



**Guide Share Europe Nordic Region Conference
May 2008
Featuring CICS, DB2, IMS, WebSphere,
Infrastructure, and Application Development**



**Profiling Mainframe Applications
Jørgen Møller Larsen
Nykredit Bank (DK)**

A user's experience on profiling mainframe applications with the tool PathPoint from ASG (Allen Software Group). Learn how PathPoint can beat bad or non-existing documentation, and discover what's inside your legacy mainframe applications. PathPoint will help you to understand how your applications work, locate performance hogs and disclose bad designs. PathPoint makes runtime analysis of online transactions or batch jobs and provides detailed information about program and data usage and the performance of programs and SQL statements.

The speaker will tell how PathPoint was used to secure the quality of the CICS backend in a project for a complex webservice-based application.

Jørgen Møller Larsen has 20 years of experience working with mainframe-centric environments in the Danish financial sector, and is responsible for the use and implementation of development, testing and performance tools in Nykredit.

S18 – Profiling mainframe applications

by Jørgen Møller Larsen, Nykredit, Denmark

GSE Nordic 28-30 May 2008, Elsinore, Denmark

- **Nykredit is one of Denmark's leading financial services groups with activities ranging from mortgage lending and banking to insurance, pension and estate agency business.**
- **The Nykredit Group is the second largest lender and the largest mortgage provider in Denmark as well as one of the largest private bond issuers in Europe.**
- **Read more on www.nykredit.com.**

- **IT development in Nykredit has 400 employees in Aalborg, Copenhagen and Warsaw.**
- **Member of the developer support team.**
- **Responsible for the use and implementation of development, testing and performance tools in Nykredit.**

Our platform(s):

**IBM z9 mainframe
z/OS operating system
DB2 database
CICS transaction server**

**Windows and AIX servers
BEA Weblogic application
server
Oracle**

Use of profilers

From Wikipedia

A profiler is a performance analysis tool that measures the behavior of a program as it runs, particularly the frequency and duration of function calls. The output is a stream of recorded events (a trace) or a statistical summary of the events observed (a profile). Profilers use a wide variety of techniques to collect data, including hardware interrupts, code instrumentation, operating system hooks, and performance counters.

Performance analysis

From Wikipedia

In software engineering, performance analysis, more commonly profiling, is the investigation of a program's behavior using information gathered as the program runs (i.e. it is a form of dynamic program analysis, as opposed to static code analysis). The usual goal of performance analysis is to determine which parts of a program to optimize for speed or memory usage.

Programming languages:

- ASSEMBLER
- C++
- Java
- COBOL
- 4GL – AllFusion Gen (CA)

Profiling tools:

- TMON for CICS (ASG)
- BMC Apptune
- STROBE (Compuware)
- PathPoint (ASG)

TMON for CICS (ASG)

Collects statistics on each transaction that runs on a CICS system.

Memory usage, I/O, DB2 calls, response time, wait time, CPU time etc.

Totals pr. transaction.

BMC Apptune

Collects statistics on each SQL statement that is run on a DB2 system.

Elapsed time, I/O, # of calls, wait time, CPU time etc.

Totals pr. statement pr. program (package).

STROBE (Compuware)

Creates a profile for a CICS region.

Uses sampling i.e. makes snapshots of the activity on the CICS region

Reports on top resource consumers based on CPU and wait time.

PathPoint (ASG)

Traces all activity on a CICS region by a specified user.
Reports alle kinds of stuff (more on that later).

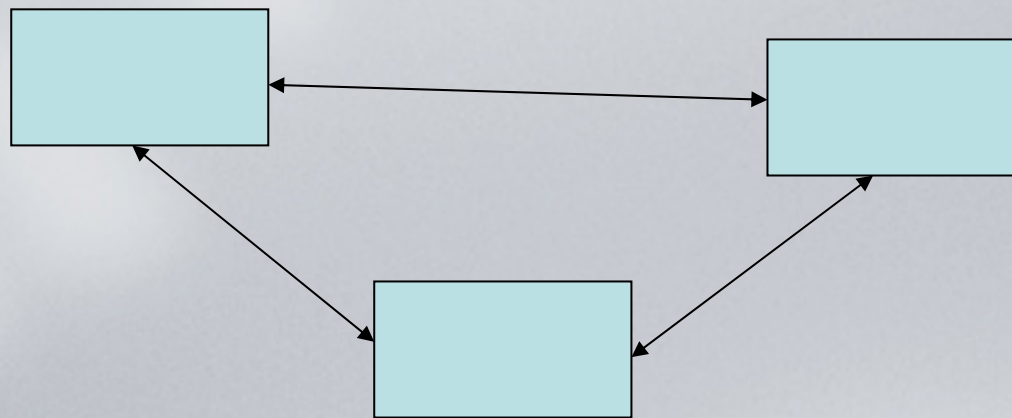
A brief history of application architecture (as I remember it)

1. generation – the monoliths
2. generation – the components
3. generation – the web-services

A brief history of application architecture

1. generation – the monoliths

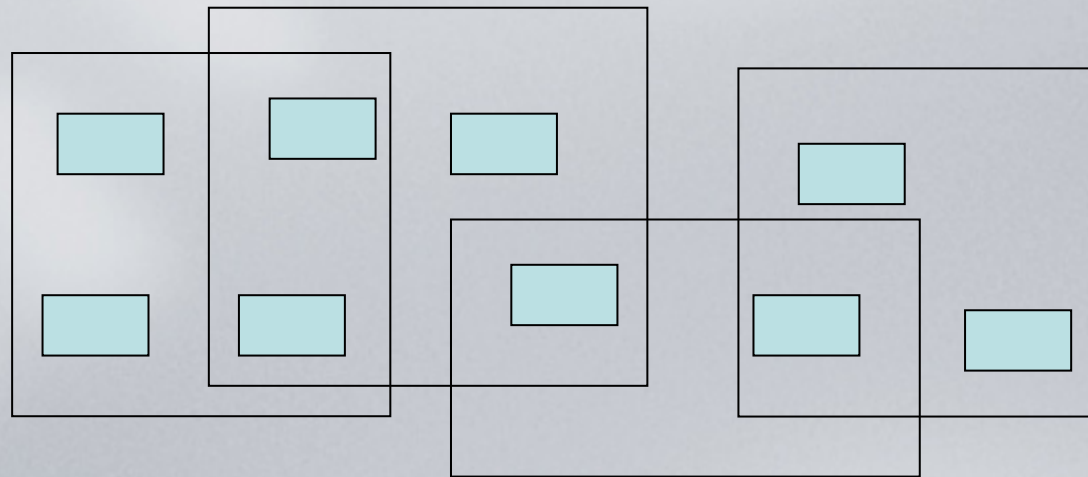
- Self contained systems with limited integration
- Non-reuseable code, redundant functionality
- Costly maintainence
- Clear responsibility



A brief history of application architecture

2. generation – the components

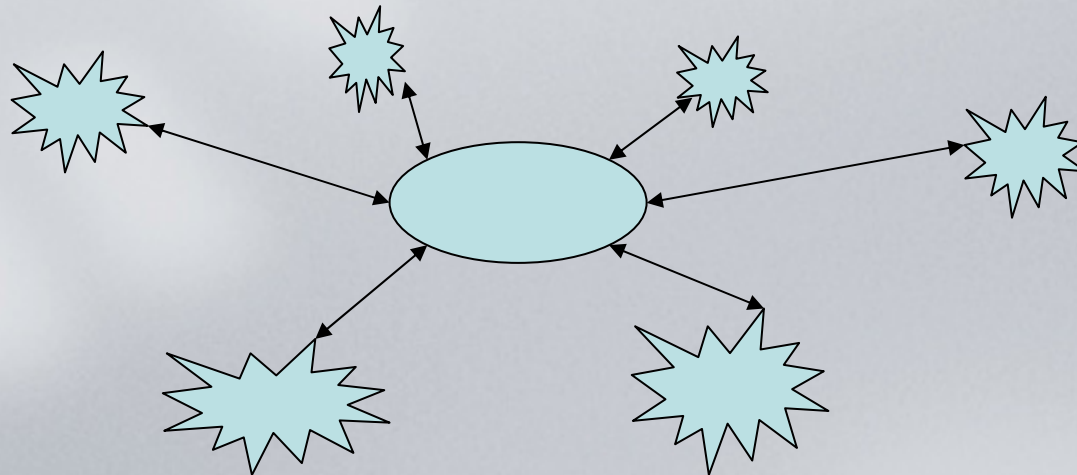
- Systems made of interacting reuseable components
- Increased complexity and abstraction
- Responsibility is diversified



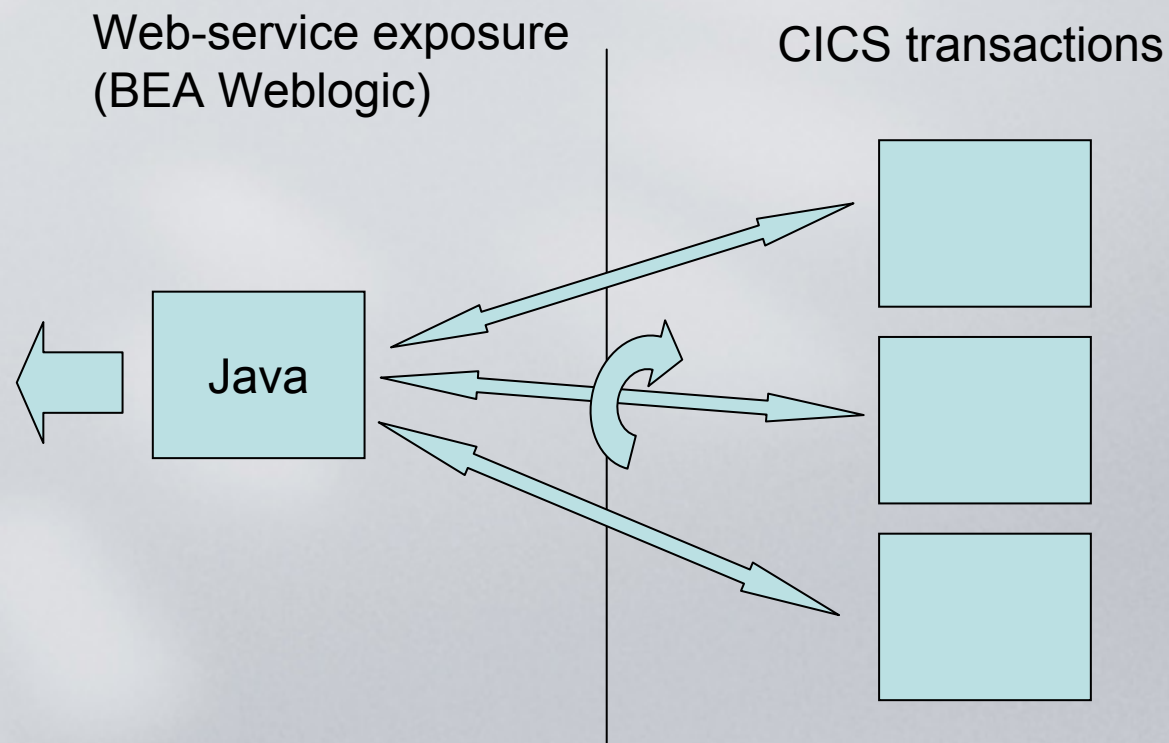
A brief history of application architecture

3. generation – the web-services

- Systems made out of services
- Totally abstract and extremely complex
- Mixed platforms, technologies and locations (even companies)



A typical web-service implementation in Nykredit



The CICS transactions are constructed using AllFusion Gen 4GL.
The Java programmers interface is a Java-class generated from the 4GL (a proxy server).

Basically the web-services are wrapped-up components.

This may be great from an architectural point of view, but one might ask a couple of questions:

1. What happens when complexity increases that much?
2. Will there be performance issues?
3. How do I ensure acceptable performance for the end user?

Let's imagine a (not so unlikely) scenario: we have a web-service that times out because of long response time.

How do we go about fixing it?

First step is to identify the CICS transaction(s) that make up the web-service. Hopefully we can do that.

Challenge: the Java and mainframe programmers are different "species" that don't communicate very well.

Different terms and tools: the Java programmer knows about "classes", "methods", "objects" and "log4j" – the mainframe programmer knows "transaction codes", "programs", "operations", "entities" and "SYSLOG".

Two ways of identifying the transactions:

1. Read the Java code (which would be static code analysis – not profiling).
2. Get someone to run the web-service, and use TMON for CICS to see which transactions it spawns off.

This works best if we are almost alone on the CICS system, otherwise we wont be able to see which transactions originates from the web-service.

TMON for CICS search screen

```

JOBNAME: NKPPCICS          CURRENT DATA SEARCH CRITERIA          DATE: 05/19/08
APPLID  : NKPPCICS H SYS: NK04          CSM: NKPDTMON V=L          TIME: 7:46:33
COMMAND: _____          CYCLE: MMSS

JOBNAME:  NKPPCICS  SCAN LIMIT: 0001000  TIME LIMIT: MMSS  DISP LIMIT: 1000
START DATE: 051908  START W/1ST REC:  N  IGNORE FILTERS AFTER 1ST TRANS:  N
START TIME: HMMSS   START W/LAST REC:  N  ALL SEARCH FILTERS MUST MATCH:  Y
(MUST SET TO REFRESH)

SEARCH FILTERS TO BE USED TO PRODUCE THE LIST OF TRANSACTIONS

TRANID(S): _____  TERMID(S): _____  WORKLOAD: _____
OPERID(S): _____  LUNAME(S): _____  SRVCLASS: _____
USERID(S): _____  USRFLD(S): _____  RPTCLASS: _____
PGRMID(S): _____  FILEID(S): _____  TRNCLASS: _____

ABEND CODE (S): _____
ABENDING PGM(S): _____

SELECT TRANSACTIONS ONLY WHEN:

RESP > SSSTTT  EIP REQUEST TIME > SSSTTT OR NN % DSA USED > NNNNN K
CPU > SSSTTT  EIP WAIT TIME > SSSTTT OR NN % EDSA USED > NNNNN K
DISP > SSSTTT OR NN % EXCEPTION TIME > SSSTTT OR NN % PAGE IN/OUT > NNNNN
WAIT > SSSTTT OR NN % TOT FILE IO TIME > SSSTTT OR NN % TOT FILE IO > NNNNN

PRESS DOWN FOR FLAG FILTERS          USE DISPLAY PROFILES: N
HELP INFORMATION = PF1          NKPDTMON          PF KEY ASSIGNMENTS = PA1
    
```

We can filter on time, transaction code and userid.

TMON for CICS search result

```

JOBNAME: NKPD TMON          TA          -TRANSACTION ACTIVITY          DATE: 05/19/08
APPLID  : NKPD TMON          TIME: 7:49:46
COMMAND: _____          CYCLE: MMSS

DISPLAYED: 16 TO 31 OF 1000          SORT COLUMN# : __ (1-10)          ORDER: __ (A/D)
DATE      TIME      JOBNAME  TRANID  TASK#  TERMID  RESP      CPU      PAGING  FILEIO
*         *         *         *       *     *      *         *       *       *
05/19 07:00:13 NKPPCICS H57N   22488  3.4784  1.3119  0      7,440
05/19 07:00:15 NKPPCICS T20J   22490  0.5095  0.0151  0       45
05/19 07:00:15 NKPPCICS H54H   22492  0.0092  0.0069  0       15
05/19 07:00:15 NKPPCICS H59M   22494  0.0041  0.0026  0        2
05/19 07:00:15 NKPPCICS H54T   22496  0.0142  0.0099  0       27
    
```

H57N could be the culprit – long response time, high CPU usage, and a lot of I/O.

Let's take a closer look.

TMON for CICS – transaction timings

```

JOBNAME: NKPDITMON          TRANSACTION TIMINGS          DATE: 05/19/08
APPLID : NKPDITMON          TIME: 7:57:00
COMMAND: █                  CYCLE: MMSS
                                _NEXT? _GNXT? _MENU?

DISPLAYED: 17 TO 32 OF 63
JOBNAME: NKPPCICS          TRANSACTION: H57N  #22488
TIMING TYPE                COUNT      TOT TIME    AVG TIME    % OF RESPONSE
WAIT ON REDISPATCH         0          0.0000     0.0000     0.00
-----
* REQUEST SUMMARY
-----
TOTAL I/O REQUESTS        7,440      2.6564     0.0004     76.36
MRO/ISC REQUESTS          0          0.0000     0.0000     0.00
EXCEPTION WAIT            0          0.0000     0.0000     0.00
TERMINAL WAIT             0          0.0000     0.0000     0.00
STORAGE WAIT              0          0.0000     0.0000     0.00
ALL EIP REQUESTS          97,870     0.1297     0.0000     3.72
ALL EIP REQ WAIT          8          0.0008     0.0001     0.02
FCP REQUESTS              0          0.0000     0.0000     0.00
FCP REQ WAIT              0          0.0000     0.0000     0.00
DLI REQUESTS              0          0.0000     0.0000     0.00
DLI REQ WAIT              0          0.0000     0.0000     0.00
DB2 REQUESTS              7,440      2.6564     0.0004     76.36
    
```

HELP INFORMATION = PF1 NKPDITMON PF KEY ASSIGNMENTS = PA1

So, the time is spent in DB2 on I/O.

And thats about as far as TMON for CICS will get us.

We have identified which transaction is the problem, and that the time is spent in DB2, so now we need to identify the actual code (SQL) that takes the time.

TMON for CICS wont tell you which modules or which SQL-statements that were executed.

We will try another tool: BMC Apptune

```

ASQEQRPW/1                               View a Report                               LINE 4 OF 1218
Command ===> _____ Scroll ==> CSR_

BMCSftwr.SQMCACTR  --      PROGRAM ANALYSIS (DATA)      --      19/05 08:09:28
Source : DC04-ACTIVE  Intvl : 19/05 07:00 - 19/05 07:59  More:  - + >
-----
Actions: D-Subsys L-CorrID P-Plan          N-ClntAp E-SQL Error O-Object H-Header
         C-ConnID U-User   V-SAP          I-ClntID X-Exception Q-CatSQL A-Average
         G-AppGrp T-Detail S-Stmt         W-ClntWS M-AccMatr          Z-zIIP

Subsys:          CorrID:          Plan:          ClntAp:
ConnID:          User:            StmtNo:        ClntID:
AppGrp:                                     ClntWS:

Program  Pgm  SQL  +----- Total IN-SQL Time -----+  +- Sync
         Type Calls Elapsed % CPU % Getpage Number
-----
+ 02156A29 PKGE 65176 07:48,26842 8,3% 00:05,75733 0,8% 209456 52755
+ 02500B33 PKGE 39908 05:05,79814 5,4% 00:04,42413 0,6% 175161 49266
+ 02550A08 PKGE 50332 04:40,74491 5,0% 00:11,34773 1,6% 293966 38599
+ 02500A27 PKGE 9458 02:18,90624 2,5% 00:02,26007 0,3% 30013 10381
+ 02500A45 PKGE 36808 02:15,23762 2,4% 00:03,94087 0,6% 112247 14879
+ 02550A26 PKGE 15871 01:58,44562 2,1% 00:03,20342 0,5% 69947 14810
    
```

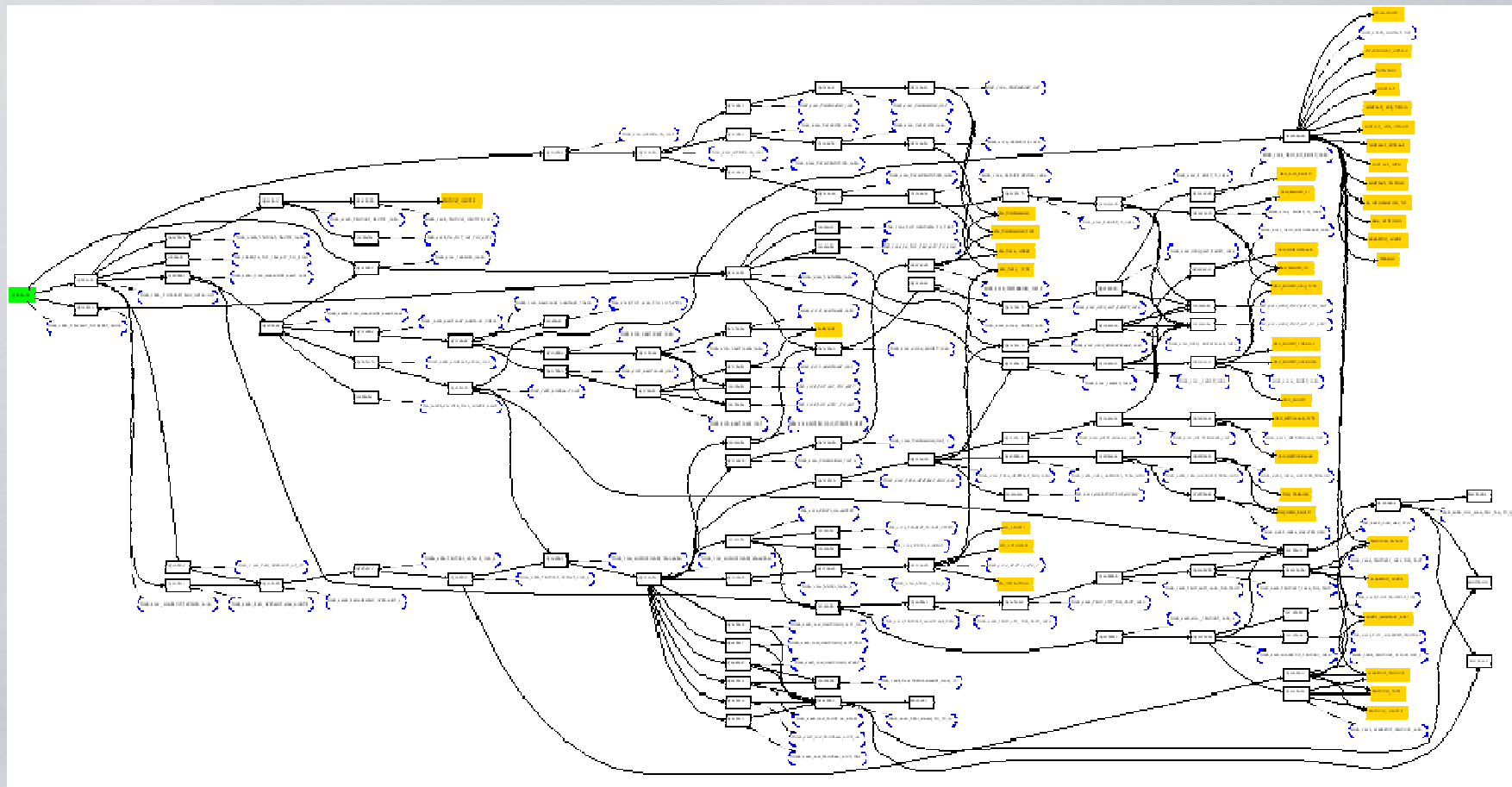
Apptune will show us the modules that spends most time in DB2.
 Maybe one of these is the cause of our problem.

The problem with Apptune is that it will not tell us which transactions called these modules...

We can make static code analysis on our transaction, to find out if some of these modules are called from our transaction.

If you have some kind of cross-reference tool that may be fairly easy, if not ... it's hard work.

What a CICS transaction might look like in our X-ref.



(I deliberately chose one of the less complex transactions...)

Even if we establish that one of the modules detected in Apptune is called from our transaction, it may not be the cause of our long response time.

Hi-use modules will rank high in Apptune (and may very well have a tuning potential) - but it may not be our problem.

Low-use modules will not stand out in Apptune, and therefore they may not attract our attention.

Luckily we're not out of tools yet – let's try STROBE.

We will start a STROBE measurement session, and get someone to run the web-service, and hope that STROBE catches something.

The STROBE report on our transaction

Transaction	Count	Margin of error	CPU %	Delay	Suspend	Exec	Total
H54Y	19		1.80	0.02	0.02	0.02	0.05
H54Z	69		3.28	0.01	0.01	0.01	0.03
H57L	1		2.38	0.67	0.11	0.46	1.24
H57N	2	4.41	4.52	0.85	0.07	0.43	1.35
Non-API totals				0.85	0.00	0.30	
Resource type				Delay	Suspend ↓		
.DELAY				0.41	0.00		
.DELAY				0.45	0.00		

A lot of TCB-switches = a lot of DB2 activity. We knew that already. Problem with a sampling profiler – we'll need more load to make the significant results.

We have one last tool - PathPoint.

We will start a PathPoint measurement session, and get someone to run the web-service, and see what PathPoint can show us.

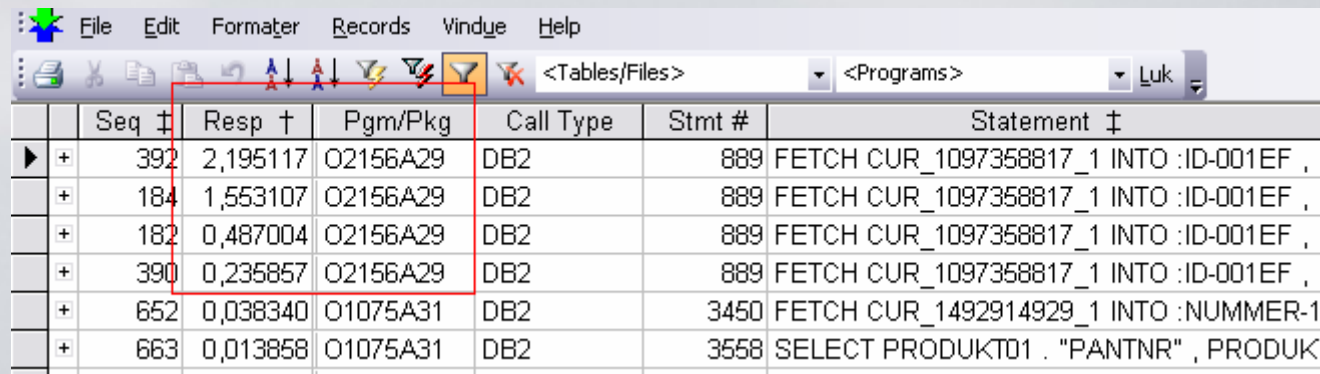
I don't even have to know the transaction – PathPoint will trace anything that the user runs!

The PathPoint report on our transaction in the GUI

Seq	Resp	Pgm/Pkg	Call Type	Stmt #	Statement	Status	Interim
1	0,000000	CICS	CICS/INIT		CALL C1075011 (INIT) IN CO2		0,000204
2	0,000101	C1					,000221
3	0,000104	C1					,000272
4	0,000133	C1075011	CICS/TS	x'32106'	WRITEQ TS QUEUE(SIP02455)		0,000804
5	0,000410	C1075011	CICS/TD	x'324CC'	WRITEQ TD QUEUE(TISL)		0,004491
6	0,000095	C1075011	CICS/TS	x'171F4'	WRITEQ TS QUEUE(0002455)		0,000674
7	0,000000	-	CICS/CALL	x'184BA'	CALL TIRENTC (LOAD) IN ASM		0,000152
8	0,000000	-	CICS/CALL		CALL Q1075031 (LOAD) IN COB		0,000741
9	0,000000	-	CICS/CALL		CALL Q1075A31 (LOAD) IN COB		0,000084
10	0,000000	-	CICS/CALL		CALL O1075A31 (LOAD) IN COB		0,000078
11	0,000000	-	CICS/CALL		CALL Q2514009 (LOAD) IN COB		0,000211
12	0,000000	-	CICS/CALL		CALL Q2514A09 (LOAD) IN COB		0,000723
13	0,000000	-	CICS/CALL		CALL O9062361 (LOAD) IN COB		0,000105
14	0,000000	-					0,000062
15	0,000000	-					0,000052
16	0,000000	-	CICS/CALL		CALL O2486A16 (LOAD) IN COB		0,000088
17	0,000361	O2486A16	DB2	782	OPEN CUR_0418131245_1	0	0,001317
18	0,000048	O2486A16	DB2	801	FETCH CUR_0418131245_1 INTO :BRAND-IDE	0	0,000291
19	0,000020	O2486A16	DB2	801	FETCH CUR_0418131245_1 INTO :BRAND-IDE	0	0,000705
20	0,000615	O2486A16	DB2	801	FETCH CUR_0418131245_1 INTO :BRAND-IDE	100	0,000252
21	0,000028	O2486A16	DB2	843	CLOSE CUR_0418131245_1	0	0,001148
22	0,000000	-	CICS/CALL		CALL L2486A14 (LOAD) IN COB		0,000182

Provides a complete program flow for the actual run of the transaction.

The PathPoint report on our transaction



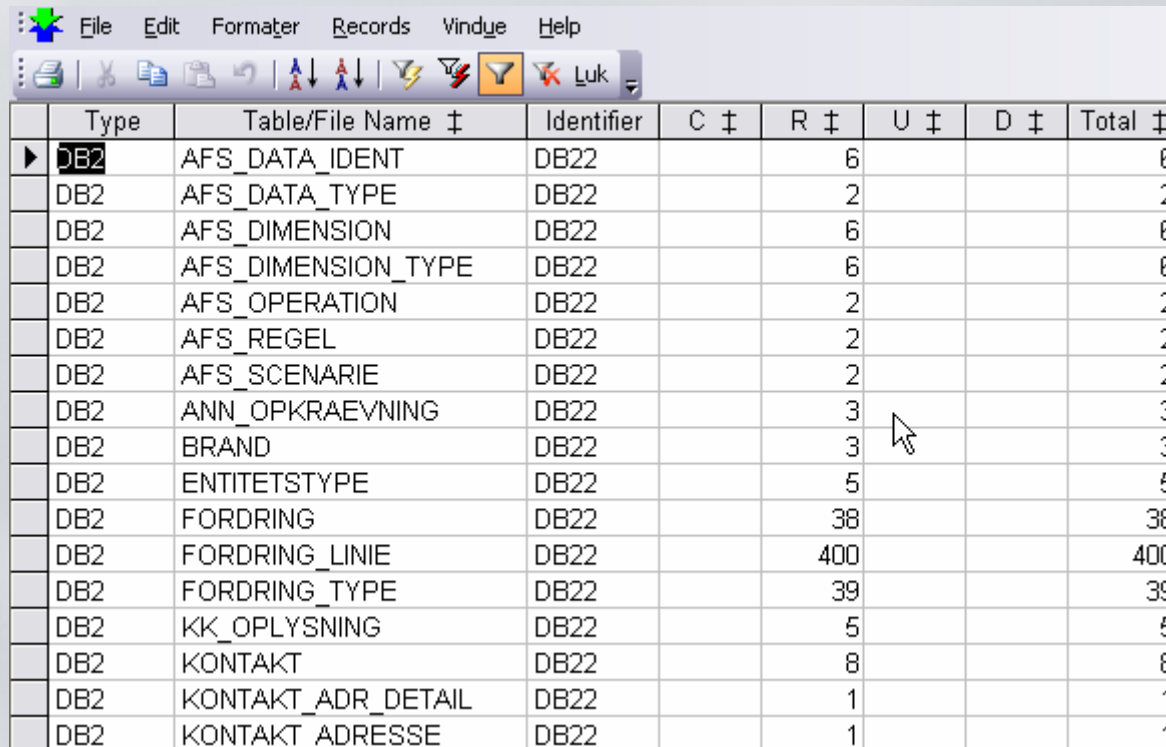
	Seq ‡	Resp †	Pgm/Pkg	Call Type	Stmt #	Statement ‡
▶ +	392	2,195117	O2156A29	DB2	889	FETCH CUR_1097358817_1 INTO :ID-001EF ,
+	184	1,553107	O2156A29	DB2	889	FETCH CUR_1097358817_1 INTO :ID-001EF ,
+	182	0,487004	O2156A29	DB2	889	FETCH CUR_1097358817_1 INTO :ID-001EF ,
+	390	0,235857	O2156A29	DB2	889	FETCH CUR_1097358817_1 INTO :ID-001EF ,
+	652	0,038340	O1075A31	DB2	3450	FETCH CUR_1492914929_1 INTO :NUMMER-1
+	663	0,013858	O1075A31	DB2	3558	SELECT PRODUKT01 . "PANTNR" , PRODUK

Sorting by response time reveals the module and the DB2 statement that causes most of the response time for the transaction.

That was easy, wasn't it?

And the PathPoint report contains a wealth of other information.

The PathPoint report on our transaction

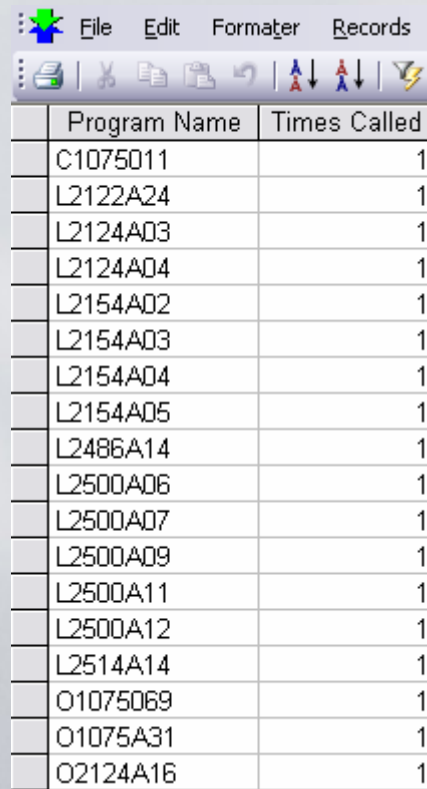


The screenshot shows a software window with a menu bar (File, Edit, Formater, Records, Vindue, Help) and a toolbar. Below the toolbar is a table with the following columns: Type, Table/File Name †, Identifier, C †, R †, U †, D †, and Total †. The table lists 18 database tables, all identified as DB2. The 'R †' column shows the number of reads for each table, with 'FORDRING' having the highest value at 38. The 'Total †' column shows the sum of all operations for each table.

Type	Table/File Name †	Identifier	C †	R †	U †	D †	Total †
DB2	AFS_DATA_IDENT	DB22		6			6
DB2	AFS_DATA_TYPE	DB22		2			2
DB2	AFS_DIMENSION	DB22		6			6
DB2	AFS_DIMENSION_TYPE	DB22		6			6
DB2	AFS_OPERATION	DB22		2			2
DB2	AFS_REGEL	DB22		2			2
DB2	AFS_SCENARIO	DB22		2			2
DB2	ANN_OPKRAEVNING	DB22		3			3
DB2	BRAND	DB22		3			3
DB2	ENTITETSTYPE	DB22		5			5
DB2	FORDRING	DB22		38			38
DB2	FORDRING_LINIE	DB22		400			400
DB2	FORDRING_TYPE	DB22		39			39
DB2	KK_OPLYSNING	DB22		5			5
DB2	KONTAKT	DB22		8			8
DB2	KONTAKT_ADR_DETAIL	DB22		1			1
DB2	KONTAKT_ADRESSE	DB22		1			1

A CRUD matrix for all tables and files.

The PathPoint report on our transaction

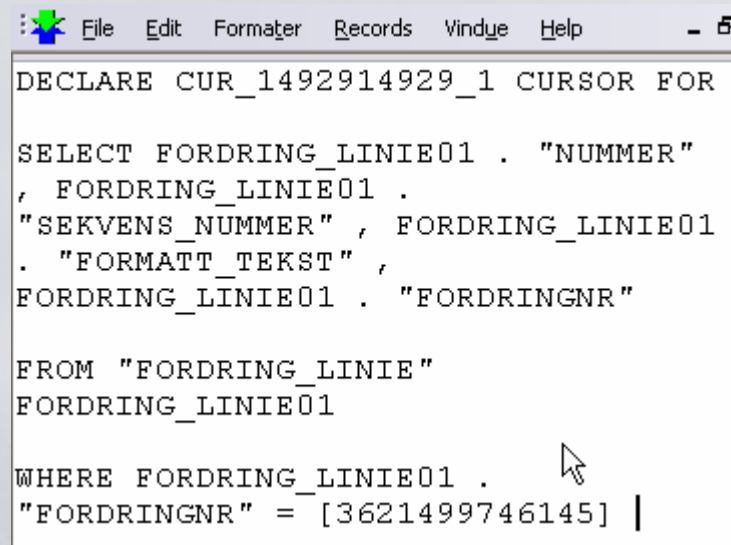


The screenshot shows a software window with a menu bar (File, Edit, Formater, Records) and a toolbar. Below the toolbar is a table with two columns: 'Program Name' and 'Times Called'. The table lists 20 different program names, each with a value of 1 in the 'Times Called' column.

Program Name	Times Called
C1075011	1
L2122A24	1
L2124A03	1
L2124A04	1
L2154A02	1
L2154A03	1
L2154A04	1
L2154A05	1
L2486A14	1
L2500A06	1
L2500A07	1
L2500A09	1
L2500A11	1
L2500A12	1
L2514A14	1
O1075069	1
O1075A31	1
O2124A16	1

A list of all invoked programs.

The PathPoint report on our transaction

A screenshot of a mainframe terminal window. The window has a menu bar with the following items: File, Edit, Formater, Records, Vindue, Help, and a status bar with a minus sign and the number 5. The main area of the window contains the following SQL code:

```
DECLARE CUR_1492914929_1 CURSOR FOR  
  
SELECT FORDRING_LINIE01 . "NUMMER"  
, FORDRING_LINIE01 .  
"SEKVENNS_NUMMER" , FORDRING_LINIE01  
. "FORMATT_TEKST" ,  
FORDRING_LINIE01 . "FORDRINGNR"  
  
FROM "FORDRING_LINIE"  
FORDRING_LINIE01  
  
WHERE FORDRING_LINIE01 .  
"FORDRINGNR" = [3621499746145] |
```

Content of host-variables for each SQL-statement invocation is collected too.

Nykredit used PathPoint to ensure quality in a major SOA-project where independent teams developed web-services.

All web-services were required to run tests with PathPoint.

We would look for excessive response times, excessive data access, odd return codes (-811, -501 indicating poor logic).

Profiling is not just about performance:

Use it to understand your application logic when documentation is less than perfect (that happens).

Use it for debugging when error handling is less than perfect (that happens too).

Different profiling tools for different purposes:

TMON for CICS, BMC Apptune and STROBE is great when you want an overall picture on CICS and DB2 performance.

PathPoint is my tool of choice if I have a performance problem in a specific application.

Questions?

Comments?

If you come to Aalborg,
I'll be happy to show you more.

jml@nykredit.dk