Embedded Application Performance Instrumentation
(aka Rational Application Performance Investigation Details...)
Tom Bascom, White Star Software

Abstract: You can only make an application go faster by improving the parts that are slow. It doesn't help to optimize code that is already fast enough or that is only rarely used.

User table stats, temp table usage stats and the application profiler can all be easily embedded in your application to provide powerful insight into exactly where performance problems are originating.

In this session we will explore these 3 under-utilized OpenEdge features, explain why they are useful and how you can embed them into your own code base to focus your performance improvement activities on code that really is slow rather than wasting time trying to guess which bits aren't as fast as they could be! You will take home simple but powerful code samples that can be easily used with any OpenEdge environment: .NET GUI, Webspeed, App server, PASOE, Character, Batch or anything else!
Embedded Application Performance Instrumentation

aka: Rational Application Performance Investigation Details 😊

Tom Bascom, White Star Software
tom@wss.com
A Few Words about the Speaker

• Tom Bascom: Progress user & roaming DBA since 1987
• Partner: White Star Software & DBAppraise, LLC
  – Expert consulting services related to all aspects of Progress and OpenEdge.
  – Remote database management service for OpenEdge.
  – Author of: protop
  – Simplifying the job of managing and monitoring the world’s best business applications.
  – tom@wss.com
Agenda

• Why?
• The Profiler
• User Table and Index Statistics
• Temp Table Statistics
• What is This “Embedded” Nonsense?
Why?
Why **NOT** just use ETIME() and MESSAGE???

- Non-Repeatable
- Subject to a host of external factors
  - CPU speed, disk throughput, virtualization, contention from other user activity, buffer cache efficiency, phase of the moon
- Granularity is too gross (only millisecond timings)
- You have to **guess** the right target code to measure
- You have to write a lot of custom logging code
- Coding in your production environment is a **BAD** idea
The performance enhancement possible with a given improvement is limited by the fraction of the execution time that the improved feature is used.

-- Amdahl’s Law

\[ S = \frac{T_s}{T} = \frac{1}{f_p/n + f_s} \]
In other words:

• Trying to improve small things that nobody notices probably isn’t the road to fame and fortune.

• Big queries that return lots of data and which are frequently used by lots of users will be much more noticeable.
The Profiler
The Profiler

• Microsecond timings of actual code execution time!
  – Does not include waiting for IO (disk IO, sockets, UPDATE, PAUSE etc statements…)
  – Covers ALL of the code!

• First introduced with version 8.2 (-zprofile)

• Greatly Improved with version 9.0 (profiler: handle)

• “Unsupported” is a myth (“lightly documented” is not a myth)
  • The profiling capability *is* supported, the example GUI tool to evaluate the results was not. That GUI tool is also completely unnecessary eye candy.
Unsupported Profiling Eye Candy
New PDSOE Eye Candy
ChUI Eye Candy

- Dynamically capture line by line execution time at the point where issues occur.
- Send output to a user-defined destination.
- `<Esc>P` to configure and enable. (Don’t forget “Yes”)
- `<Esc>P` again to complete the capture and disable.
Using the Profiler

• -profile startup parameter
  – Non-intrusive
  – Non-selective (all or nothing)
  – Must end session cleanly – no aborts or kills!

• profiler: handle
  – Selective
  – Can turn it on or off at will
  – Can flush the data when you want to
  – But requires code insertion or tooling of some sort
Minimal Embedded Usage

assign
  profiler:enabled = yes
  profiler:profiling = yes
.
doi = 1 to 1000000:
  /* some code to be profiled! */
end.

assign
  profiler:enabled = no
  profiler:profiling = no
.
profiler:write-data().
  /* create the output file */
PROFILER Attributes

- DESCRIPTION – optional text describing this session
- LISTINGS – whether or not to create debug listings
- DIRECTORY – where to create debug listings (default to –T)
- FILE-NAME – name of output file (default profile.out)
- ENABLED – yes/no; initializes listings and so forth
- PROFILING – turn profiling on or off
Other PROFILER Attributes

- TRACE-FILTER – CSV list of “matches” criteria for procedure tracing
- TRACING – line level tracing
- COVERAGE - % coverage support
PROFILER Methods

- Write-Data() – flush accumulated data to output file.
- User-Data(char) – write user defined data, such as VST statistics, to the output file.
Profiling an Entire Session

- Create file called profiler.cfg with 3 lines:
  -OUTFILE /tmp/profiler.out
  -LISTINGS /tmp
  -DESCRIBE someDescription

- Add –profile to session startup:
  mpro dbName -p start.p -profile profiler.cfg

- Run normally.

- Terminate cleanly & analyze the output.

- Multiple gigabytes of data may be generated very quickly!!!
Profiler Tooling

zprof_on.p - initializes profiling
zprof_off.p - ends profiling and writes the output file
zprof_topx.p - produces a report on the results
zprof_flush.p - flushes profiling output, this is optional but necessary if your program does not terminate cleanly
zprof_check.p - checks for a basename.zprof "flag" file and starts profiling if it exists, stops profiling if it does not

to use:
run lib/zprof_on.p ( "baseName", "description", no ).
run interestingStuff.p
run lib/zprof_off.p.

This will create a single basename*.prf file that can be loaded into a profiler analysis tool such as zprof_topx.p

define variable zprofilerState as logical initial ?.

procedure zprofiler:

if zprofilerState = yes then /* the profiler is already running */
do:
    run zprofiler_off. /* flip the state of the profiler to "off" */
    zprofilerState = no.
    run zprofiler_load. /* load profiler data into temp-tables to analyze */
    run zprofiler_proc. /* process the data */
    run zprofiler_topx( no ). /* report on the top 20 execution time lines -- to file */
    run zprofiler_topx( yes ). /* report on the top 20 execution time lines -- to TTY */
    return. /* do not continue after this -- return */
end.

- Continued on next slide -
if zprofilerState = ? then
    do:
        run lib/zprofiler.p persistent ("protop3",
            "ProTop3 Execution Profile").
        zprofilerState = yes.
    end.
if zprofilerState = no then
    zprofilerState = yes.
if zprofilerState = yes then
    do:
        run zprofiler_on.
        zprofilerState = yes.
    end.
### Sample Profiling Output

**Profiler: Top 20 Results**

**Description:** ProTop3 Execution Profile [00:00:23]

**Session Total Execution Time:** 00:00:06

| Line 0 = initialization, line -1 = cleanup |

<table>
<thead>
<tr>
<th>Program/Class</th>
<th>Line</th>
<th>Time</th>
<th>Avg Time</th>
<th>Calls</th>
<th>Internal Procedure/Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>dc/dashboard.p</td>
<td>3105</td>
<td>3.702797</td>
<td>0.000001</td>
<td>400004</td>
<td>mon-update</td>
</tr>
<tr>
<td>dc/dashboard.p</td>
<td>0</td>
<td>2.097173</td>
<td>0.524293</td>
<td>4</td>
<td>mon-update</td>
</tr>
<tr>
<td>dc/dashboard.p</td>
<td>3106</td>
<td>0.497054</td>
<td>0.000000</td>
<td>4000000</td>
<td>mon-update</td>
</tr>
<tr>
<td>dc/dashboard.p</td>
<td>3087</td>
<td>0.393956</td>
<td>0.004061</td>
<td>97</td>
<td>mon-update</td>
</tr>
<tr>
<td>lib/dynscreen.p</td>
<td>1238</td>
<td>0.025957</td>
<td>0.006489</td>
<td>4</td>
<td>dynScreenUpdate</td>
</tr>
<tr>
<td>ssg/sausage00.p</td>
<td>1949</td>
<td>0.024891</td>
<td>0.000007</td>
<td>3472</td>
<td>dataSet2JSON</td>
</tr>
<tr>
<td>lib/dynscreen.p</td>
<td>936</td>
<td>0.021612</td>
<td>0.000027</td>
<td>800</td>
<td>dynViewerUpdate</td>
</tr>
<tr>
<td>ssg/sausage00.p</td>
<td>1752</td>
<td>0.020572</td>
<td>0.000007</td>
<td>3084</td>
<td>scanDataSet</td>
</tr>
<tr>
<td>ssg/sausage00.p</td>
<td>1920</td>
<td>0.019844</td>
<td>0.000005</td>
<td>4252</td>
<td>dataSet2JSON</td>
</tr>
</tbody>
</table>
if pt_bogomips > 0 then
    do:
        estart = etime.
        do i = 1 to pt_bogomips: /* 100,000 should take 20ms to 30ms on c.2010 HW */
            end.
        tt_Dashboard.bogoMIPS = ( 1000 / ( etime - estart )) * ( pt_bogomips / 1000000 ).
        if tt_Dashboard.bogoMIPS > 0 and tt_Dashboard.bogoMIPS <> ? then
            do:
                maxBogo = max( maxBogo, tt_Dashboard.bogoMIPS ).
                tt_Dashboard.bogoMipPct = ( tt_Dashboard.bogoMIPS / maxBogo ) * 100.
            end.
    end.
end.
About Line Numbers

• Many tools use DEBUG LIST line numbers.
• This listing expands all include files and pre-processor statements.
• COMPIL "program.p" DEBUG-LIST "program.dbg".
• Line numbers can vary if things have changed since code was deployed!
  – It is very helpful to create and deploy .dbg files whenever you create and deploy r-code
Caveat!

define variable i as integer no-undo.

assign
    profiler:enabled = yes
    profiler:profiling = yes
.

do i = 1 to 1000000:
end.

i = 0.
do while i < 1000000:
    i = i + 1.
end.

i = 0. do while i < 1000000: i = i + 1. end. /* line 16 */

profiler:enabled = no.
profiler:profiling = no.

profiler:write-data().
## Caveat!

1 02/28/2007 "Generic" 07:55:03 "tom"

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>&quot;profile.p&quot; &quot;&quot;</td>
<td>63126</td>
<td>0 0 2 1</td>
<td>0 0 1 0.000000 30.935828</td>
<td>2 11 1 0.000000 0.000002</td>
<td>2 8 1000001 4.607678 4.607678</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>9 1000000 1.719586 1.719586</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>14 1000000 1.501487 1.501487</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>12 1000001 3.013981 3.013981</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>13 1000000 3.032433 3.032433</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>16 1 0.000003 0.000003</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note 1:** There are 2 formats – the 1st character specifies the version

**Note 2:** The input file is in “dot-d format”, a “.” on a line causes a 4GL LEAVE.
User Table and Index Statistics
User Table and Index Statistics

- Aggregate Table and Index stats were introduced in 8.3
- That was such a great feature that user level stats were introduced in OE 10!
- Now you can see how much of your database activity is from a given user.
- This is run-time behavior – not static, compile time analysis of index selection; IOW, what really happens vs what “should” happen.
Enabling User Table and Index Statistics

- basetable default = 1
- baseindex default = 1

- tablerangesize default = 50
- indexrangesize default = 50

```
select count(*) from _file
select count(*) from _index
```

Metaschema and VST tables have negative _file-num, SYS* start after 32000 – User tables are generally from 1 to 10000. Index _idx-num has no pattern.
run lib/usertablestats.p persistent.

for each dictdb.order no-lock:
end.

{lib/userstats.i}
run getUStats ( output table tt_usrTblInfo by-reference,
    output table tt_usrIdxInfo by-reference ).

for each tt_usrTblInfo by tt_usrTblInfo.tblRd descending:
    display tblName tblRd tblCr tblUp tblDl with 5 down.
end.

for each tt_usrIdxInfo by tt_usrIdxInfo.idxRd descending:
    display idxName idxRd idxCr idxDl with 5 down.
end.
# Top 5 User Tables & Indexes

## Top 5 Tables Used by My Session

<table>
<thead>
<tr>
<th>tblName</th>
<th>tblRd</th>
<th>tblCr</th>
<th>tblUp</th>
<th>tblDl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order-Line</td>
<td>2,619</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Customer</td>
<td>332</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Order</td>
<td>207</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Item</td>
<td>165</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Salesrep</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

## Top 5 Indexes Used by My Session

<table>
<thead>
<tr>
<th>idxName</th>
<th>idxRd</th>
<th>idxCr</th>
<th>idxDl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order-Line.order-line</td>
<td>2,622</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>_Field._Field-Name</td>
<td>2,191</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>_Index._File/Index</td>
<td>873</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>_File._File-Name</td>
<td>577</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Customer.Cust-Num</td>
<td>336</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Caution

- BLOB and CLOB field activity is misreported!
- It will be recorded as activity on tables that have the same “table Id” as the “LOB Id” (will be fixed in OE12).
- Memory use:
  \[-n + (-Mn + 1) \times \text{tablerangesize} \times 32\]
  \[-n + (-Mn + 1) \times \text{indexrangesize} \times 40\]
Temp Table Statistics
Temp Table Statistics

• The trend is very clear – applications are increasing TT usage!
• Temp-tables and ProDataSets are vital components of modern applications
• Programmers have very little insight into how the temp tables in their code are behaving
• Temp Table Statistics were introduced in OE11
Progress.Database.TempTableInfo

- ArchiveIndexStatistics
- ArchiveTableStatistics
- TempTableCount
- TempTablePeak

- GetTableInfoByPosition()
- GetTableInfoByID()
- GetTableStatHistoryHandle()

- GetIndexInfoByID()
- GetIndexStatHistoryHandle()

- GetVSTHandle()

• -ttbaseindex 1
• -ttbasetable 1
• -ttindexrangesize 1000
• -tttablerangesize 1000

“Id” is what you need to link things together!
<table>
<thead>
<tr>
<th>Temp-Table Info</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>/home/pt3dev/tmp/DBI-9950762889q2d8B</td>
<td></td>
</tr>
<tr>
<td>1048576 DBI File Size</td>
<td>84 current temp-tables</td>
</tr>
<tr>
<td>1KB TT DB Block Size</td>
<td>5 archived</td>
</tr>
<tr>
<td>1288 TT DB Total Blocks</td>
<td>125 peak</td>
</tr>
<tr>
<td>193 TT DB Empty Blocks</td>
<td>275 tt indexes</td>
</tr>
<tr>
<td>2 TT DB Free Blocks</td>
<td>1669 total current records</td>
</tr>
<tr>
<td>0 TT DB RM Free Blocks</td>
<td>109831 total current bytes</td>
</tr>
</tbody>
</table>

99.53% tt hit ratio

3225 tt rec create
34660 tt rec read
3032 tt rec update
85 tt rec delete
96186 tt rec log rd
453 tt rec os rd
1046 tt rec os wr
5376 tt TRX
64 tt Undos
## Detailed Temp-Table Info

<table>
<thead>
<tr>
<th>TT Name</th>
<th>Procedure Name</th>
<th>Bytes</th>
<th>Records</th>
<th>Create</th>
<th>Read</th>
<th>Update</th>
<th>Del</th>
<th>OSRd</th>
</tr>
</thead>
<tbody>
<tr>
<td>tt_tbl</td>
<td>protop.p</td>
<td>5863</td>
<td>184</td>
<td>184</td>
<td>17145</td>
<td>9</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>tt_tbl.xid-idx</td>
<td>protop.p</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tt_idx</td>
<td>protop.p</td>
<td>10650</td>
<td>201</td>
<td>201</td>
<td>416</td>
<td>32</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>tt_idx.xid-idx</td>
<td>protop.p</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tt_screenElement</td>
<td>lib/dynscreen.p</td>
<td>34254</td>
<td>408</td>
<td>408</td>
<td>825</td>
<td>165</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>tt_screenElement.scrFrame</td>
<td>lib/dynscreen.p</td>
<td>418</td>
<td>1409</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tt_screenElement.elNm_frNum</td>
<td>lib/dynscreen.p</td>
<td>418</td>
<td>407</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tt_browseColumnList</td>
<td>lib/dynscreen.p</td>
<td>2701</td>
<td>65</td>
<td>65</td>
<td>989</td>
<td>37</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>tt_browseColumnList.brwCol</td>
<td>lib/dynscreen.p</td>
<td></td>
<td>65</td>
<td></td>
<td>468</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tt_browseColumnList.brwHdl</td>
<td>lib/dynscreen.p</td>
<td></td>
<td>102</td>
<td></td>
<td>734</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Embedded Nonsense
Embedding Instrumentation

• Do not disturb the user!
• Make it easy to enable or disable remotely and IN PRODUCTION
• Gather detailed data only when you need it.
• Simple and immediately useful output.
output stream outStrm to value( flgFileName ).

do while true:

    file-info:file-name = flgFileName. /* check (frequently)... */
    if file-info:full-pathname = ? then leave. /* if the flag disappears then gracefully exit */

    file-info:file-name = dbgFileName.
    if file-info:full-pathname <> ? And
        ( file-info:file-mod-date <> dbgChgDate or file-info:file-mod-time <> dbgChgTime )
        then do:
            dbgChgDate = file-info:file-mod-date. /* remember when we last looked! */
            input stream inStrm from value( file-info:full-pathname ).
            import stream inStrm dbgMode. /* read the new value of “dbgMode” */
            input stream inStrm close.
            if dbgMode = “profiler” then run lib/zprof_on.p.
        end.

        /* do the work... */

end.

White Star Software
Embedding in a “Headless” Process

This approach allows you to enable and disable profiling externally -- just create `baseName.zprof` to enable profiling and remove it to disable profiling. With the `zFlush` flag set to "yes".

Each iteration of the loop will create a `basename.YY.MM.DD.HH.MM.SS.prf` output file.

```
  do while true:
    run lib/zprof_check( "baseName", "description", yes ).
    run interestingStuff.p
    pause 10 no-message.
  end.
```
Embedding in a “Headless” Process

/* zprof_check.p -- simple stand-alone non-persistent profiler control
 */
* run lib/zprof_check.p ( "baseName", "description", zFlush ).
* check for baseName.zprof in session:temp-directory -- if it exists
* start profiling, if it does not then end profiling
* if baseName is blank or unknown then check for "zprof.zprof"
*/

flagfile = ( if zBaseName = "" or zBaseName = ? then "zprof" else zBaseName ).

file-info:file-name = session:temp-directory + "/" + flagFile + ".zprof".
if file-info:full-pathname = ? then
    run lib/zprof_off.p.
else
    run lib/zprof_on.p ( zBaseName, zDescription, zFlush ).
Resources!

• All provided as part of protop http://protop.wss.com
  Profiler: lib/zprof*.p use the “y” command (lower case)
  TT Stats: lib/ttinfo.p use the “0” command (zero)
  User Stats: lib/usertablestats.p use the “Y” command (upper case)

• Cool Kids:
  – https://github.com/consultingwerk/TableStatistics
Questions?
Thank You!
Profiler Example
A Calculation Bottleneck?

\[ \pi = 4 \times (1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} \ldots) \]
Profiler Example
A Calculation Bottleneck?

function piterm returns decimal ( input n as integer ).
    return ( 1.0 / (( n * 2 ) + 1 )).
end.

do while abs( newpi - oldpi ) > precision:
    oldpi = newpi.
    i = i + 1.
    if i modulo 2 = 0 then
        pi = pi + piterm( i ).
    else
        pi = pi - piterm( i ).
    newpi = ( 4.0 * pi ).
display newpi oldpi.
end.