Using AIX ProbeVue to look at MQ application performance

Simon Davitt |July 7 2016 Updated

One of the most commonly asked questions for users of MQ is 'How can I measure the performance of my MQ application?'.

There are many tools to assist the user including MQ  application trace, MQ internal trace, and other debugging tools. All of these tools have their pros and cons not limited to the fact that the trace itself has it's own performance overhead.

For users of MQ in the AIX and Linux environment there are a couple of tools which can be used to investigate low level information about specific operating system and application function calls. On AIX this is the **ProbeVue** utility which was introduced at AIX V6 and has been improved at subsequent releases.  ProbeVue is a dynamic tracing facility that can be used to monitor the entire AIX operating system down to specific system call level.

The purpose of this blog is to show how ProbeVue can be used to instrument a customer application without the need to modify, re-link, or re-instrument the application in any way. In fact it can be used to instrument a running application with the need to stop and restart the process.

The following is a simple ProbeVue script created by modifying several of the existing scripts in the IBM developerWorks ProbeVue resources (see References section).

**$  more app\_monitor.e**

#!/usr/bin/probevue

#

#  Syntax:  probevue app\_monitor.e <PID>

#

int global\_count;
int interval\_count;

\_\_list   s\_timeList;
\_\_list   l\_timeList;

\_\_thread probev\_timestamp\_t s\_time;

\_\_kernel long time;

\_\_thread long long start\_time;

@@BEGIN
{
   printf("ProbeVue app\_monitor.e started\n\n");
   printf("Monitoring  process PID=%d\n\n", $\_\_CPID);
   printf("\*\*Note\*\* The times displayed are given in MICROSECONDS\n\n");
   global\_count = 0;
   interval\_count = 10;
}
@@END
{
   printf("ProbeVue putMessage ended\n");
}

@@uft:$\_\_CPID:\*:MQPUT:entry
{
  global\_count++;
  s\_time = timestamp();
}

@@uft:$\_\_CPID:\*:MQPUT:exit
{
   \_\_auto e\_time;
    \_\_auto long d\_time;

   e\_time = timestamp();

   d\_time = diff\_time(s\_time,
                      e\_time,
                      MICROSECONDS);
   if (d\_time > 0) {
      time\_array[d\_time]++;
      append( s\_timeList, d\_time);
      append( l\_timeList, d\_time);
   }
}

@@interval:\*:clock:10000
{

   if (count(s\_timeList) == 0) {
      exit();
   }

  if (0==interval\_count%10) {
     printf("\n\t\t\t\t10s Count\t\tTotal\t\tMaxTime\t\tMinTime\t\tAvgTime\n");
  }
  interval\_count++;

  printf("\t\t\t\t%d\t\t\t%d\t\t%d\t\t%d\t\t%d\n",
            count(s\_timeList),
            count(l\_timeList),
            max(l\_timeList),
            min(l\_timeList),
            avg(l\_timeList));
  s\_timeList = list();
}

This example script calculates the difference between the start of an MQPUT operation and the end of an MQPUT operation and outputs summary information every 10 seconds (10000 milliseconds).

**Example 1:**

**# probevue  app\_monitor.e 15663232**

welcome to blast
ProbeVue app\_monitor.e started

Monitoring  process PID=15663232

\*\*Note\*\* The times displayed are given in MICROSECONDS

                                10s Count               Total           MaxTime         MinTime         AvgTime
                                40174                   40174           260430          81              159
                                40430                   80604           260430          81              166
                                19396                   100000          260430          81              162
ProbeVue putMessage ended
#

In Example 1 we have hooked into a running application which issued 100000 MQPUTs and the  ProbeVue script has produced statistic information every 10 seconds.

In addition to using the syntax described in the script header  probevue allows a target application to be started and arguments passes to that application.  For example,

**# probevue -X <PATH\_TO\_MY\_APPLICATION> -A "arg1 arg2 arg3 . . . argn" app\_monitor.e**

The following example uses a modified version of the MQ sample program amqsblst.c to write 1 million persistent messages under syncpoint to a local queue.

**Example 2:**

**# probevue -X /home/xxxxxxxx/my\_blast -A "MYQMGR MYQL -W -c 1000000 -u 10 -q  " app\_monitor.e**

welcome to blast
ProbeVue app\_monitor.e started

Monitoring  process PID=10748156

\*\*Note\*\* The times displayed are given in MICROSECONDS

                                10s Count               Total           MaxTime         MinTime         AvgTime
                                36304                   36304           68741           81              120
                                38623                   74927           68741           81              115
                                37563                   112490          68741           81              115
                                35734                   148224          68741           81              115
                                37843                   186067          68741           81              115
                                37385                   223452          76492           79              115
                                37919                   261371          76492           79              115
                                38339                   299710          76492           79              115
                                37279                   336989          76492           79              115
                                37641                   374630          83854           78              115

                                10s Count               Total           MaxTime         MinTime         AvgTime
                                36070                   410700          84037           78              115
                                37594                   448294          89533           78              115
                                36276                   484570          95950           78              115
                                37060                   521630          121083          78              115
                                37490                   559120          121083          78              115
                                35874                   594994          121083          78              116
                                37930                   632924          121083          78              116
                                37066                   669990          121083          78              116
                                36498                   706488          121083          78              116
                                37372                   743860          121083          78              116

                                10s Count               Total           MaxTime         MinTime         AvgTime
                                36730                   780590          121083          78              116
                                36500                   817090          121083          78              116
                                35100                   852190          121083          78              116
                                34610                   886800          121083          78              116
                                35406                   922206          145262          78              117
                                37134                   959340          145262          78              117
                                36453                   995793          145262          78              117
                                4207                    1000000         145262          78              117
ProbeVue putMessage ended
#

Well that was interesting we already have some useful information  knowing that the average time to put a message was 117 microseconds (0.117 milliseconds) and the maximum time for an individual MQPUT was 145262 microseconds (145.262 milliseconds).

What happens if you change the application to write persistent messages outside syncpoint?  This is achieved by removing the **-u 100** option from the command. Let's have a look (with a fewer messages because we don't have all day).

**Example 3:**

**# probevue -X /home/xxxxxxxx/my\_blast -A "MYQMGR MYQL -W -c 100000 -q  " app\_monitor.e**

welcome to blast
ProbeVue app\_monitor.e started

Monitoring  process PID=10747916

\*\*Note\*\* The times displayed are given in MICROSECONDS

                                10s Count               Total           MaxTime         MinTime         AvgTime
                                13182                   13182           20027           373             700
                                13034                   26216           42075           368             704
                                12936                   39152           52285           368             708
                                12861                   52013           98490           368             710
                                12866                   64879           127816          368             713
                                13178                   78057           127816          368             711
                                13151                   91208           127816          368             710
                                8792                    100000          127816          368             710
ProbeVue putMessage ended
#

Wow!! Such a difference! The throughput dropped from 3500+ messages/second down to 1200-1300 messages/second and the average MQPUT time increased by 6 times.

The following example is the same as Example 2 but it shows the implications of other application accessing the same queue simultaneously and the resulting queue contention.

**Example 4:**

**# probevue -X /home/xxxxxxxx/my\_blast -A "MYQMGR MYQL -W -c 1000000 -u 10 -q  " app\_monitor.e**

welcome to blast
ProbeVue app\_monitor.e started

Monitoring  process PID=15663216

\*\*Note\*\* The times displayed are given in MICROSECONDS

                                10s Count               Total           MaxTime         MinTime         AvgTime
                                37070                   37070           124524          80              120
                                35950                   73020           150994          80              122
                                36829                   109849          150994          80              121**\*1 Normal processing up to this point**
                                15066                   124915          150994          80              163**\*2 Suddenly the throughput drops**
                                9900                    134815          150994          80              208
                                9833                    144648          150994          80              248
                                8751                    153399          257960          80              285
                                5457                    158856          257960          80              326
                                5532                    164388          257960          80              363
                                5382                    169770          257960          80              399

                                10s Count               Total           MaxTime         MinTime         AvgTime
                                5410                    175180          257960          80              432
                                5665                    180845          257960          80              463
                                5605                    186450          257960          80              492
                                5321                    191771          257960          80              521
                                5234                    197005          257960          80              548
                                4985                    201990          257960          80              574
                                4900                    206890          257960          80              600
                                5013                    211903          257960          80              624
                                4717                    216620          257960          80              647
                                2957                    219577          826792          80              677

                                10s Count               Total           MaxTime         MinTime         AvgTime
                                2192                    221769          826792          80              709
                                2153                    223922          826792          80              740
                               **2070**                   225992          826792          80              771**\*3 Throughput drops to it's lowest level**
                                2303                    228295          826792          80              800
                                2384                    230679          826792          80              829
                                2393                    233072          826792          80              857
                                2562                    235634          826792          80              883
                                2356                    237990          826792          80              910
                                2916                    240906          826792          80             **935**\***4 Average time highest value**
                                8223                    249129          826792          80              935**\*5 Performance starts to recover**

                                10s Count               Total           MaxTime         MinTime         AvgTime
                                29256                   278385          826792          80              856
                                35827                   314212          826792          80              772
                                35631                   349843          826792          80              706
                                35853                   385696          826792          80              651
                              **36012**                  421708          826792          80             **606** **\*6 Throughput back to normal level.**
^C

ProbeVue putMessage ended

#

Example 4 shows the application running at it's baseline for the first 30 seconds. Suddenly the throughput and average MQPUT response time start to drop. This is **\*2**in the output. This was caused because further instances of the my\_blast program were started using various combinations of writing persistent and non-persistent messages outside syncpoint, and also opening the queue for output with the sampe amqsget program.

Notice that as the other applications were terminated (**\*6**) the throughput starts to recover together with the average response time.

Notice also that there is a gradual increase in the MaxTime value. More on this later in this blog.

**Example 5:**

Some fun stuff.

ProbeVue provides some interesting facilities including the ability to output simple graphs. (Refer to the ProbeVue **quantize** function.   A slight modification to the script above allows the use to display a histogram of the times taken for the MQPUT operations and an indication of the maximum response time. Note that this script limits the number of entries in the histogram to just 40. That is for purposes of convenience in this blog entry.

#!/usr/bin/probevue

int global\_count;
int interval\_count;
int max\_interval;

\_\_list   s\_timeList;
\_\_list   l\_timeList;

\_\_thread probev\_timestamp\_t s\_time;
\_\_kernel long time;

\_\_thread long long start\_time;

@@BEGIN
{
   global\_count = 0;
   interval\_count = 10;
   max\_interval = 0;
}
@@END
{
   printf("ProbeVue putMessage ended\n");
}

@@uft:$\_\_CPID:\*:MQPUT:entry
{
  global\_count++;
  s\_time = timestamp();
}

@@uft:$\_\_CPID:\*:MQPUT:exit
{
   \_\_auto e\_time;
    \_\_auto long d\_time;

   e\_time = timestamp();

   d\_time = diff\_time(s\_time,
                      e\_time,
                      MICROSECONDS);
   if (d\_time > 0) {
      time\_array[d\_time]++;
      append( s\_timeList, d\_time);
      append( l\_timeList, d\_time);
   }
}

@@interval:\*:clock:10000
{
  if (count(s\_timeList) == 0) {
     exit();
  }

# Clear the screen
  printf("\033[2J");

  printf("Monitoring  process PID=%d    Time: %A\n\n", $\_\_CPID, timestamp());
  printf("\*\*Note\*\* Displaying histogram of 40 lowest intervals\n");
  printf("\nMaximum interval = %d", max(l\_timeList) );
  if ( max(l\_timeList) > current\_max ) {
     printf("\n\*\* Warning\*\* Increased Max Time Interval. Previous Interval (%d). New Interval (%d).\n", current\_max, max(l\_timeList) );
     current\_max = max(l\_timeList);
  } else {
     printf("\n\n");
  }
  printf("\nTime Interval(uS)\t\tFrequency\n");

# Display a histogram of the data collected.
**quantize(time\_array,40 );**
  s\_timeList = list();
}

Note: To run this command successfully required increasing some the ProbeVue default properties using the **probvctrl** command.  The list of properties can been seen as follows:

**# probevctrl -l**
Probevue Features: on
MAX pinned memory for Probevue framework(in MB): 16
Default per-CPU trace buffer size(in KB): 8
Default trace buffer read rate(in ms): 100
Size of per-CPU computation stack(in KB): 12
MAX concurrent sessions allowed for regular user: 1
MAX pinned memory for regular user sessions(in MB): 2
MIN trace buffer read rate for regular user(in ms): 100
Size of per-CPU local table size(in KB): 4
Number of page faults to be handled: 0
Number of threads to be traced: 32
Minimum interval allowed in an interval probe: 1
Percent of dynamic data structure's memory allocated: 50
Max network buffer size(in bytes): 96
Max systrace probe execution time in Interrupt environment(in ms): 0
Max sysproc probe execution time in Interrupt environment(in ms): 0
Max io probe execution time in Interrupt environment(in ms): 0
Max network probe execution time in Interrupt environment(in ms): 0
#

In this case the property "Default per-CPU trace buffer size" is too small.  It is increased as follows:

**# probevctrl -c default\_buffer\_size=32**
Attention: The command "/usr/sbin/bosboot -a" must be run for the change to take effect in the next boot.

By default the change is temporary unless the **bosboot** command is also executed.

OK The environment is set, let's see what it produces.

Using this script with the same command as in the previous examples the following output is seen:

**# probevue -u -X /home/xxxxxxxx/my\_blast -A "MYQMGR MYQL -W -c 1000000 -u 100 -q  " app\_monitor\_2.e**

Monitoring  process PID=12517568    Time: Jul/06/16 15:51:22

\*\*Note\*\* Displaying histogram of 40 lowest intervals

Maximum interval = 311470
\*\* Warning\*\* Increased Max Time Interval. Previous Interval (269086). New Interval (311470). **\*1**

Time Interval(uS)               Frequency
key                             value
75                              2
76                              1
77                              2
78                              4
79                              2
80                              1
81                              20
82                              205
83                              983                            ===
84                              3002                           =========
85                              4858                           ===============
86                              6574                           ====================
87                              8065                           ========================
88                              9629                           =============================
89                              10237                          ===============================
90                              10150                          ===============================
91                              9959                           ==============================
92                              9638                           =============================
93                              9340                           ============================
94                              9415                           =============================
95                              10135                          ===============================
96                              10339                          ================================
97                              9996                           ==============================
98                              8790                           ===========================
99                              7289                           ======================
100                             5422                           ================
101                             3973                           ============
102                             2856                           ========
103                             2212                           ======
104                             1876                           =====
105                             1608                           ====
106                             1422                           ====
107                             1313                           ====
108                             1134                           ===
109                             1099                           ===
110                             1019                           ===
111                             944                            ==
112                             1036                           ===
113                             1060                           ===
114                             1224                           ===

Well it is pretty but was there anything significant? Yes there is. The line highlighted here in red shows and increase in the Maximum Interval shows a decrease in the MQPUT response time. The implication of this is that a possible build up of messages can occur on the target queue. If the script continued to show increases then this warrants further investigation. There are a number of possible causes including slow response from the underlying operating system write() call, contention on the queue resource as seen in previous examples. MQ is heavily dependent on the IO latency provided by the file systems that it uses. Although average IO times (IOPS)  can look very healthy sudden and temporary blips in IO latency can have a severe affect on performance.

For more information about IOPS and latency refer to,  <http://recoverymonkey.org/2012/07/26/an-explanation-of-iops-and-latency/>

**Summary**

From one very simple ProbeVue script we can find a great deal of information. It can be an extremely powerful and useful tool to give insights into both the AIX operating system and our applications.

It can be used to highlight

* performance impact of using MQ persistent messages outside of syncpoint
* performance implications of contention for a single MQ queue resource
* potential impact of underlying operating system IO performance on MQ

The first 2 of this list can be investigated by the customer looking at the design and implementation of their applications.  The IBM developerWorks wiki has some excellent sample scripts which can be used for investigating potential issues with performance of the IO subsystem.

**References**

Introduction to ProbeVue on YouTube refer to <https://www.youtube.com/watch?v=_UMurdX0d5k>

For more information about ProbeVue refer to <http://www.ibm.com/support/knowledgecenter/ssw_aix_72/com.ibm.aix.genprogc/probevue_userguide.htm>

For IBM developerWorks resources refer to <https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/ProbeVue/page/Resources>