Let non-production servers fight out whatever memory is left ;). VMware often calls VMs with more than 8 CPUs 'monster VMs'. High transaction IRIS database servers are often monster VMs. There are other considerations for setting memory reservations on monster VMs, for example if a monster VM is to be migrated for maintenance or due to a High Availability triggered restart then the target host server must have sufficient free memory. There are stratagies to plan for this I will talk about them in a future post along with other memory considerations such as planning to make best use of NUMA.

Summary

This is a start to capacity planning memory, a messy area - certainly not as clear cut as sizing CPU. If you have any questions or observations please leave a comment.

As this entry is posted I am on my way to Global Summit 2016. If you are attending this year I will be talking about performance topics with two presentations, or I am happy to catch up with you in person in the developers area.

<u>#Best Practices</u> <u>#InterSystems Business Solutions and Architectures</u> <u>#Performance</u> <u>#System Administration</u> <u>#Caché</u> <u>#InterSystems IRIS</u> <u>#InterSystems IRIS for Health</u>

Source

URL:https://community.intersystems.com/post/intersystems-data-platforms-and-performance-part-4-lookingmemory