Memory Really Matters in 2015

Fabio Massimo Ottaviani – EPV Technologies

July 2015

1 Introduction

The amount of available mainframe memory continues to increase with every new IBM machine generation as well as the amount of memory supported by the latest z/OS releases.
Up to 10TB memory can be installed in IBM z13 while z/OS 2.2 will support up to 4TB in a single LPAR.

The first benefit of an adequate amount of memory is to eliminate paging activity.
While it's true that z/OS can remain stable and run even when sustaining high paging activity, paging and especially page faults are very disruptive to online applications' performance.
Furthermore you have to consider that CPU cycles are needed to perform paging.

Large amounts of memory also allow the full exploitation of the many available DIM (Data In Memory) techniques.
Maintaining data in memory has two additional advantages:

- it improves performance by avoiding the delay caused by I/O operations
- it eliminates the CPU consumption needed to perform I/O operations.

This is particularly important for DB2 workloads with poorly performing buffer pools requesting many read I/Os to reload the required data.

Finally, if enough memory is available, 1MB and 2GB memory frames can be used to greatly reduce the overhead of virtual address translation.

In this paper we will discuss the most important metrics you can use to measure memory related activities.

2 Paging

The most critical paging activity within z/OS is the Page Fault Rate (PFR) which is the rate of page-in operations excluding VIO.
When an address space gets a page fault it suspends its activity waiting for the requested page to become available.
A high number of page faults is undesirable for any address space but it can be especially critical for transactional fast response activities made through IMS, CICS, WEBSPHERE as well as DB2 subsystems.

PFR can be calculated by dividing the SMF71PIN field, provided in SMF 71 records, by the interval duration.

The first question to answer is: is my system memory constrained?

Of course looking at online monitors or to some daily RMF report it’s not enough. You need to review daily trend reports to get a clear understanding of page fault activity.

Nowadays most systems show very small or zero PFR values. For critical production systems you should aim at a single digit PFR.

In EPV for z/OS our default threshold is 50; in the table below we show one month of data, excluding week ends, of a customer’s system. We highlighted in pink all the values higher than 50; we also highlighted in black all the values higher than 1.000.

DATE	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
27/02/2015	251	966	223	535	452	227	587	386	338	438	316	696	1114	556	849	686	367	239	694	164	167	844	184	70
26/02/2015	152	219	117	76	45	31	59	154	92	130	406	239	927	133	176	261	184	239	131	76	52	136	218	111
25/02/2015	89	129	72	51	24	31	47	72	104	78	692	886	1007	308	792	633	157	98	29	91	37	85	180	82
24/02/2015	166	346	185	55	32	19	102	485	228	294	240	601	880	756	546	307	110	260	231	66	33	46	125	106
23/02/2015	24	41	22	9	2	3	22	15	88	51	211	249	889	242	333	382	318	207	201	105	73	51	324	90
20/02/2015	110	118	107	139	55	27	89	93	82	94	344	559	335	145	79	90	420	155	242	91	23	213	100	62
19/02/2015	57	132	74	56	28	60	50	92	105	74	237	348	304	135									263	191
18/02/2015	243	122	65	32	17	11	183	66	240	132	214	336	153	73	253	295	459	197	142	100	30	186	218	91
17/02/2015	100	139	109	49	27	10	11	816	171	377	483	473	499	384	371	618	546	179	201	89	68	24	458	439
16/02/2015	36	106	47	16	10	6	27	31	32	231	302	144	244	421	179	219	296	137	44	77	50	168	200	76
13/02/2015	171	169	91	33	32	20	36	73	59	74	110	265	240	108	61	402	403	431	566	157	85	587	160	85
12/02/2015	107	101	57	28	28	29	40	43	47	172	125	218	646	406	174	710	263	223	121	56	49	273	557	399
11/02/2015	34	88	39	45	19	16	27	62	51	102	309	333	208	131	175	344	464	138	194	50	198	203	231	276
10/02/2015	80	172	111	33	22	35	16	17	118	295	642	510	193	108	355	237	181	300	125	56	72	187	161	38
09/02/2015	46	119	58	31	28	9	67	32	53	56	46	59	222	106	211	220	400	604	279	50	23	55	549	120
06/02/2015	66	142	59	61	30	23	66	153	101	100	221	323	385	365	570	683	528	873	338	128	286	449	171	48
05/02/2015	100	142	59	50	19	57	50	98	82	79	213	244	198	87	109	200	663	200	236	99	51	49	113	39
04/02/2015	99	130	69	100	23	48	40	99	107	121	175	256	210	174	160	691	282	339	440	131	75	70	167	69
03/02/2015	159	174	239	337	108	179	107	718	350	278	383	259	557	207	96	202	159	399	599	305	161	78	201	82
02/02/2015	107	794	427	209	141	93	165	554	673	289	164	80	96	41	186	511	752	675	411	125	69	48	388	399

Figure 1

You can see that PFR is generally high and very high values have been measured in the late morning hours of the last week (from 23rd to 27th February). By looking at this data we can say that this system is memory constrained.

The second question is: who is suffering for this high PFR?

The answer can be found in the following SMF 30 record fields:

- SMF30PGI – private page-in;

- SMF30CPI – common page-in;
- SMF30LPI – LPA page-in.

The graph in the next figure shows the PFR hourly profile of the system on 27th February 2015.

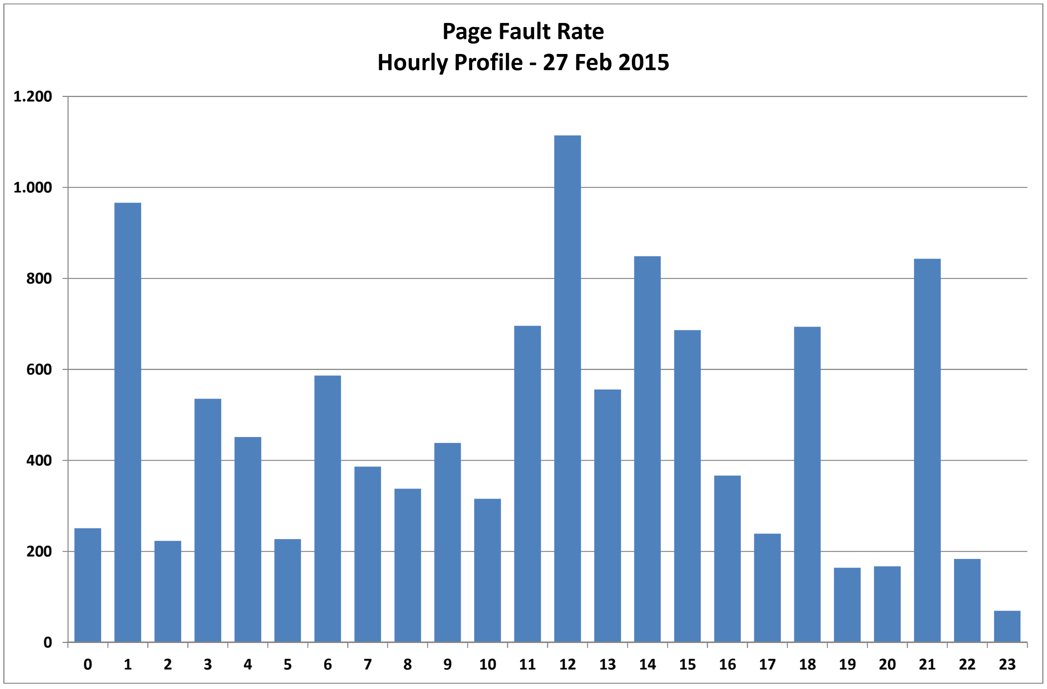


Figure 2

If SMF interval accounting is active and synchronized with RMF you can easily decompose the system paging activity by address space.

In the next table we show the top 10 address space by paging activity at 12 on the same day.

	TOP PAGE FAULT RATE		TOP PAGE OUT RATE	
HOUR	ADDRESS SPACE	PFR	ADDRESS SPACE	PGO
12	WASTA13S	104,3	WASTADM	51,6
12	WASTADMS	92,5	WASTA13S	51,4
12	WASTA10S	83,0	WASTA10S	46,7
12	WASTADM	47,6	WASTADMS	42,6
12	BPXZFS	36,9	BPXZFS	38,5
12	WASTA11S	31,6	DB2HDBM1	22,1
12	WASTA15S	30,7	WASTA12S	20,5
12	WASTA1N	30,6	WAST1DM	20,5
12	WASTA14S	30,4	WASTA1N	19,4
12	WASTA1AS	29,8	WASTA12A	18,2

Figure 3

The Flash Express feature available from zEC12 provides a solid-state internal device which can be used for faster paging. This new storage layer is called Storage Class Memory (SCM).

Same measurements provided in SMF 75 as for standard Page Datasets are also provided for SCM:

- SMF75SLA – Total number of slots;
- SMF75AVU – AVG number of slots used;
- SMF75SIO – Total number of I/Os;
- SMF75PGX – Total number of pages transferred.

You can identify SCM by looking at the page set type flag (SMF75FL2).
Obviously no information is provided for volser (SMF75VOL) and device number (SMF75CHA).

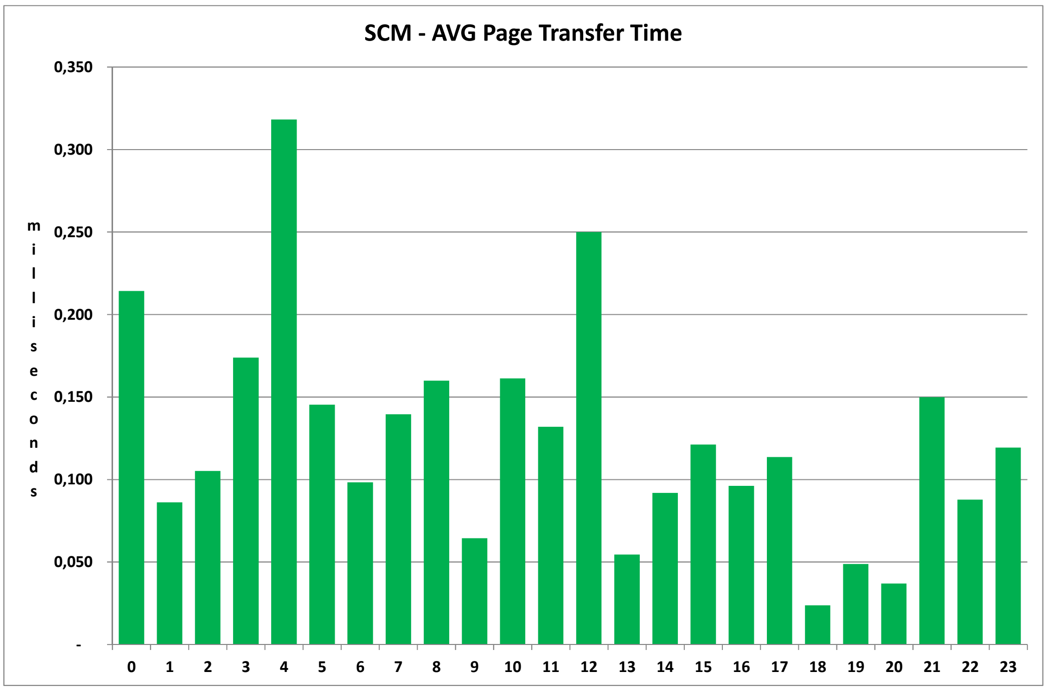


Figure 4

The graph in Figure 4 shows an example of SCM performance. You can see that the average page transfer time range is from 50 to 300 microseconds.

Average page transfer time can be calculated by using the following formula:

*(1000 * SMF75USE * RMFINT)/ (RMFSAMPLES * SMF75PGX)*

SCM performance is normally much better than disks, however:

- SCM performance is still much worse than memory;
- paging to SCM reduces the applications performance penalty but not the CPU overhead due to paging.

An indirect metric of the CPU cost of paging activity (both PFR and page-out) is the un-captured ratio.

Un-captured ratio is the complement to 1 of the capture ratio that is:

1 – (total CPU accounted to any workload) / total CPU used by the system)

The total CPU accounted to workloads can be obtained by SMF 72 (service class records) while the total CPU used by the whole system is provided by SMF 70.

In a production environment the capture ratio should be around 90% or better. So the un-captured ratio should be less than 10%.
Higher values should be investigated.

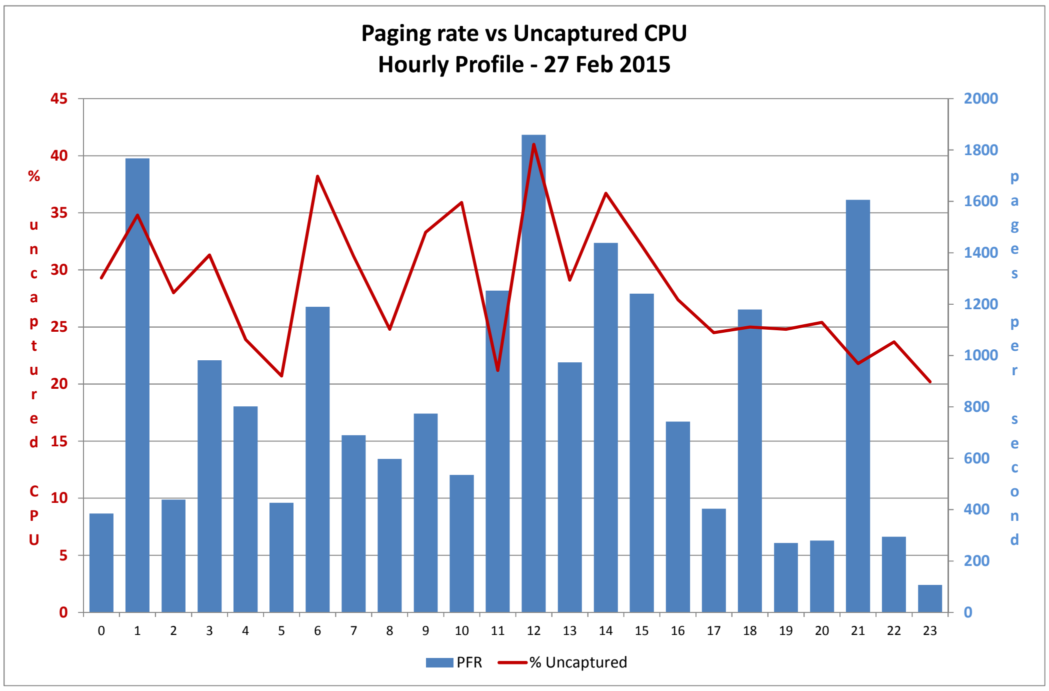


Figure 5

You can see that the un-captured ratio is never lower than 20%.
Paging activity is not the only possible reason of a high un-captured ratio but, in this case, it is probably the main reason.

To reduce or eliminate paging activity you have normally three options:

1. buy more memory;
2. take memory from some other system where memory is abundant;
3. reduce the memory footprint of some workload in the constrained system.

If capacity planning and tuning activities are perfectly performed the only possibility is option 1.

However we don't live in a perfect world so it's very likely that option 2 and 3 could solve the problem.

To exploit option 2 you can start by analysing the PFR data of all the systems running on the same machine. An example is provided in the next table.
You can see that PFR is practically zero all the time. This is a signal that memory is abundant on this system.

DATE	DAY	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
27/02/2015	Fri	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26/02/2015	Thu	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25/02/2015	Wed	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24/02/2015	Tue	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23/02/2015	Mon	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20/02/2015	Fri	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19/02/2015	Thu	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18/02/2015	Wed	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,1	0	0	0,2	0,2	0	0	0	0	0
17/02/2015	Tue	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16/02/2015	Mon	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,1	0,1	0,1	0	0	0	0	0
13/02/2015	Fri	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12/02/2015	Thu	0	0	0	0	0	0	0	0	0	0	0,1	0,5	0	0	0,2	0	0	0	0	0	0	0	0	0
11/02/2015	Wed	0	0	0	0	0	0	0	0	0	0	0	0	0	0,6	0	1,7	0	0	0	0	0	0	0	0
10/02/2015	Tue	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
09/02/2015	Mon	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
06/02/2015	Fri	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
05/02/2015	Thu	0	0	0	0	0	0	0	0	0	0	0	0,1	0	0	0	0	0	0	0	0	0	0	0	0
04/02/2015	Wed	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
03/02/2015	Tue	0	0	0	0	0	0	0	0	0	0,8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
02/02/2015	Mon	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6

Part of the memory in excess can be moved to the other system.
The question is: how much memory can be stolen from this system without impacting its applications performance?

On the other hand if we want to exploit option 3 another question needs an answer: who is using the system memory?

3 Memory usage

As discussed in the previous chapter, when a system is not paging at all what is important to understand is how much memory is used and how much is available.

The answer can be found in the following SMF 71 record fields:

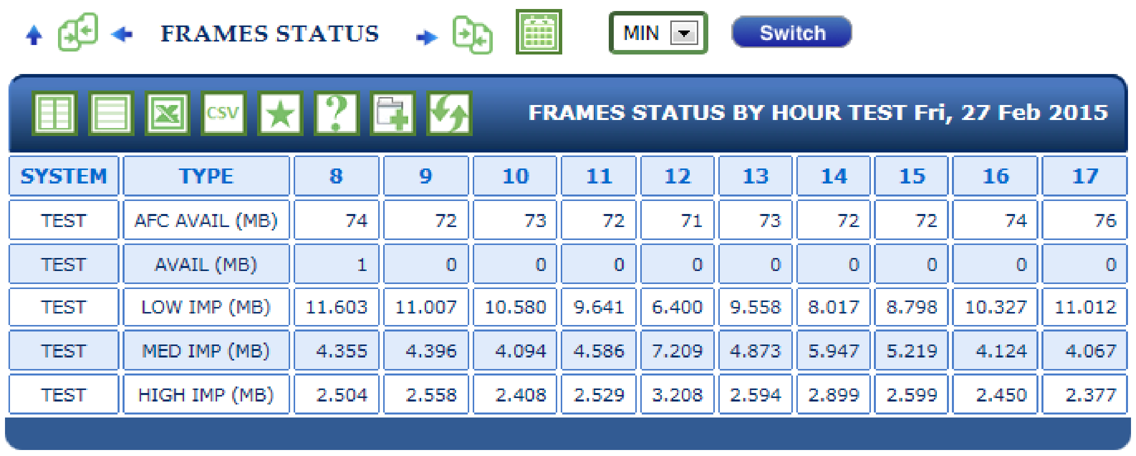
1. SMF71MNF – minimum number of unused central storage page frames;
2. SMF71CAM – minimum number of available central storage frames;
3. SMF71AVF – average number of unused central storage page frames;
4. SMF71CAA – average number of available central storage frames.

You see there are two sets of minimum and average values. So you may wonder what is the difference between "unused" and "available" frames.
The "unused" frames include a certain number of frames that SRM always wants to keep free for his job while the "available" frames don't include them.

Before z/OS 1.9 the low and OK values of the number of frames kept by SRM in the Available Frame Queue was determined solely by the MCCAFCTH OPT parameter.
Now they are dynamically determined by also taking into account the system pageable storage size.
Initial values are:

- RCEAFCLO = MAX (low value in MCCAFCTH, 400, 0,2% of system pageable storage)
- RCEAFCOK = MAX (high value in MCCAFCTH, 800, 0,4% of system pageable storage)

When the number of available frames goes below RCEAFCLO the system replenishes the Available Frame Queue up to RCEAFCOK.



FRAMES STATUS BY HOUR TEST Fri, 27 Feb 2015

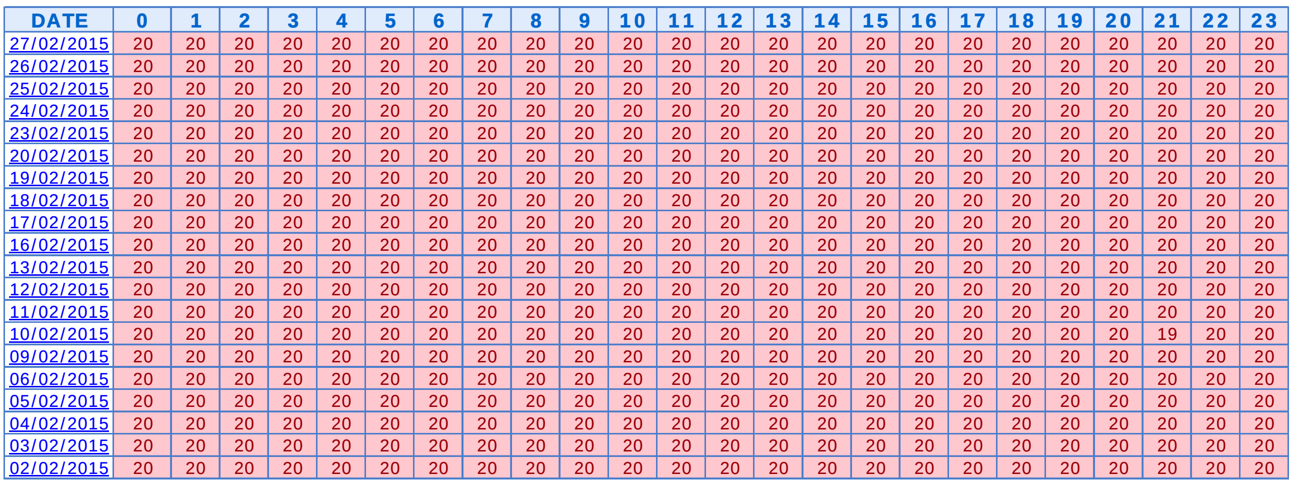
SYSTEM	TYPE	8	9	10	11	12	13	14	15	16	17
TEST	AFC AVAIL (MB)	74	72	73	72	71	73	72	72	74	76
TEST	AVAIL (MB)	1	0	0	0	0	0	0	0	0	0
TEST	LOW IMP (MB)	11.603	11.007	10.580	9.641	6.400	9.558	8.017	8.798	10.327	11.012
TEST	MED IMP (MB)	4.355	4.396	4.094	4.586	7.209	4.873	5.947	5.219	4.124	4.067
TEST	HIGH IMP (MB)	2.504	2.558	2.408	2.529	3.208	2.594	2.899	2.599	2.450	2.377

Figure 7

The difference between AFC AVAIL (SMF71MNF) and AVAIL (SMF71CAM) in the figure above corresponds to the RCEAFCOK value. It tells you how many MB are kept free by SRM (about 70 in this case)

Starting from SMF71CAM, you can estimate memory usage by using the following simple formula:

*memory usage = system memory – (SMF71CAM * 4096)*



DATE	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
27/02/2015	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
26/02/2015	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
25/02/2015	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
24/02/2015	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
23/02/2015	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20/02/2015	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
19/02/2015	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
18/02/2015	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
17/02/2015	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
16/02/2015	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
13/02/2015	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
12/02/2015	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
11/02/2015	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
10/02/2015	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	19	20	20
09/02/2015	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
06/02/2015	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
05/02/2015	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
04/02/2015	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
03/02/2015	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
02/02/2015	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20

Figure 8

The result is pretty obvious in this case; there were practically no available frames in this system so the memory usage, reported in the table above (values higher than 19GB are highlighted), corresponds to the system memory size (20GB).

The next case is much more interesting. It refers to another system in the same site. In the table below we highlighted all the memory usage values higher than 50GB. You can note that only one value trespassed that threshold during a full month.

DATE	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
27/02/2015	37	40	37	37	37	37	38	41	43	45	47	48	46	45	46	45	46	44	45	44	40	39	40	38
26/02/2015	37	37	37	37	37	37	38	41	43	46	48	49	49	47	47	47	47	45	45	45	40	39	40	38
25/02/2015	37	37	36	37	37	37	38	40	42	46	47	48	49	46	48	50	49	45	45	44	40	39	40	38
24/02/2015	37	39	38	38	37	37	37	41	42	46	48	47	48	46	48	48	48	46	45	44	40	39	40	37
23/02/2015	36	40	39	38	36	36	37	40	42	46	47	49	50	46	48	49	47	45	45	44	40	39	39	37
20/02/2015	36	36	36	37	36	36	36	40	41	46	46	49	49	47	46	47	44	42	44	43	40	39	39	38
19/02/2015	35	37	38	36	36	36	36	39	41	44	47	47	48	47	46	47	46	44	44	43	39	38	38	36
18/02/2015	34	36	33	34	34	33	35	38	40	43	45	47	46	44	47	50	48	46	46	45	39	38	38	36
17/02/2015	33	34	35	36	36	36	37	37	39	41	43	43	46	42	43	45	45	42	42	41	37	36	36	34
16/02/2015	32	32	32	32	32	32	33	35	37	39	42	42	42	40	42	44	42	43	43	41	36	35	36	33
13/02/2015	32	33	35	32	31	32	32	35	39	43	44	45	44	44	40	40	39	38	40	39	35	34	35	33
12/02/2015	31	32	33	34	34	33	33	35	39	41	42	45	42	42	45	43	42	40	40	40	35	34	34	32
11/02/2015	31	32	31	32	32	31	32	35	38	44	44	45	46	43	44	43	41	40	41	39	35	34	34	32
10/02/2015	32	32	32	33	32	32	33	35	37	41	44	45	44	41	44	43	42	41	40	40	35	34	34	32
09/02/2015	32	34	35	34	32	32	32	35	37	40	42	43	42	40	41	45	42	40	41	40	35	34	35	32
06/02/2015	31	31	31	32	31	31	32	34	37	39	42	42	42	40	40	39	40	39	39	38	34	34	34	33
05/02/2015	32	32	32	31	31	31	32	35	36	40	42	42	43	39	41	42	41	39	39	39	34	33	33	31
04/02/2015	31	30	30	31	30	30	32	33	35	42	43	43	43	38	42	44	44	39	40	39	35	33	34	32
03/02/2015	30	30	30	30	30	30	30	33	35	40	42	41	42	40	40	41	42	38	38	37	33	32	32	30
02/02/2015	34	33	34	32	31	31	32	35	37	39	42	43	44	40	41	41	41	38	38	37	33	32	32	30

Figure 9

The system memory size is in this case 64GB so part of the memory in excess (about 14GB) could be stolen from this system and donated to the other if needed.

The other important information to know about the system memory is how much is being used, and by who.

Any performance analyst would like to be able to split memory usage by address space. Unfortunately this is not possible because the current metrics provided in SMF 30 are not complete and most of them refer to virtual storage not to real storage; the most interesting are:

- SMF30HVR, high water mark for the number of real storage frames that is used to back 64-bit private storage;
- SMF30HVA, high water mark for the amount of auxiliary storage that is used to back 64-bit private storage; this value is a total of the number of paging data set slots and storage-class memory (SCM) blocks;
- SMF30HVO, amount of 64-bit private storage in bytes that is obtained by this step or job; this includes guarded virtual storage;
- SMF30HSO, amount of 64-bit shared storage in bytes, which this step or job has addressability or access to.

So to understand who is using the system memory the best solution is looking at SMF 72 (or at online monitors).

The R723CPRS field in SMF 72 provides "Total page residency time (1024-microsecond units)" for each service and report class period. Starting from this field you can estimate service and report class memory usage by using the following simple formula:

*service/report class memory usage = R723CPRS * 1024 / RMFINT * 4096*

The graph below refers to another system at a different site. The blue bars is the total amount of memory accounted to all the service classes[1].

[1] We use an aggregated value for all the service classes to make the graph more readable.

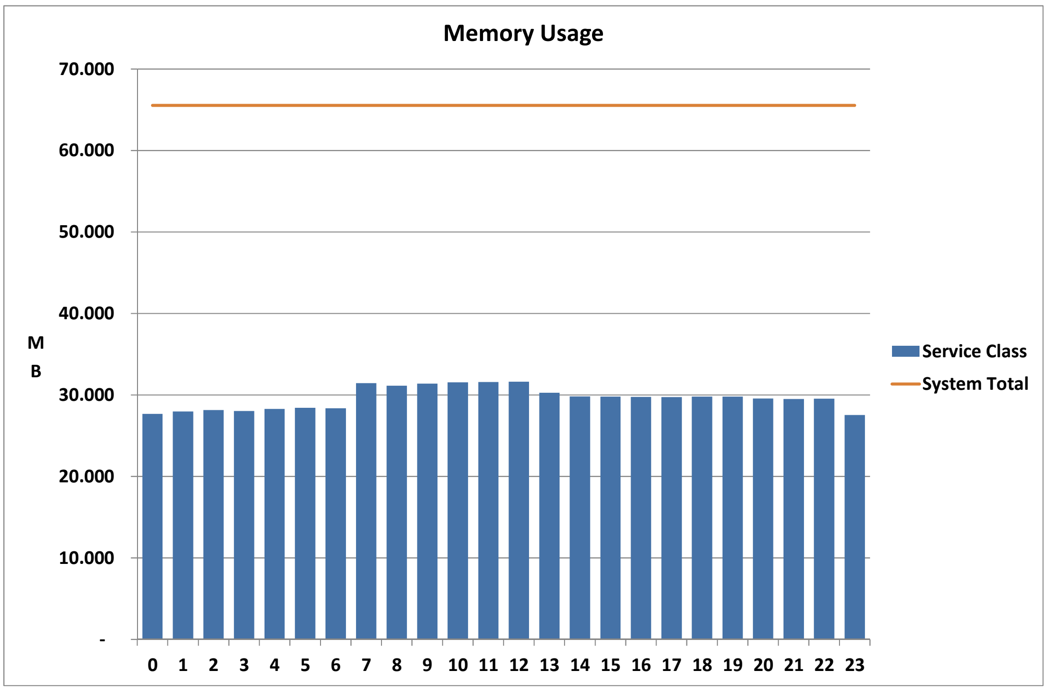


Figure 10

The system assigned memory is 64GB and it is represented by the horizontal red line.

To guarantee good performance to logically swappable address spaces some memory has also to be used in z/OS systems. Unfortunately no direct measurement is available. You can estimate it globally by using the following formula:

LogSwap AS memory usage = Pageable memory usage (SMF71LVS) - Service class memory usage

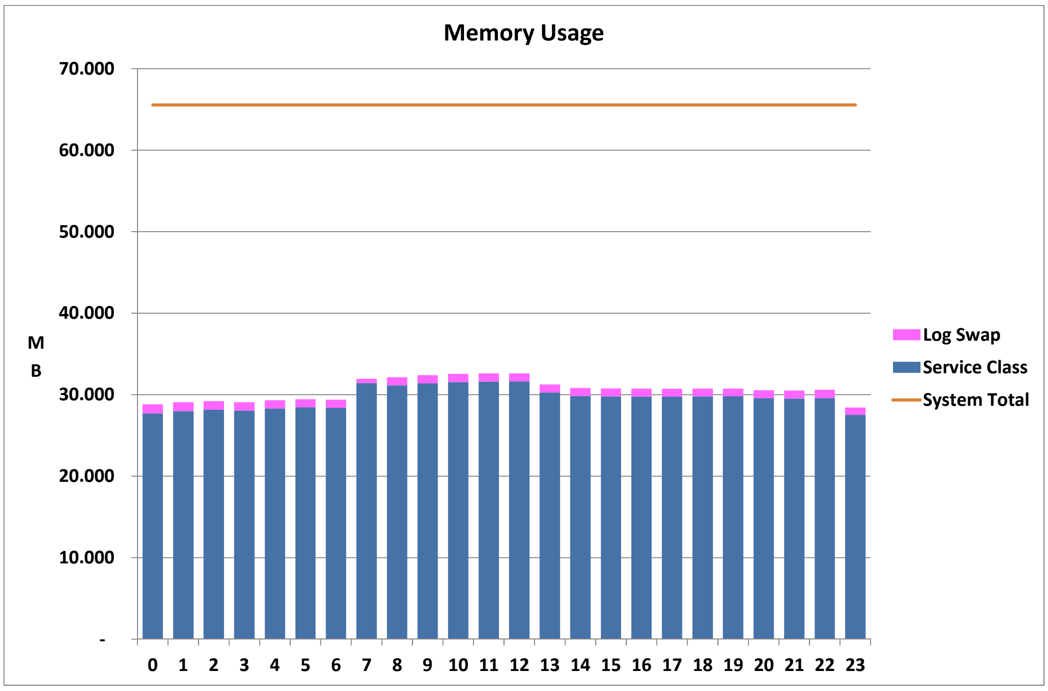


Figure 11

As you can see we're still very far from the total system memory.

What is missing are the frames used by high virtual (64bit) shared and common storage which are not accounted in SMF 72[2].

You can get global usage from the following fields provided in SMF 71:

- SMF71CRA – average number of high virtual common storage frames
- SMF71SRA – average number of high virtual shared storage frames

About 10GB frames are used to back high virtual shared storage (violet bars) while the portion used by high virtual common storage is very small.

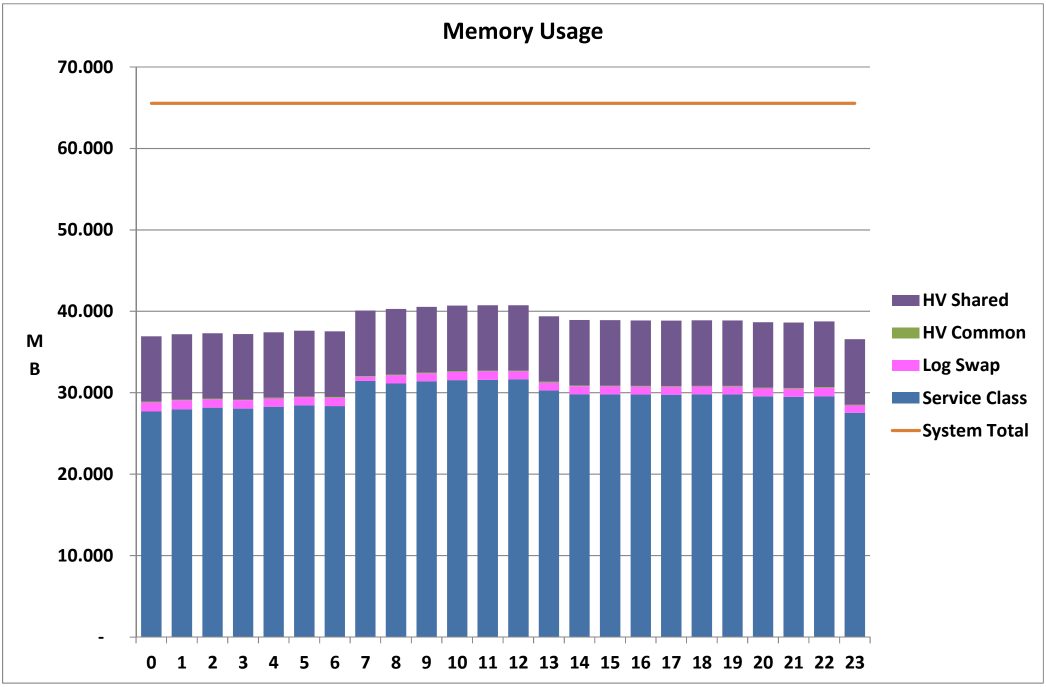


Figure 12

By adding the available frames, more than 20GB on the analysed day, we explained almost all the system memory usage.

[2] A major user of high virtual storage is DB2; many detailed metrics about DB2 memory usage are provided in IFCID 225 and externalized to SMF 100 subtype 4.

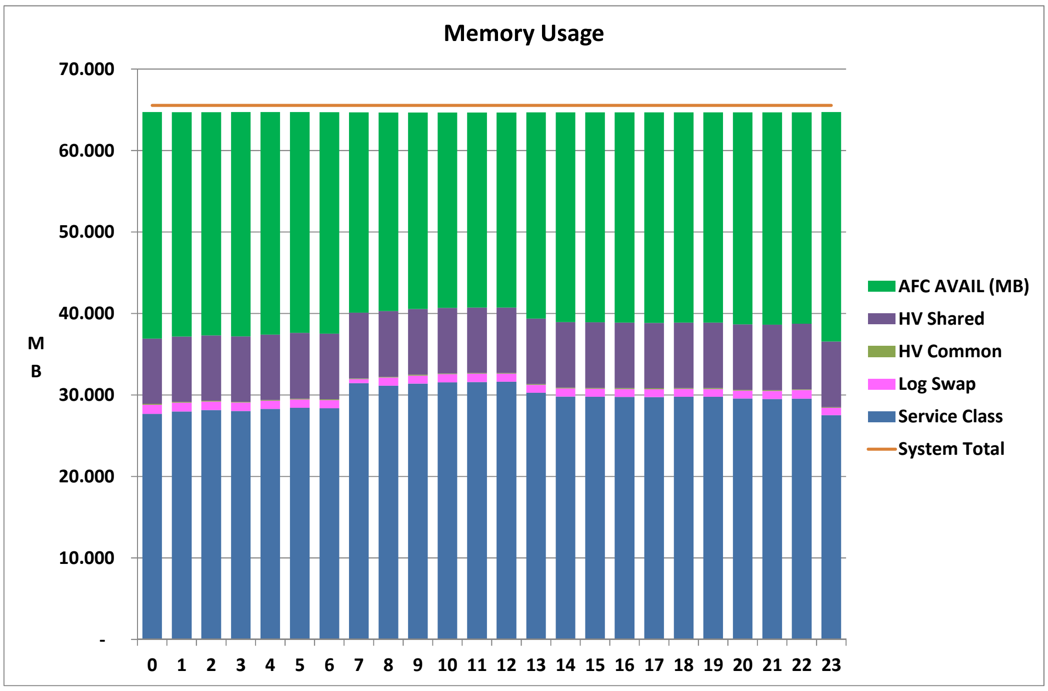


Figure 13

What is still missing? We miss the frames used by common areas below 2GB and the nucleus. Information is provided in SMF 71. They normally account to less than 1GB.

4 Data In Memory

Having a lot of unused memory is generally not positive.
Sometimes it means that your capacity planning has been too generous but in most cases it simply means that you are not exploiting the many Data In Memory (DIM) techniques available in z/OS for years.
The DIM idea is that you should use memory to avoid I/O operations thus improving application performance and reducing costs.

Current storage processors are so powerful that disk performance is very good at most sites. So nobody cares enough about the number of I/Os performed and many sites continue doing far too many avoidable I/Os.
They often forget that, even if the I/O response time is very quick, it will always be much worse than memory and performing I/Os requires expensive CPU cycles.

This is why the old motto which says “The best I/O is no I/O” is nowadays even more valid than before.

To understand if I/O activity is excessive we suggest you use the IOC index.
The IOC is an index, we use in EPV for z/OS, calculated by dividing the AVERAGE DISK I/O RATE by the AVERAGE MIPS USED[3].
In our experience values higher than 3 should be investigated but, generally speaking, the lower the value the better it is.

[3] It includes CPU, zAAP and zIIP MIPS.

In Figure 14 we show the IOC of two production systems at a customer's site. Even if the workload is very similar the average weekly values are quite different.

PRDA shows values generally higher than 3 while PRDB values are always lower than 1.

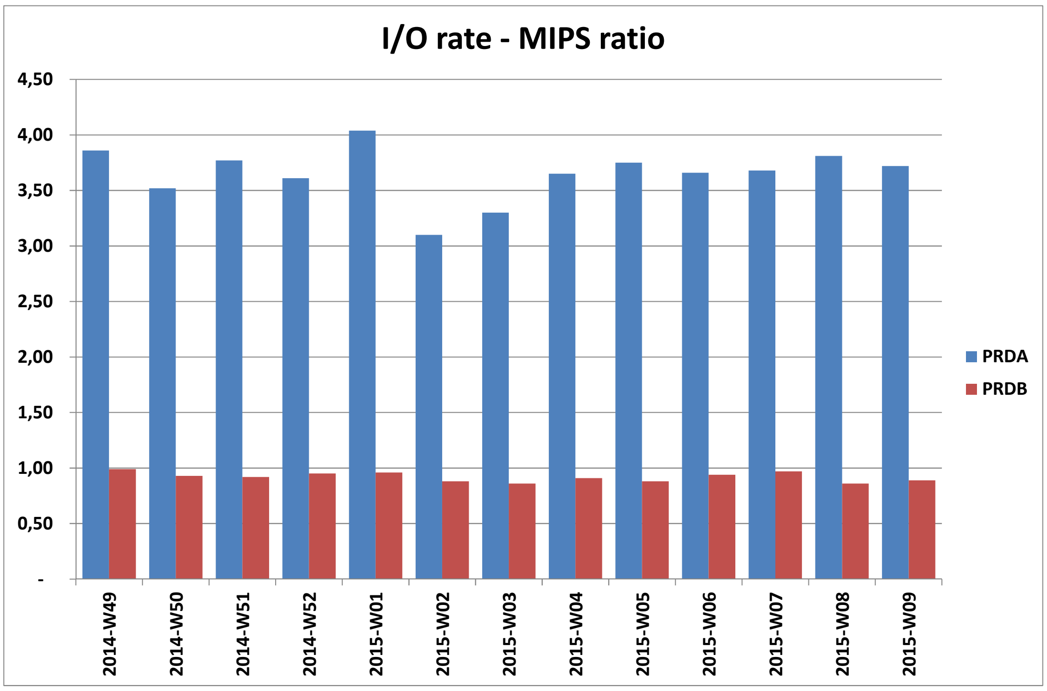


Figure 14

By analyzing the I/O activity in more detail we discovered an intensive load on two disk logical volumes used by IMS. As you can note in the next table all the operations were read and they were performed on very few datasets.

HOUR	SSID	VOLSER	DEVNR	HPAV	UCBS	IORATE	DS ALLOC	% WRITE
8	309	IMS10A	1947	Y	2,1	686	4	0,0
9	309	IMS10A	1947	Y	1,4	1.148	4	0,0
10	309	IMS10A	1947	Y	1,5	1.184	4	0,0
11	309	IMS10A	1947	Y	1,6	1.332	4	0,0
12	309	IMS10A	1947	Y	1,2	873	4	0,0
13	309	IMS10A	1947	Y	1,1	603	4	0,0
14	309	IMS10A	1947	Y	1,3	649	4	0,0
15	309	IMS10A	1947	Y	1,3	1.026	4	0,0
16	309	IMS10A	1947	Y	1,1	622	4	0,0
17	309	IMS10A	1947	Y	1	463	4	0,0
8	412	IMS20A	122D	Y	3,1	1.099	4	0,0
9	412	IMS20A	122D	Y	4,3	1.623	4	0,0
10	412	IMS20A	122D	Y	4,4	1.783	4	0,0
11	412	IMS20A	122D	Y	4,4	1.901	4	0,0
12	412	IMS20A	122D	Y	4,2	1.306	4	0,0
13	412	IMS20A	122D	Y	3,1	985	4	0,0
14	412	IMS20A	122D	Y	3,2	1.041	4	0,0
15	412	IMS20A	122D	Y	4,2	1.628	4	0,0
16	412	IMS20A	122D	Y	3,1	882	4	0,0
17	412	IMS20A	122D	Y	2	656	4	0,0

Figure 15

In the end we discovered that they had forgotten a couple of libraries out of LLA/VLF. After correcting the error most of this activity was completely eliminated.

In our experience this is the most common reason for avoidable I/Os at many sites.
The second most common reason is the DB2 buffer pools size.

According to IBM, most sites have less than 10 GB used by all the buffer pools of a production DB2.
The next table refer to another customer where only one disk logical volume was performing more than 10.000 I/O per second (with peaks over 15.000) continuously during the peak hours.
You can see that only read operations are performed; but in this case they are spread over many datasets.

HOUR	SSID	VOLSER	DEVNR	HPAV	UCBS	IORATE	DS ALLOC	% WRITE
8	325	DB1111	9D0C	Y	9,3	14.696	160	0,0
9	325	DB1111	9D0C	Y	11,9	14.379	125	0,0
10	325	DB1111	9D0C	Y	11,5	13.852	136	0,0
11	325	DB1111	9D0C	Y	15	16.619	126	0,0
12	325	DB1111	9D0C	Y	9,7	11.784	166	0,0
13	325	DB1111	9D0C	Y	7,2	9.323	220	0,0
14	325	DB1111	9D0C	Y	13,2	11.294	200	0,0
15	325	DB1111	9D0C	Y	11,7	15.884	203	0,0
16	325	DB1111	9D0C	Y	5,8	7.324	225	0,0
17	325	DB1111	9D0C	Y	3,3	3.622	197	0,1

Figure 16

We discovered that this volume hosted some of the most accessed DB2 tables of this production environment and that all this read operations were needed to continuously loading data in two DB2 buffer pools which were not big enough.

It’s worth saying that there were more than 10 GB memory unused on this system and that the total size of these two buffer pools was about 1 GB; so we suggested doubling their size and measuring the results[4].

Other common reasons for excessive I/Os are:

- Bad DB2 access paths;
- Bad SQL;
- Small WMQ buffer pools;
- Small CICS VSAM LSR buffer pools;
- Small Java heaps.

At this point we need to answer a very difficult and important question: how much CPU does an I/O cost?

We did a study, some years ago, estimating 1 MIPS every 50 I/O per second for directory reads (libraries not in LLA). In this case by saving 1.000 I/O per second you can estimate a 20 MIPS saving[5].

[4] In DB2 V11 a new function (buffer pool simulation) is available to determine the right buffer pool s size.

By analyzing the result of a recent IBM study[6] we estimated about 35 CPU microseconds (on a 2827-712) per DB2 synchronous I/O. In this case by saving 1.000 I/O per second you can save 41 MIPS.

The bottom line is that you should investigate any I/O excessive workload and try to reduce it as much as possible.

The best starting point are the DISK TOP I/O INTENSITY views in EPV for z/OS.

As an alternative you can collect, combine and analyze the raw information provided in:

- SMF 74 subtype 1;
- SMF 74 subtype 5;
- SMF 42 subtype 6.

[5] See the “The Best I/O is no I/O” white paper published in the EPV Newsletter.
[6] See “System z: Advantages of Configuring More Memory for DB2 Buffer Pools” – February 2015.

5 Exploiting Large Pages

Information provided in this chapter is a complement and an update to what discussed in the Large Memory Pages white paper published in two parts in the April and May 2013 EPV Technologies Newsletter.

Ever since the days when z/OS was called MVS, virtual memory has always been managed in 4096 byte pages. However, with the advent of z/OS 1.9 and z10 machines, 1MB pages can be used, whilst 2GB pages are also supported starting with z/OS 1.13 and zEC12.

The reason for this fundamental breakthrough is the exploitation of 64–bit architecture; it is now possible to create huge z/OS address spaces: up to 16 ExaBytes of virtual memory. In order to back this virtual memory, the IBM z13 is able to provide up to 10 TB of real memory which is also, by the way, much cheaper than previously.
In 31-bit mode the maximum address space size was 2GB; it could be mapped by using 256*2.048 (524.288) 4K pages.
In 64-bit mode to map all the address space virtual memory 256*2.048*2.048*2.048*2.048 (4.503.599.627.370.500) 4K pages would be required.

It’s intuitive that managing such big address spaces with so many small 4K pages would not be very efficient; so to improve performance and to reduce CPU consumptions of memory-intensive workloads (e.g. DB2 and WebSphere applications), it is possible and advisable to use large memory pages.

5.1 Virtual to real address translation

A 31-bit address space is mapped by using 2.048 segments of 1MB size; each segment is mapped by using 256 pages of 4K size.

So a 31-bit virtual address is written as follows:
- 11 bits to point to the appropriate page table entry in the segment table;
- 8 bits to point to the appropriate page in the page table;
- 12 bit to point to the appropriate point of the page.

A 64-bit address space is mapped by using three additional levels of translation tables, called region tables, on top of the segment table (see schema n Figure 17):
- region third table (R3T), pointing to 2048 segment tables,
- region second table (R2T), pointing to 2048 R3T,
- region first table (R1T), pointing to 2048 R2T.

So a 64-bit virtual address is written as follows:
- 11 bits to point to the appropriate R2T in R1T;
- 11 bits to point to the appropriate R3T in R2T;
- 11 bits to point to the appropriate segment table in R3T;
- 11 bits to point to the appropriate page table entry in the segment table;
- 8 bits to point to the appropriate page in the page table;
- 12 bit to point to the appropriate point of the page.

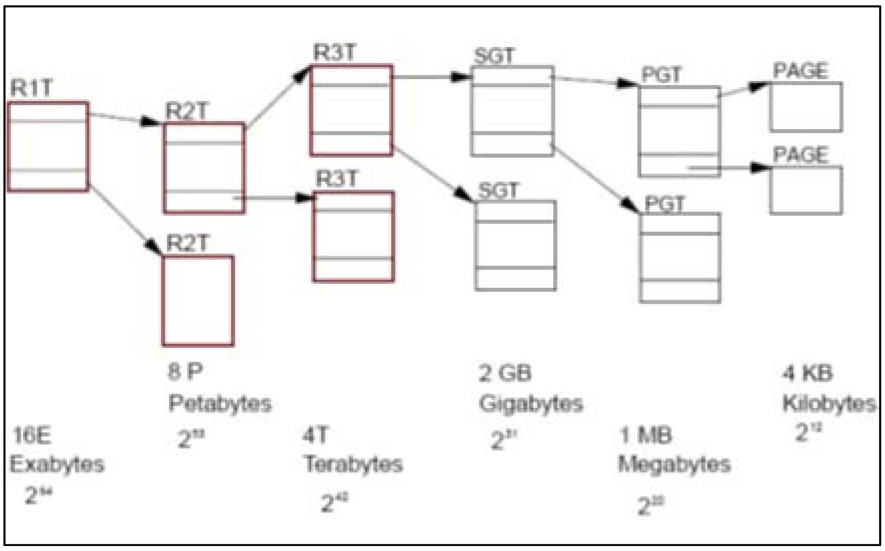


Figure 17[7]

Note that Real Storage Manager only creates the additional levels of region tables when it is necessary to back the address space getmained virtual storage. So R3T is created only if virtual memory is used above the 2 GB bar. In the same way R2T and R1T are created only if virtual storage addresses greater than 4 TB and 8 PB, respectively, are getmained.

A virtual address is converted to a real address by using the Digital Address Translation hardware or DAT included in each PU of the IBM machines.
During this translation, DAT should access all the address space mapping tables, described above, in real memory to complete the process (up to 5 real memory accesses might be needed for address spaces exploiting 64-bit addressing).

To avoid these accesses the Transaction Lookaside Buffer was introduced many years ago; TLB is a fast array memory within each PU. After translating using the address space tables, DAT keeps the relation page/frame in a TLB entry and the next time, before accessing real memory to go through a table translation, DAT inspects the TLB looking for the referred page. When there is a hit, the translation process is much faster.

To improve performance more TLBs were added in the last few years. One of them is dedicated to 1MB pages another one to 2GB pages.

5.2 Measuring TLB effectiveness

Direct measurements of TLB effectiveness are provided in the extended counters collected in SMF 113[8]. The relevant extended counters for TLB analysis of z13 machines are[9]:

- E129 – Translation entry written to the Data TLB1
- E130 – Data TLB1 miss cycles
- E131 – Translation entry written to the Data TLB1 for 1MB pages

[7] From “ABCs of z/OS System Programming - Volume 1”
[8] All the SMF 113 counters are collected in the EPV for z/OS data base.
[9] Extended counters meaning is different depending on the machine type.

- E132 – Translation entry written to the Data TLB1 for 2GB pages
- E134 – Translation entry written to the Instruction TLB1
- E135 – Instruction TLB1 miss cycles
- E137 – Translation entry written to the Page Table Entry in TLB2
- E138 – Translation entry written to the Common Region Segment Table Entry in TLB2 for 1MB page
- E139 – Translation entry written to the Common Region Segment Table Entry in TLB2

By looking at E131 and E132 you can evaluate TLB miss activity for 1MB and 2GB pages.

A graph tracking the percentage of CPU cycles used to satisfy TLB1 misses over the total CPU cycles used and the average number of CPU cycles needed to satisfy one TLB1 miss for a production system is shown in Figure 18. The graph refers to a zEC12 machine

You can see that the CPU cycles used to satisfy TLB1 misses accounts for 5-6% of the total CPU cycles used, while between 20 and 30 CPU cycles are needed, on average, to satisfy a TLB1 miss.

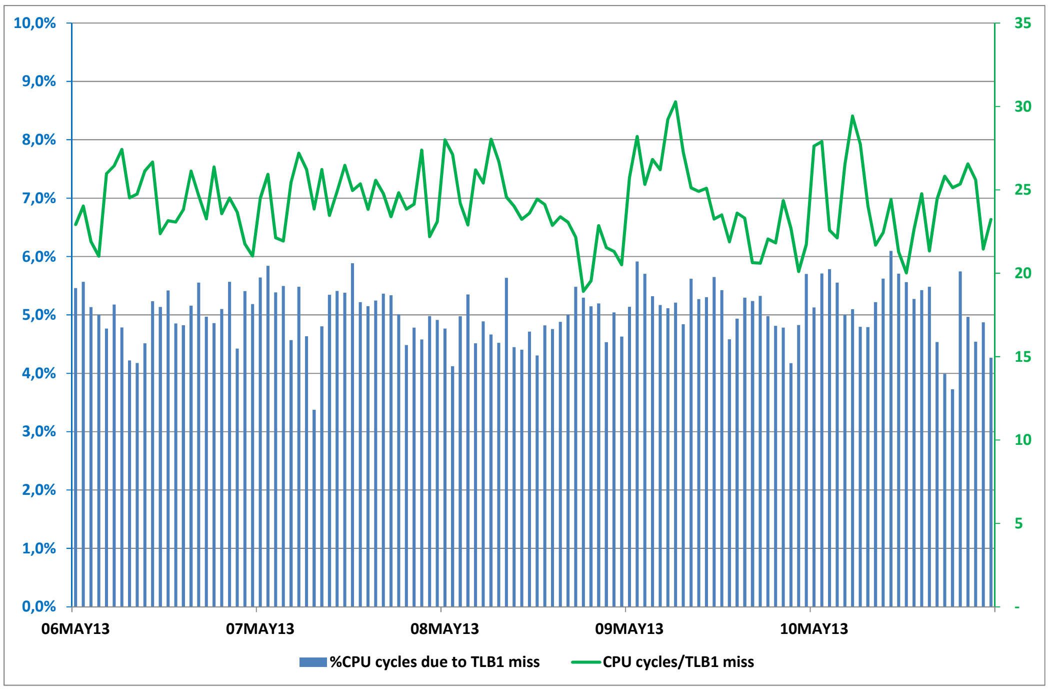


Figure 18

When using 1MB and 2GB large memory pages:

- one large memory page (1MB) TLB entry covers 256 4K pages;
- one large memory page (2GB) TLB entry covers 524.288 4K pages;
- page tables are not needed for large memory pages; they are pointed directly from the segment table;
- segment tables are not needed for large memory pages; they are pointed directly from the R3T table.

The final advantage is the reduction of the CPU cycles needed for virtual address translation.

5.3 Enabling large pages

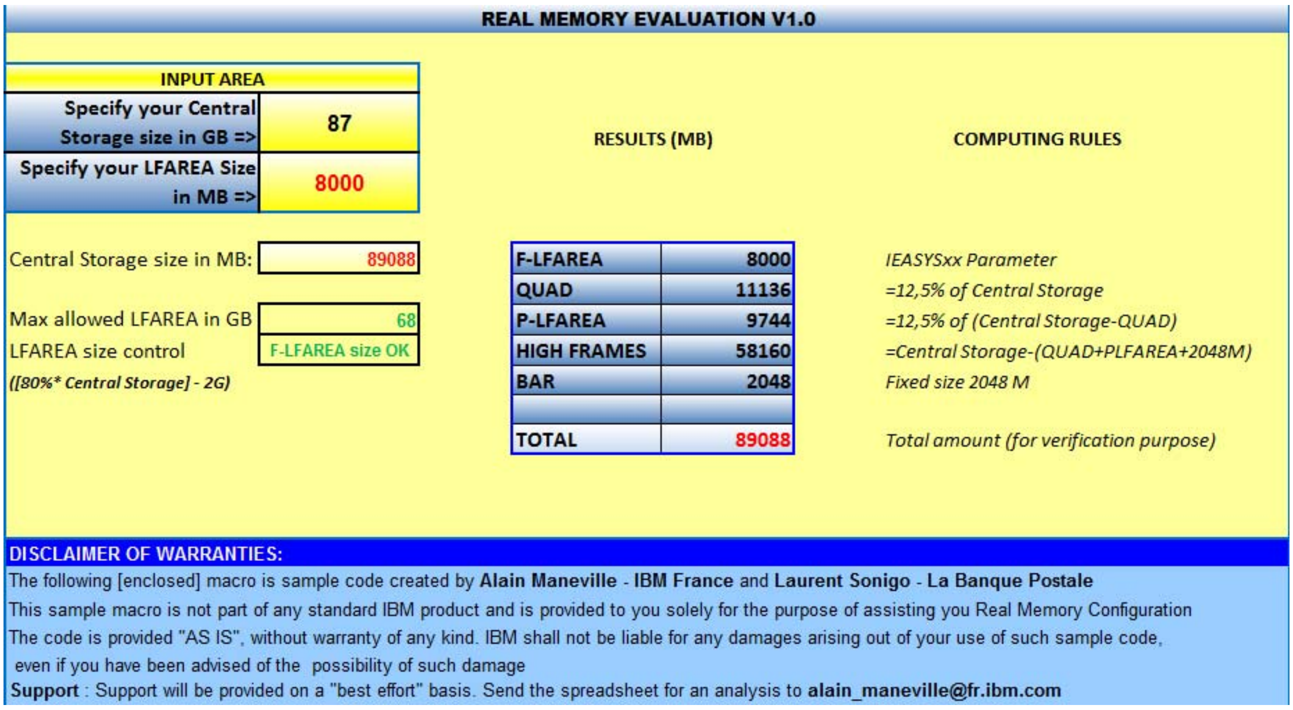
First implementation only allowed 1MB fixed pages. Then the possibility to use pageable 1MB memory pages was added but only if Flash Express is enabled. Starting with zEC12 2GB fixed pages can also be used.

To enable fixed large pages you have to define the amount of real storage to be used for them by setting the LFAREA (Large Fixed Area) parameter in IEASYS. LFAREA can use up to 80% of the online storage available at IPL minus 2 or 4 GB.

LFAREA parameter syntax can be complex especially if you want to reserve memory both for 1MB and 2GB pages. Note that if you simply code: LFAREA=xG you are reserving xGB of real memory for 1MB pages only; no memory will be reserved for 2GB frames[10].

A PLAREA (Pageable Large Area) is created by default if you run z/OS 2.1 or 1.13 with RSM Enablement Offering and you run on a SCM capable machine; even if you don’t really have SCM (Flash Memory).
You can disable PLAREA by setting PAGESCM=NONE in IEASYSxx but you have no control over the PLAREA size

Alain Maneville (IBM France) and Laurent Sonigo (Le Banque Postal) provided a spreadsheet that you can use to understand how z/OS real memory is going to be carved up when you IPL on an SCM-capable CPC.
The spreadsheet (see next figure) is available from Cheryl Watson web site at:
www.watsonwalker.com/freetools.html



REAL MEMORY EVALUATION V1.0

INPUT AREA	
Specify your Central Storage size in GB =>	87
Specify your LFAREA Size in MB =>	8000

Central Storage size in MB:	89088
Max allowed LFAREA in GB	68
LFAREA size control	F-LFAREA size OK
([80% Central Storage] - 2G)*	

RESULTS (MB)		COMPUTING RULES
F-LFAREA	8000	*IEASYSxx Parameter*
QUAD	11136	*=12,5% of Central Storage*
P-LFAREA	9744	*=12,5% of (Central Storage-QUAD)*
HIGH FRAMES	58160	*=Central Storage-(QUAD+PLFAREA+2048M)*
BAR	2048	*Fixed size 2048 M*
TOTAL	89088	*Total amount (for verification purpose)*

DISCLAIMER OF WARRANTIES:
The following [enclosed] macro is sample code created by **Alain Maneville - IBM France** and **Laurent Sonigo - La Banque Postale**
This sample macro is not part of any standard IBM product and is provided to you solely for the purpose of assisting you Real Memory Configuration
The code is provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of such sample code,
even if you have been advised of the possibility of such damage
Support : Support will be provided on a "best effort" basis. Send the spreadsheet for an analysis to **alain_maneville@fr.ibm.com**

Figure 19

[10] Please refer to “MVS Initialization and Tuning Reference” for more details.

When the amount of 4K real storage frames is low RSM will break up 1MB frames (pageable, quad, fixed in this order) and use them.

Please note that, depending on z/OS level, maintenance and settings[11], LFAREA pages may or may not be included in the available frames; only when the available frame count is lower than RCEAFCLO (see Chapter 3), RSM starts to break up 1MB pages; a lot of paging may be already happening at this point.

When the amount of large real storage frames is exhausted RSM starts “coalescing” 4K pages. Coalescing is a very CPU intensive task. So you should carefully size the LFAREA, to host both fixed and at least a good part of the pageable large pages, to avoid it happening.

5.4 Large page exploiters

As a general rule large pages may provide performance value to long-running memory access-intensive applications so it’s not surprising that the exploiters’ list is continuously growing.
Here are the most important of them:

- z/OS nucleus (since z/OS 1.12);
- DB2 buffer pools (since V10) when the PGFIX=YES parameter is specified[12];
- JVM can use large memory pages (both for code-cache and heap) by specifying the –Xlp option; more recent JVM versions will automatically use large memory pages if they are available;
- ADABAS;
- DB2 executable code (since V11);
- IMS CQS (since V12);
- various IMS pools (since V13);
- IMS OLDS (since V13);
- System Logger (since z/OS 1.13);
- USS.

5.5 Measuring Large Pages

We expected that IBM would improve the metrics provided to measure large pages activity. Unfortunately almost nothing has changed in the last two years:

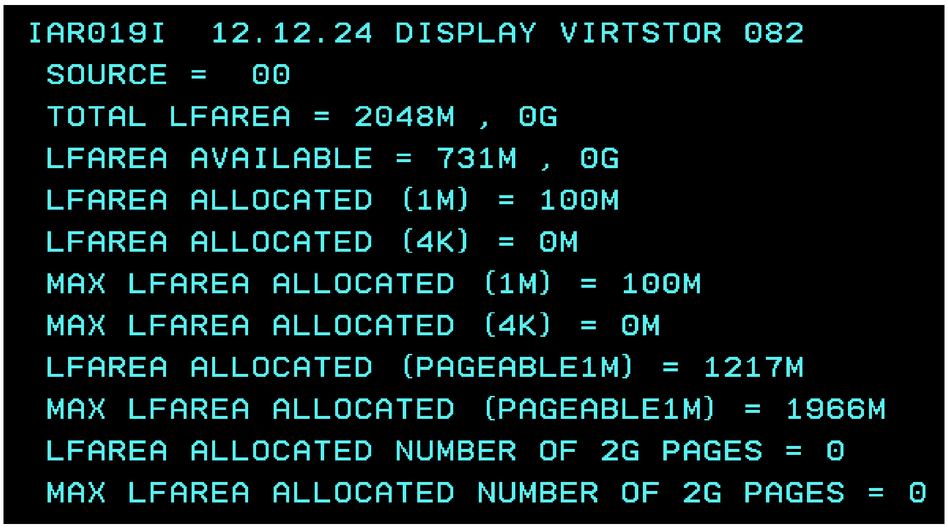
- general information only about 1MB memory pages are provided by RMF Monitor I and collected in SMF 71 records; this information is also reported in EPV for z/OS;
- details about 1MB memory pages exploiters are only provided by RMF Monitor III;
- no information is available about coalescing and PLAREA

[11] With INCLUDE1MAFC, LFAREA pages are included in AFC; it will avoid paging if there are free large pages in LFAREA.

[12] The buffer pool PGFIX(YES) option has been available since DB2 V8; it allows you to (almost) permanently fix a pool's buffers in memory to save CPU by eliminating the need to fix in memory and then release a buffer every time a page is read in from disk or is written out to disk. In DB2 V10, 1MB page frames can be used for page-fixed buffer pools providing additional benefits in terms of CPU savings and performance. Large memory pages are non-pageable in this case.

The only information available for 2GB memory pages are:

- an extended counter added to SMF 113 (see Chapter 5.2);
- the total and maximum number of 2GB pages in LFAREA provided by the DISPLAY VS,LFAREA command (see figure below).



```
IAR019I  12.12.24 DISPLAY VIRTSTOR 082
 SOURCE =  00
 TOTAL LFAREA = 2048M , 0G
 LFAREA AVAILABLE = 731M , 0G
 LFAREA ALLOCATED (1M) = 100M
 LFAREA ALLOCATED (4K) = 0M
 MAX LFAREA ALLOCATED (1M) = 100M
 MAX LFAREA ALLOCATED (4K) = 0M
 LFAREA ALLOCATED (PAGEABLE1M) = 1217M
 MAX LFAREA ALLOCATED (PAGEABLE1M) = 1966M
 LFAREA ALLOCATED NUMBER OF 2G PAGES = 0
 MAX LFAREA ALLOCATED NUMBER OF 2G PAGES = 0
```

Figure 20

6 Conclusions

The amount of available mainframe memory continues to increase with every new IBM machine generation and z/OS releases while memory GB cost is decreasing.

Buying more memory and using it to improve performance and reduce CPU usage is always a good deal. Remember that you pay software costs every month but you pay memory only once.

Exploitation of large memory pages (1MB or 2GB) may provide additional savings.
Unfortunately current metrics and tools provide incomplete information to analyze 1MB pages and almost nothing for 2GB pages.