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\(^1\). In our experience values higher than 3 should be investigated but, generally speaking, the lower the value the better it is.

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.

\(^{1}\) It includes CPU, zAAP and zIIP MIPS.
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.

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

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.

![Figure 16](image)

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.

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.

By analyzing the result of a recent IBM study 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.

---

2 In DB2 V11 a new function (buffer pool simulation) is available to determine the right buffer pool size.
3 See the “The Best I/O is no I/O” white paper published in the EPV Newsletter.
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 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 $256 \times 2^{10}$ (524,288) 4K pages.

In 64-bit mode to map all the address space virtual memory $256 \times 2^{10} \times 2^{10} \times 2^{10} \times 2^{10}$ (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 in 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.
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\(^5\). The relevant extended counters for TLB analysis of z13 machines are\(^7\):

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

---

\(^{5}\) From “ABCs of z/OS System Programming - Volume 1”

\(^{6}\) All the SMF 113 counters are collected in the EPV for z/OS data base.

\(^{7}\) Extended counters meaning is different depending on the machine type.
EPV Technologies

- 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.

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.

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

---

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\(^9\), 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\(^10\);
- 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

---

\(^9\) With INCLUDE1MAFC, LFAREA pages are included in AFC; it will avoid paging if there are free large pages in LFAREA.

\(^10\) 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.