ECMP

Enterprise . Cost . Management . Planning

Alan Mackenzie - alan.mackenzie@ecmp.co.uk
Adam Baldwin - adam.baldwin@ecmp.co.uk

Introduction – Who we are

ECMP work as an extension of our customer's organisation, working closely with the customer technicians. We provide services and solutions to bring together the key components covering Cost Management and Optimisation of an IT Estate.

We do this by bringing in experienced optimisation experts who focus on different components of the environment as needed. In addition, we have partner companies we call on for specific skills and SW needs.

- **Performance Management**
  - **Runtime environments**
  - **System settings**
  - **Application optimisation**
- **Transparency**
- **Commercial Management**

Introduction – Partnering

As you'd expect, EPS is one of these partners with their Pivotor solution. We've run a number of joint engagements to save consumption and improve overall performance for our customers.

The joint engagements with EPS have helped our customers in the US and Europe. These have ranged from optimisation engagements aimed at reducing peak / total consumption, through to focusing primarily on WLM.

When customers move to a TFP agreement we believe the offloading of their SMF post-processing provides multiple benefits. Firstly it provides space for growth on their yearly MSU consumption, an immediate cost avoidance, and secondly it allows the technical experts top focus more on performance analysis rather than producing SMF reporting.

Other partner companies focus more on the commercial side of engagements, assisting customers reduce / improve their agreements with their HW and SW vendors. This has ranged from assisting companies negotiate a move to TFP through to providing benchmarks when customers are negotiating with SPs.

What we're covering

Some key areas to cover off today:

- Transparency – ensure knowledge is available and shared – What are your important top consumers and how are they trending
- Tools – SW toolsets are very important to gain a strong look at what an address space is doing. We'll cover off some of these with some examples
- Focus Areas – Application optimisation can be a whole raft of improvements. We're going to look at some of the more common ones we come across
    - Inefficient code and SQL statements
    - Poor design
    - Run-Time overheads

Toolsets

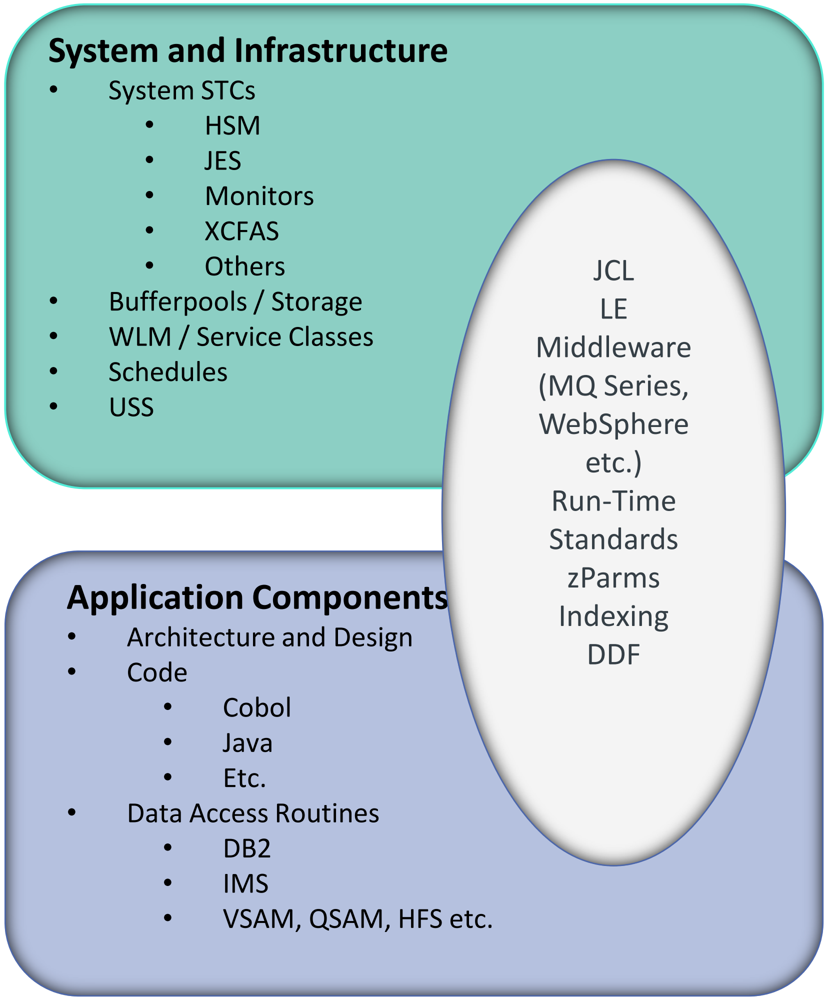
There are three fundamental tool-sets that need to be in place.

- Transparency
  - SMF post-processing. Pivotor from EPS of course, and there's others out there.
  - Too often it's not used by Application Teams, skillsets and SMF knowledge is often lacking. This adds extra pressure on the zOS teams.
- Sampling Tools – Since Strobe came about many years ago, sampling tools have been a bedrock of tuning applications and run-time environments. We've used a number over the years – IBM's APA and Compuware's Strobe products are the market leaders.
- Monitors – there are a raft of different monitors out there. Used correctly, the DB2 tools such as CQM, Detector and Apptune are very powerful tools. We'll touch more on this in a later slide

Application Optimization – Core Areas

Some core areas that we often come across

- Poor Code
  - Incorrectly defined variables resulting in a lot of conversion work to be done by the application program (under the covers from what the coder sees generally)
    - CVB, CVD etc.
  - Repetitive moving of large data
- Design issues – We recently helped a site that had huge amounts of very optimized DB2 accesses, but the design meant a lot of these were repetitive that could be avoided by different methods.
- Run-Time – Poor compile versions / options, Language Environment, overheads from external sources.



System and Infrastructure

- System STCs
  - HSM
  - JES
  - Monitors
  - XCFAS
  - Others
- Bufferpools / Storage
- WLM / Service Classes
- Schedules
- USS

JCL
LE
Middleware
(MQ Series,
WebSphere
etc.)
Run-Time
Standards
zParms
Indexing
DDF

Application Components

- Architecture and Design
- Code
  - Cobol
  - Java
  - Etc.
- Data Access Routines
  - DB2
  - IMS
  - VSAM, QSAM, HFS etc.

Code issues

It’s difficult to think of all areas where coders can cause poor performance. It’s almost too easy for them. Some more common mistakes we see are listed below:

- Poor declares of data items. For example, declaring a Numeric field as a Display Decimal. This will functionally work, but under the covers the compiler provides a host of instructions to allow it to.
- Unneeded moving and initialisation of data. You probably won’t spot these unless it’s a well used MVCL statement. Example – customer initialised a large IO-AREA before calling a subroutine that did the exact same initialisation. Removing the first one removed the overhead. Simple change but in this case, very effective.
- Invalidating Cache instructions. More likely to be in old code / compiles. Not that frequent these days, but we did come across a decent sized issue in the past year or so in a frequently used assembler program. Modifying data held in an instruction cache causes significant processor overhead. Keeping data and instructions separate avoids this overhead.

The days of experienced coders who understand the underlying fundamentals is disappearing. Those that are around still are getting more and more stretched and their beards are getting greyer.

Design issues

Understand your key Applications. Design flaws come to light when you can really understand what the program function *should* be, and how they have implemented it. It's all very well and fun to correct inefficient code usage, but generally more substantial improvements come from understanding the code in more detail.

One area we saw very high problems with at a large customer was continued retrieval of the same data from DB2. This was resolved multiple ways depending on the SQL in question.

- A caching solution. Fairly typical fix knowing that most the data will be reviewed multiple times. We created a look up cache in the program memory which provided huge savings on SQL counts, and of course the resultant saves in CPU consumption. One program we corrected resulted in a 30% reduction in ALL SQL executions on the production systems.
- The second solution was moving filtering from the application to the SQL predicates. This was a simpler change and still returned substantial saves.

Design issues

- Another case we found was spikey CPU consumption each hour. Reviewing the application showed errored transactions being re-fed into a long running batch job each hour to see if they could be processed (these were trade settlements). When trade errored, it caused the application to abend – the application had been ported from an ICL previously – and then the batch job auto-restarted and bypassed the errored trade once more. This was corrected by looking closely at the restart architecture, and implementing a 'semi-restart'. In simple terms, we prevented the 'abend', wrote the trade identifier to a single line bypass file, issued a rollback to DB2 / MQ, and when the trade was reread in we automatically put it to the error queue and bypassed it.

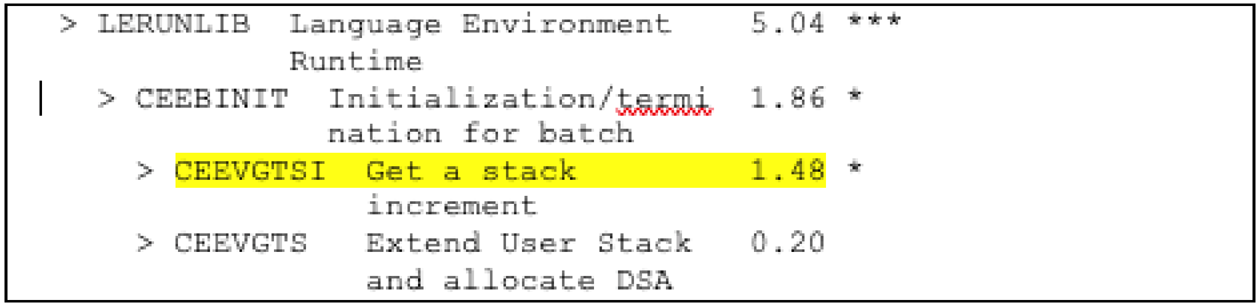
The key point here is that design flaws are often inherent in old code, and most sites do not want a huge rewrite. Whilst a couple of these examples were complex and needed a lot of testing and analysis, we did that knowing the rewards were very high.

Application Optimization – Run-time

We tend to define run-time as anything that is influencing the application address space. This can include a raft of things such a Language Environment, Traces, Monitors / Data gatherers etc.

Again, the use of a Sampling tool is the simplest way to determine what you're looking at. Here's an example from a batch job at a customer. The APA Report here is a PDF file so I've extracted some key report entries that you should look for.

This first extract is from the C01 report, drilling in the System section. For this example we see 1.68% CPU due to extending the stack size. At this particular customer we did see a LOT of stack overheads.

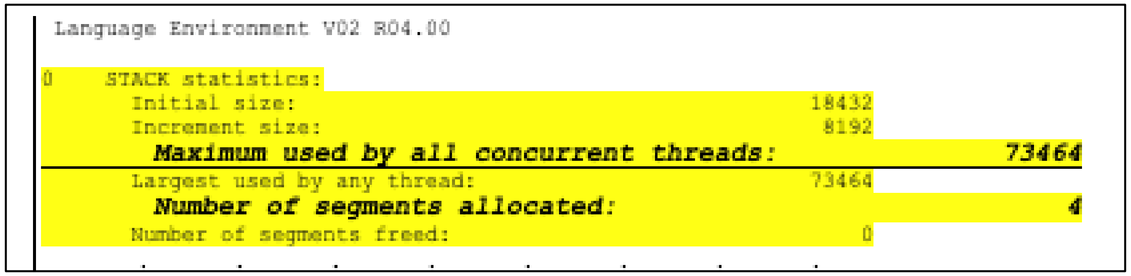


```
 > LERUNLIB  Language Environment     5.04 ***
            Runtime
   > CEEBINIT  Initialization/termi  1.86 *
               nation for batch
     > CEEVGTSI  Get a stack          1.48 *
                 increment
     > CEEVGTS   Extend User Stack    0.20
                 and allocate DSA
```

Application Optimization – Run-time

We ran some LE reports in the development environment for the same batch jobs. Fortunately they ran some large regression tests which allowed us to measure the overhead as well as the improvement from our changes.

The report below is from a RPTSTG/RPTOPT LE report we added in to the JCL via the CEEOPTS DD statement. Here we see that there were 4 segments allocated. Of course we want these to be as minimal as possible.



```
 Language Environment V02 R04.00

0     STACK statistics:
        Initial size:                                          18432
        Increment size:                                         8192
          Maximum used by all concurrent threads:                      73464
        Largest used by any thread:                            73464
          Number of segments allocated:                                    4
        Number of segments freed:                                  0
```

We're not gong through all the analysis here, but the solution was put forward a change to the allocations in the CEEPRMxx member to increase the initial allocation and remove this overhead in the majority of jobs. Whilst not huge in the overall consumption of some jobs, overall the saving was significant - a lot of small bits combined equals a decent saving.

Application Optimization – Run-time

Compiler versions and options are always interesting. The below assumes your compiler is up to date or close to, and running 6.* versions and above.

One of the key issues we continually come across is a lack of understanding or use of the ARCH option. ARCH specifies the lowest level of HW for an application to run on. By Default ARCH(8) is specified, which refers to the lowest HW being a z10. The following setting comparisons are taken from IBM’s blurb, but take these numbers with a grain of salt, even IBM’s numbers conflict across different articles.

Assuming you get within the ballpark of the improvements listed, it is a worthwhile setting to get right in your applications.

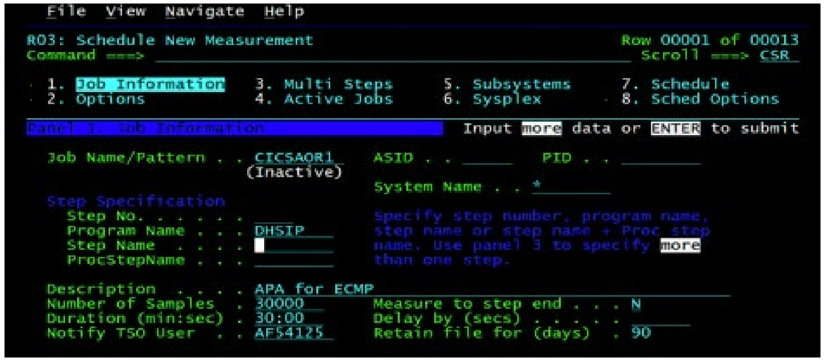
We’ve also seen great results in ensuring that when doing this you also move to OPT(2) where possible. This is not my area of speciality, but the impact of the Vector versions of old instructions seems to have provided quite the gain in some programs.

Sampling Tools

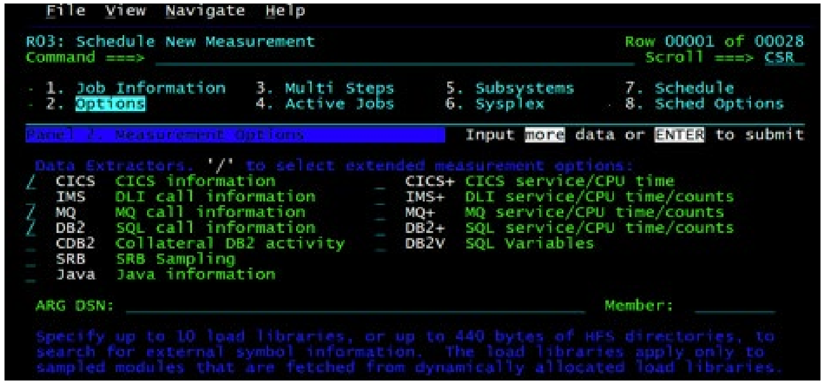
Sampling tools are a great. The best tool for application tuning.

By default we'd recommend a sample rate of about 1,000 per minute, and avoid the + collectors. For DB2, just the plain DB2 collector should be used, unless you have specific need for the extended information.

The only thing to be careful of is running it with the wrong collectors. Some of the Plus Collectors under APA will add a lot of overhead to the job. APA is kind enough to let us see it too in the reports. The second screen print is ideal.



```
  File  View  Navigate  Help

RO3: Schedule New Measurement                                    Row 00001 of 00013
Command ===> ________________________________________________      Scroll ===> CSR

  1. Job Information     3. Multi Steps       5. Subsystems       7. Schedule
  2. Options             4. Active Jobs       6. Sysplex          8. Sched Options

Panel 1. Job Information                      Input more data or ENTER to submit

  Job Name/Pattern . . CICSAOR1   ASID . . ______    PID . . ________
                       (Inactive)
                                  System Name . . *_______
  Step Specification
    Step No. . . . . . ________   Specify step number, program name,
    Program Name . . . DHSIP___   step name or step name + Proc step
    Step Name  . . . . ________   name. Use panel 3 to specify more
    ProcStepName . . . ________   than one step.

  Description  . . . . APA for ECMP
  Number of Samples  . 30000      Measure to step end . . . N
  Duration (min:sec) . 30:00      Delay by (secs) . . . . . ____
  Notify TSO User  . . AF54125    Retain file for (days)  . 90
```



```
  File  View  Navigate  Help

RO3: Schedule New Measurement                                    Row 00001 of 00028
Command ===> ________________________________________________      Scroll ===> CSR

  1. Job Information     3. Multi Steps       5. Subsystems       7. Schedule
  2. Options             4. Active Jobs       6. Sysplex          8. Sched Options

Panel 2. Measurement Options                  Input more data or ENTER to submit

 Data Extractors. '/' to select extended measurement options:
/  CICS   CICS information              _  CICS+ CICS service/CPU time
_  IMS    DLI call information          _  IMS+  DLI service/CPU time/counts
/  MQ     MQ call information           _  MQ+   MQ service/CPU time/counts
/  DB2    SQL call information          _  DB2+  SQL service/CPU time/counts
_  CDB2   Collateral DB2 activity       _  DB2V  SQL Variables
_  SRB    SRB Sampling
_  Java   Java information

 ARG DSN: ____________________________________________   Member: ________

 Specify up to 10 load libraries, or up to 440 bytes of HFS directories, to
 search for external symbol information.  The load libraries apply only to
 sampled modules that are fetched from dynamically allocated load libraries.
```

CSA driven consumption

As mentioned earlier, Sampling tools are unable to find an entry point for CSA driven consumption. Here's a tip that we use to identify the driver(s). This works for both Strobe and APA. We'll use APA as our Sampling tool and track through the steps.

[redacted]

NOSYMB in this sample is showing as 9.99% of the CPU consumption within the C01 report. We can drill down further and note the majority of the consumption is address 2900Cxxx. Interestingly there is more overhead in a similar location which will likely come from the same driver.

[redacted]

CSA driven consumption cont.

The C09 report allows us to drill into the instructions at this location. Here we see a MVCL at 2900CD9C.

[redacted]

We now jump to a lovely little free tool from IBM – TASID. This allows us to view storage easily in real time. Remember, if you're looking at an older Sample report and there's been an IPL, the offsets will have changed.

In this case, we know we had no IPL and the report had just been taken. The module loaded at that address is from SYSVIEW – so there's a nice pointer as to what is driving 10% of your consumption of that address space ☺

Questions

Any questions? If we don’t have the answer we will find it and send on to anyone interested

TASID is available as a free download from IBM on an “AS IS” basis. Instructions and download package can be found at:

https://www.ibm.com/support/pages/tasid-v521-tool