Performance Analysis for VM Applications

SHARE August 1998 - Session 9227

Bill Bitner
IBM Endicott
1701 North St.
Endicott, NY 13760
607-752-6022
bitner@vnet.ibm.com
USIB1E29 at IBMMAIL

Disclaimer

The information contained in this document has not been submitted to any formal IBM test and is distributed on an "as is" basis without any warranty either express or implied. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environment do so at their own risk.

In this document, any references made to an IBM licensed program are not intended to state or imply that only IBM's licensed program may be used; any functionally equivalent program may be used instead.

Any performance data contained in this document was determined in a controlled environment and, therefore, the results which may be obtained in other operating environments may vary significantly.

Users of this document should verify the applicable data for their specific environments.

It is possible that this material may contain references to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country or not yet announced by IBM. Such references or information should not be construed to mean that IBM intends to announce such IBM products, programming, or services.

Should the speaker start getting too silly, IBM will deny any knowledge of his association with the corporation.

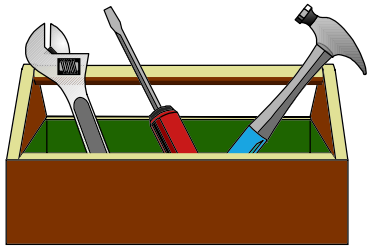
Trademarks

The following are trademarks of the IBM Corporation:

- IBM
- OfficeVision
- VM/ESA

Introduction

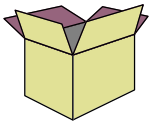
- To performance analysis everything looks like a performance tool
- various tools documented/undocumented
- goal of better understanding the application or program
- Look at a subset of the tools that exist



I have been accused of being one-minded in that I cannot think about anything other than performance. That applies to looking at tools and utilities as well. Whenever a new command or function is created, I find myself asking "What does this have to do with performance?".
There are a number of tools with varying degrees of documentation, that were not created for use in performance analysis. However, they can be used to gain a better understanding of an application or program. In this presentation, we will look at a subset of these tools and discuss how they can be applied to performance analysis.

What do we want to know?

- How many, when, or how much?
  - Processor Resources
  - Storage
  - I/O
  - System Services
    - CMS
    - CP
    - Servers

What are some of the things we care about when it comes to performance analysis of applications or solutions? The first three items most commonly mentioned are processor, storage, and I/O. The fourth item we should remember is system services. These could include storage management calls, I/O or communication calls, authorization checking, etc. .

CP INDICATE USER * EXP

- snap shot: requires taking delta of two indicate commands
- Covers processor, I/O, and Storage
- Use expanded version
  - avoids anomalies
  - better storage view

The CP INDICATE command gives a snap shot of various resource utilization numbers. Since it is only a snap shot, we often need to bracket what we are measuring with INDICATE commands and then take the deltas of the two results. I recommend use of the EXPanded option. It avoids some anomalies and also provides additional views of storage usage.

```
Userid=BITNER  Mach=XC  V=V  Attached xstore=NONE
Iplsys=CMS  Devnum=32
Spool:  Reads=536  Writes=133
Owned spaces:  Number=1  Owned size=103M
  Primary space:  ID=BITNER:BASE  PRIVATE
    Defined size=128M  Address limit=814M
  Private spaces:  Number=1  Owned size=103M
    Pages:  Main=929  Xstore=0  Dasd=0
            Locked=1  WS=908  Reserved=0
  Shared spaces:   Number=0  Owned size=0
    Pages:  Main=0  Xstore=0  Dasd=0
            Locked=0
  Private paging:
    Xstore:  Reads=0          Writes=0          Migrates=0
    Dasd:    Reads=0          Writes=0
  Shared paging:
    Xstore:  Reads=0          Writes=0          Migrates=0
    Dasd:    Reads=0          Writes=0
CPU 00: Ctime=0 04:35:22  Vtime=0 00:00:17 Ttime=0 00:00:19
        Rdr=17634  Prt=439  Pch=901  IO=5516
```

This is an example of the output returned by the INDICATE USER command with the EXPanded option. Note that output would be slightly different if the virtual machine was configured differently, such as with vectors or multiple virtual processors.

SET RDYMSG LMSG

- Gives processor usage with better granularity
- T= Virtual / Total processor time

example

Ready; T=1.40/1.54 18:04:35

Another way of getting some information on processor usage is by changing the format of the CMS ready message to long message. The INDICATE USER provides processor time rounded to the nearest second, while long message displays processor time in hundredths of seconds. The downside is you only get this information at the ready prompt.

CP QUERY TIME

- Granularity of long ready message

```
q time
TIME IS 18:46:03 EST SATURDAY 02/14/98
CONNECT= 05:12:43 VIRTCPU= 000:21.09
                  TOTCPU= 000:24.00
Ready; T=0.01/0.01 18:46:03
```

The CP Query Time command gives the same granularity as the long ready message, but can be issued within a program. The Query Time command displays the total accumulated processor time. Therefore, like the Indicate command, you would need to issue two Query Time commands and compute the delta for a given function.

CP TRACE command

- Use with COUNT option to just count instructions.
- Use with DIAG to count I/Os via diagnose x'A4' and x'A8'
- Could output as input to profiling tools for both storage and pathlengths
- Can slow down the application, depending on what is being traced.
- Tracing IUCVs or diagnose x'68' for server machine requests.

The CP Trace (or Per) command can gather information about the virtual pathlength. Especially helpful for performance analysis is the COUNT option. Instead of gathering all the data, only the occurrences are counted. Output from the Trace command could also be used as input to profiling tools for both storage and pathlengths. Note that tracing can slow down an application severely.

CMS EXECMAP command

- For EXECLOADed execs
- Gives a running usage count (2 invocations)
- Also gives segment information

```
execmap example exec
Name         Type         Usage    Records      Bytes
EXAMPLE      EXEC             2         10        344
Ready;
```

To get a feel for what execs are being used, you can issue the CMS EXECMAP command to gather statistics on execs that have been execloaded. The usage count is a running count, so again you would need to issue multiple commands and compute the deltas.

CMS RTNMAP Command

- Running usage count for CSL loaded CSL routines

```
rtnmap *
Alias    Name     Library  UseCount LoadAddr     Size
-------- -------- -------- -------- -------- --------
DMSCOMM  DMSCOMM  VMLIB         416 20C09F38        0
DMSCLOSE DMSCLOSE VMLIB         349 20C14740        0
DMSPOPWU DMSPOPWU VMLIB         118 20C19450        0
DMSOPEN  DMSOPEN  VMLIB         349 20C11CD8        0
DMSPUSWU DMSPUSWU VMLIB         118 20C199D0        0
DMSREAD  DMSREAD  VMLIB         871 20C094E0        0
DMSEXIFI DMSEXIFI VMLIB          67 20C0DF40        0
VMTCPDT  VMTCPDT  VMMTLIB        67 0123B888        0
DMSERP   DMSERP   VMLIB          34 20C83400        0
DMSCALLR DMSCALLR VMLIB          64 20C83360        0
VMEVCR   VMEVCR   VMMTLIB         0 011F83C0        0
Ready;
```

The CMS RTNMAP command is similar to the EXECMAP command, except it provides information on CSL routines. In the example above, you see a size of 0 for all of these routines. A size of 0 indicates that they are running from a saved segment, which is good.

CMS Storage Utilities

- STDEBUG
  - monitor OBTAIN and RELEASE requests
- STORMAP
  - maps allocated and unallocated storage within virtual machine
  - EXTSET and FILE options
- SUBPMAP
  - maps subpools and associated storage within your virtual machine
  - EXTSET and FILE options

We will now move on to CMS storage analysis. There are a series of storage analysis utilities which have their roots from former VMer, Steve Jones, affectionately known as Frodo. There is a great deal of function in these utilities, but we'll only focus on a small portion which allow you to: monitor storage obtain or release requests and map storage. Some of these utilities support the EXTSET option to capture information midstream in program execution. The EXTSET option allows you to delay the capture of information until the given code external interrupt occurs. In addition, some of the commands support the FILE option so that data is captured to disk for analysis.

STDEBUG Example

```
stdebug USER (obtain release con
 thrasher 10 2 1
 19:19:41  * MSG FROM BITNER  :
    OBTAINED 0000A000 07EFE000 USER     000100E2
 19:19:43  * MSG FROM BITNER  :
    RELEASED 0000A000 07EFE000 USER     000101AA
Ready; T=0.01/0.01 19:19:43
```

- Note subpool name in capitals
- Provides
  - how much/where storage (A000)
  - address of call (100E2, 0101AA)

STDEBUG can monitor obtain and release requests. In the above example, we are monitoring obtain and release requests to the USER subpool with the output directed to the console. Then the Thrasher program is written. There is one request to obtain storage and one to release storage. And it is good to note that both are for the same amount of storage x'A000' bytes. Note that the subpool (USER) is case sensitive.

SUBPMAP Example

```
subpmap

Subpool   Key   Anchor         Full      Part
DMSINTSP  F0    07FFFB44       1487         0
DMSBLOKN  F0    07FFFB74        337        31
NUCLEUS   F0    07FFFBD4        180        49
DMSIUCV   F0    07FFFE74          1         1
DMSSVQUS  E0    07FFFE44          1         0
DMSCPIC   E0    07FFFE14          0         0
DMSLICMD  F0    07FFFDE4          0         0
DMSDCSYS  F0    07FFFD84       2304         0
Ready; T=0.01/0.01 19:31:09
```

Above is an example of the SUBPMAP command. It shows how many pages are part of each of the subpools and of those pages, how many are fully allocated and how many are only partially allocated. This can be used in peeling onion layers. If we suddenly notice an increase in virtual storage requirements, we can issue the SUBPMAP and get an idea of which subpool or trype of activity is worth investigating further. By comparing the partial and full page count, we can get an idea on storage fragmentation.

STORMAP Example

```
stormap (sub DMSIUCV all
                                  Storage Map
                                  ------- ---

VMSIZE      NUCALPHA    NUCSIGMA    NUCOMEGA    NUCPHI       NUCCHI
08000000    00F00000    00F54D60    01400000    01000000     013016C8

                          Unallocated Free Storage Queue
                          ----------- ---- ------- -----

      <16Mb                      >16Mb
Total       Largest        Total       Largest      Total Unallocated
00920000    0087A000       06580000    05CDE000     06EA0000

Address Range: 00000000 - 07FFFFFF

Subpool  Start     End       Bytes   Pages  Key  Attributes
DMSIUCV  00EF8000  00EF8267  00000268    p  F0   UNALLOC
DMSIUCV  00EF8268  00EF8FFF  00000D98    p  F0   ALLOC GLOBAL SYSTEM
DMSIUCV  07F3D000  07F3DFFF  00001000    1  F0   ALLOC GLOBAL SYSTEM
Ready; T=0.01/0.01 19:38:51
```

If we found a suspect subpool with the SUBPMAP command we can then issue the STORMAP command to see what pages make up that subpool, We can then go even further by displaying storage at the given locations.
You can also get information such as the largest available contiguous area which can highlight fragmentation problems.

Rita

- On samples disk
- Replace "PIPE" with "RITA"
- Shows processor time associated with various stages, plus pipe overhead for management
- See various papers and info on http://pucc.princeton.edu/~pipeline/

Rita is a wonderful utility from Melinda Varian of Princeton University. It is shipped with VM/ESA on the samples disk. It can be used to analyze a pipeline to understand how processor time is associated with each stage. To use it, simply replace "PIPE" with "RITA". More information is available at the website listed in the foil.
A word of caution, while some stages are more expensive than other stages, the overhead is greatly impacted by how many records are being processed in a given stage. We will look at an example on the next foil.

Rita Example

```
rita < bitner netlog a | spec 31-45 2 | sort count
     | locate /MORRISDL/ | cons
         2    to MORRISDL
         2  from MORRISDL
    CPU Utilization by Pipeline Specification  14 Feb 1998
19:47:49

     9.399 (9.399) ms in pipeline "NoName001"
                                    (1 invocation)
    45.052 ms total.

Detailed output from Rita in UNNAMED RITA002.
Ready; T=0.08/0.10 19:47:49
```

I need to measure how much communication I have with my manager. I do this by analyzing my Netlog file with the above pipeline. So in the example above, I replace "PIPE" with "RITA", followed by the regular stages. The pipeline completes with the two output lines from the normal pipeline of "2 to MORRISDL" and "2 from MORRISDL". Some summary information is also output at this time along with a pointer to the file which contains the detailed information, UNNAMED RITA002 in this case.

Rita Output

```
CPU Utilization by Pipeline Specification   from: 14 Feb 1998 19:47:49
                                              to: 14 Feb 1998 19:47:49

CPU utilization of pipeline "NoName001":
    0.635 ( 0.635) ms ( 33K) in stage  1 of segment  1: < bitner netlog a
    4.248 ( 4.248) ms (  4K) in stage  2 of segment  1: spec 31-45 2
    4.011 ( 4.011) ms ( 16K) in stage  3 of segment  1: sort count
    0.486 ( 0.486) ms (  1K) in stage  4 of segment  1: locate /MORRISDL/


    0.019 (  0.019) ms (   2K) in stage   5 of segment  1: cons
    9.399 (  9.399) ms in pipeline "NoName001" (1 invocation) <=====

    9.399 ms attributed to stages; 82 virtual I/Os.

    0.016 ms in general overhead.
    0.460 ms in scanner.
    3.792 ms in dispatcher.
   31.385 ms in accounting overhead.

   45.052 ms total.
```

The detailed output file provides a line of output for each stage which reports processor and storage usage. A summary line of the processor time for all stages and virtual I/Os comes next. Lastly, a breakdown of management overhead between various pipeline processes is given. In our example, we can see that a large portion of our overhead is in the management of the pipeline with just over 20% of the overhead in the actual stages. The bulk of the stage overhead is in SPEC and SORT.

Improved Pipe

```
rita < bitner netlog a | locate /MORRISDL/
     | spec 31-45 2 | sort count | cons
         2    to MORRISDL
         2  from MORRISDL
    CPU Utilization by Pipeline Specification
14 Feb 1998 19:55:13

   3.841 (  3.841) ms in pipeline "NoName001"
                                  (1 invocation)

  37.201 ms total.

Detailed output from Rita in UNNAMED RITA004.
Ready; T=0.07/0.09 19:55:13
```

You might have realized that a possible improvement was to move the Locate up earlier in the pipeline so as to minimize the number of records that need to be processed later on. When we run the pipeline again with Rita, we see that we have saved about 17% of the processor time. Not bad for a simple change.

Improved Pipe Rita Output

```
CPU utilization of pipeline "NoName001":
 0.609 (0.609) ms (33K) in stage 1 of segment  1: < bitner netlog
a
 3.024 (3.024) ms ( 1K) in stage 2 of segment  1: locate
/MORRISDL/
 0.135 (0.135) ms ( 4K) in stage 3 of segment  1: spec 31-45 2
 0.053 (0.053) ms (16K) in stage 4 of segment  1: sort count
 0.020 (0.020) ms ( 2K) in stage 5 of segment  1: cons
 3.841 (3.841) ms in pipeline "NoName001" (1 invocation) <=====

 3.841 ms attributed to stages; 82 virtual I/Os.

 0.019 ms in general overhead.
 0.477 ms in scanner.
 1.808 ms in dispatcher.
31.056 ms in accounting overhead.

37.201 ms total.
```

The detailed report with the new pipe shows that overall pipeline management time is about the same (37.056 compared to the previous 31.385), but processing time in the stages decreased dramatically. Now the Locate stage is the most expensive and Spec and Count are considerable smaller. You may also notice that there is some run variability. The disk read stage that starts the pipeline did not change, but the processor time is different.

TRACEXEC

- Written by Kent Fiala of SAS Institute
  - SASKLF@VM.SAS.COM
- Analyze and debug EXEC applications
- Available at http://ukcc.uky.edu:80/~tools.1998/
- Statistical processor usage at Exec level
- Profiling and debug capabilities
- Handles execs calling other execs

TRACEXEC is a powerful tool for collecting processor usage for execs. It was written by Kent Fiala of SAS and is available on the VM Workshop virtual tools tape.
It has various functions, but the one we will look at here is for collecting processor usage at the exec level, including handling the case where one exec calls another exec.

TRACEXEC Example

```
tracexec on
Ready;
gohome
TRACEXEC: Enter GOHOME  EXEC     *  gohome
TRACEXEC: Enter NQ      EXEC     *  NQ DOUG
TRACEXEC: Exit  NQ      0.007782 seconds, rc= 0
TRACEXEC: Enter NQ      EXEC     *  NQ RON
TRACEXEC: Exit  NQ      0.007742 seconds, rc= 0
TRACEXEC: Enter QSTOCK  EXEC     *  QSTOCK
TRACEXEC: Exit  QSTOCK  0.024401 seconds, rc= 0
TRACEXEC: Enter QCPU    EXEC     *  QCPU
TRACEXEC: Exit  QCPU    0.004249 seconds, rc= 0
TRACEXEC: Exit  GOHOME  0.052223 seconds, rc= 0
Ready;
```

In this example, we simply turn on Tracexec with the TRACEXEC ON command. This will result in enter and exit messages to be displayed for each exec that is run. The QUIET option can be used to suppress these messages.
The basic flow is that the GOHOME exec calls three other execs: NQ, QSTOCK, and QCPU. Note that the NQ exec is called twice.

TRACEXEC Stats Command

```
tracexec stats
QCPU   EXEC  n=1  0.004249 secs  0.004249 secs.
QSTOCK EXEC  n=1  0.024401 secs  0.024401 secs.
NQ     EXEC  n=2  0.015524 secs  0.015524 secs.
GOHOME EXEC  n=1  0.052223 secs  0.008049 secs.

Ready;
```

For scenarios where many execs are called, the QUIET option can be used when enabling Tracexec. Statistics can be displayed with the TRACEXEC STATS command. These statistics are also shown when the TRACEXEC OFF command is issued.
Three items are given for each exec: the number of times the exec was invoked, the accumulated cpu time for execution of the exec, and the accumulated cpu time spent exclusively in the given exec. Since GOHOME is the only exec shown here that calls other execs, it is the only one where the two cpu numbers differ. The NQ exec was called twice, so the average cpu time used would be 0.015524 / 2 = .007762. The majority of the cost of GOHOME (0.052223) is in the other execs. GOHOME is only 0.008049 by itself.

Other Possibilities

- CMS Commands
  - SVCTRACE
  - QUERY FILEPOOL
- Other CP Commands
  - CP TRACE TABLE
  - TRSOURCE DATA trace

Some other possible commands for insight into your applications include the CMS SVCTRACE command and various flavors of the Query Filepool command. For most of the Query Filepool information you will again need to issue the command twice and then take deltas. The CP trace table can be a wonderful source of information and for really specific things you can use TRSOURCE data traces.

Instrumenting Applications

- Monitor Application Data
  - Diagnose x'DC'
- Diagnose x'70'
- Internal trace tables or footprints
- Accounting Data

I could spend a whole session on how to instrument applications. There are various methods of accomplishing this. The hardest part tends to be determining what data to capture in the first place. I could also do a whole session on that.

Summary

- There are a number of tools in VM that were not meant for performance analysis
- That does not mean you cannot use them
- Be careful of
  - skewing results
  - repeatable environments
  - multiple changes between measurements

I hope this brief review of using nonperformance tools for performance work was helpful. There are a few things to remember when doing analysis of this type. Be careful that by monitoring or using tools to analyze a system that you do not impact the system in the process or you will need to deal with the possible skew in the measurements. Try to make the environments as repeatable as possible. The order of commands, first time execution, and virtual machine configuration can all be factors. When measuring changes you make to your application, it is best to make only independent changes. Otherwise one change could improve performance and another could degrade performance, but your measurements might show no change in performance.