AP4Z™

WATSON & WALKER APPLICATION PROFILER FOR Z

Cheryl Watson
Watson & Walker
cheryl@watsonwalker.com
ap4z@watsonwalker.com

Watson Walker

Table of Contents

Who are we?

- Watson & Walker was founded in 1988 by Cheryl Watson & Tom Walker.
- Publisher of *Cheryl Watson's Tuning Letter* and *CPU Charts* since 1991. If you have never seen one of our newsletters, send an email for a free copy.
- After the *Tuning Letter*, our primary focus has been helping our customers reduce their software costs. We use our SCRTPro tools for studies such as Tailored Fit Pricing (TFP).
- We are completely independent, not beholden to any vendor, so we can offer objective information based on our collective experience and what we see in other customers, allowing clients to make a fully informed decision.
- In addition to our publications, our team provides consulting, classes, and software products (both free and chargeable) to help our customers.

Background

- Historically the focus for performance tuning has been more on infrastructure optimization than on applications. This is because of:
  - Better documentation and tooling.
  - More widely available infrastructure tuning skills.
  - Small changes having wide impact.
- *But:*
  - *The efficiency of commercial software products is usually better than that of applications.*
  - *The infrastructure is usually changed less frequently than applications, and usually in a more controlled manner.*
  - *After so many years of infrastructure tuning the opportunities for significant improvements are lessening.*

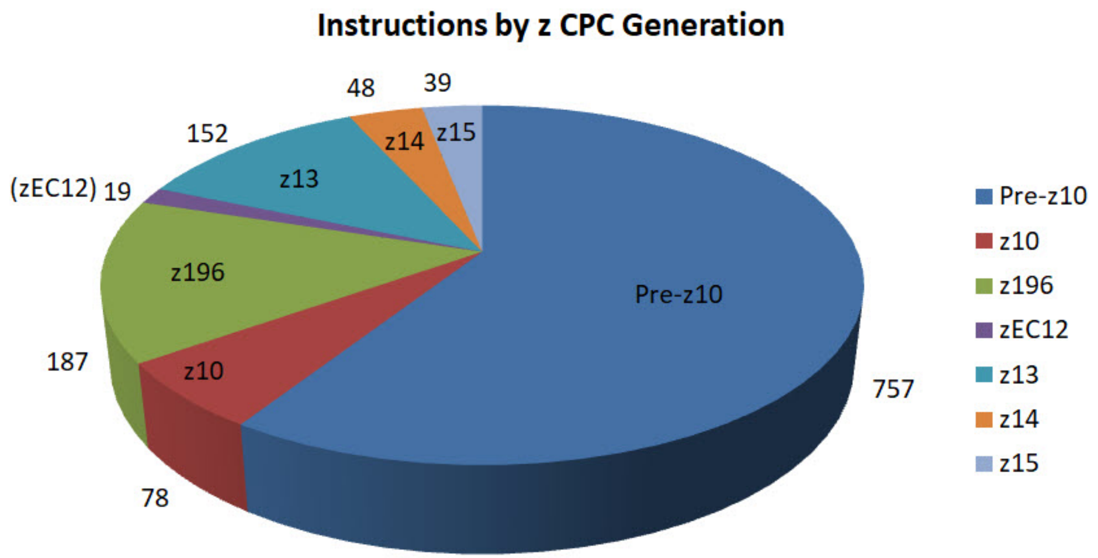
Issues with application tuning

- Application tuning may provide significant benefits, but:
  - Application teams are typically under pressure from their funders to release new functions – they can’t spend a long time to find tuning opportunities for existing applications.
  - There is no clear owner for application tuning initiatives:
    - IT infrastructure people can’t perform the tuning on their own.
    - Application developers don’t pay the software bills, so they have little incentive.
  - The required skills and techniques cross the boundaries between infrastructure and applications, with very few tools, if any, specifically aimed at supporting large scale application performance analysis.
  - With the looming end-of-service (April 30, 2022) for COBOL V4 and migration to COBOL V6, teams are struggling with a lack of tools to identify the most important programs to work on..

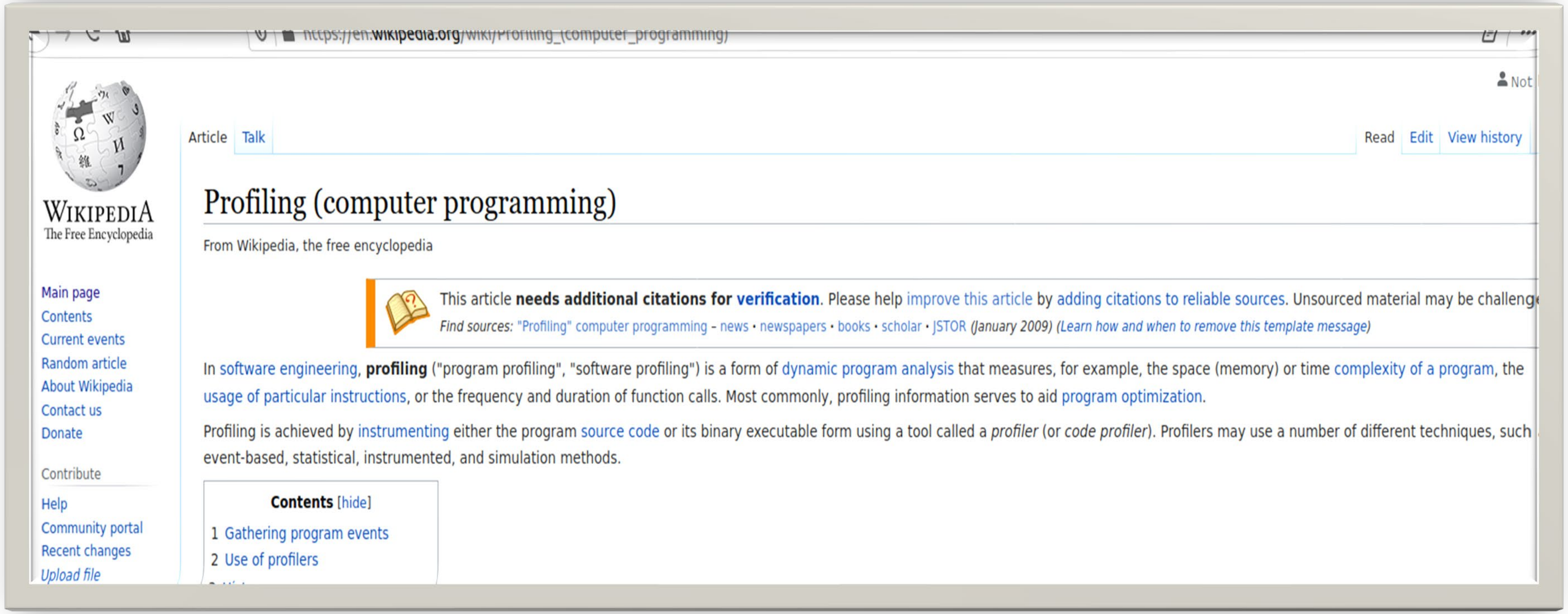
All the above prompted us to consider developing an application workload profiler

COBOL – Why the push?

- There are two driving forces
  - COBOL V4 compiler goes out of service on April 30, 2022. The compiled programs can run fine if they are LE-enabled. But if recompiled, problems can be reported only on COBOL V6+. So, the migration would need to be done at that time.
  - Companies want to reduce CPU time in order to reduce software costs. COBOL V6 allows them to do that with ARCHLVL compiler options (some clients can see a 20% CPU reduction, while others don't). The chart at the rights shows that over 40% of all z/OS instructions can be used only on higher ARCHLVLs. Many of them are designed to reduce the CPU time for decimal calculations (i.e., a large part of COBOL apps).
- Migration to COBOL V6 can be difficult
  - Many sites have thousands of COBOL programs. A tool is needed to find those most frequently loaded and those using the most CPU time.



What is application profiling?



An application profiler monitors the actual execution of application code to collect information (CPU time, number of calls, etc.) about noticeable events (calls to other programs, error conditions, memory management, etc.) to aid program optimization.

Our idea of a z/OS application profiler

- We created a tool that is:
  - Specifically designed to profile application execution (only applications).
  - Aimed at providing a full picture about programs / subroutines and their actual relationships.
  - Easy to use, with a negligible, measureable impact on CPU consumption, to allow it to always be running.
  - Able to report elapsed time and CPU consumption at the individual program level.
  - Able to collect each module’s compile data (compile date, compiler release, compiling options).
  - Able to collect information to build a call graph, showing who calls who.

What is available today?

- Load Module Analyzers
  - Static ("point in time") view of load libraries, including compiler data (version, date, options). Can't tell if used.
- Software Asset Management Tools
  - Track program loads, but main focus is on software license utilization compliance. Don't provide call graphs or any application-oriented information (like compiler data).
- Execution Samplers
  - Deep dive analysis of specific programs' behavior down to the single instruction. But due to overhead, can be used only on a few programs at a time.
- SMF Type 30 Data
  - Everyone collects these, so no additional overhead, but is usually used to collect step-level (not module-level) CPU and I/O counts. It doesn't provide call graphs or any application-oriented information (like compiler data).

Load Module Analyzers

Software Asset Management Tools

Execution Samplers

SMF Data

Our solution – AP4Z

Watson & Walker Application Profiler for Z - AP4Z

AP4Z works with existing load modules and JCL procedures.

- Mario Bezzi, author of W&W free tools, *WWUNTERSE* and *IMPORTANT_MESSAGES Health Check*, was the designer and lead developer.
- Looks at programs actually executed rather than static load libraries.
- Does not require source code changes or recompiles.
- No changes to existing JCL.
- Does not install system-wide hooks or user exits.
- Doesn’t require APF-authorization.
- Supports Dynamic (most common) and Static calls.
- Supports COBOL, PL/I, C/C++, and Assembler (if LE-enabled).

Our solution – AP4Z

Watson Walker

- AP4Z can trace Module Load activity, Program Call activity, Memory Management and Condition Handling.
- It can optionally provide information about the program call tree and a detailed Program Call Trace.
- Options can be enabled separately, and the level of profiling can change for different Jobs / Programs.
- Base AP4Z collector for batch jobs and Db2 stored procedures will be available September 2021.
- CICS and additional functions expected to ship by YE 2021.

Data we collect

Job Step Level	
CPC model	z/OS level
Sysplex name	System name
Userid	Job name
Job-id	Job start date-time
Step name	Step program name
Step start date-time	# created enclaves
# created threads	#/type memory req
# handled conditions	

Module Level	
Module name	Load count
Call count	Elapsed time
CPU times	Amode
Routine type[1]	Compiler version
Compile date	Code page option
All compile options	

[1] LE conforming, Fastlink, IEEE floating point, XPLINK, DLL

AP4Z – Application Performance Profiling Tool

Watson Walker

Performance impact of running AP4Z is negligible and can be tracked. For this reason, we recommend always keeping it on. If needed, the scope of profiling can be limited to specific programs and/or jobs. Different objects can use different profiling options.

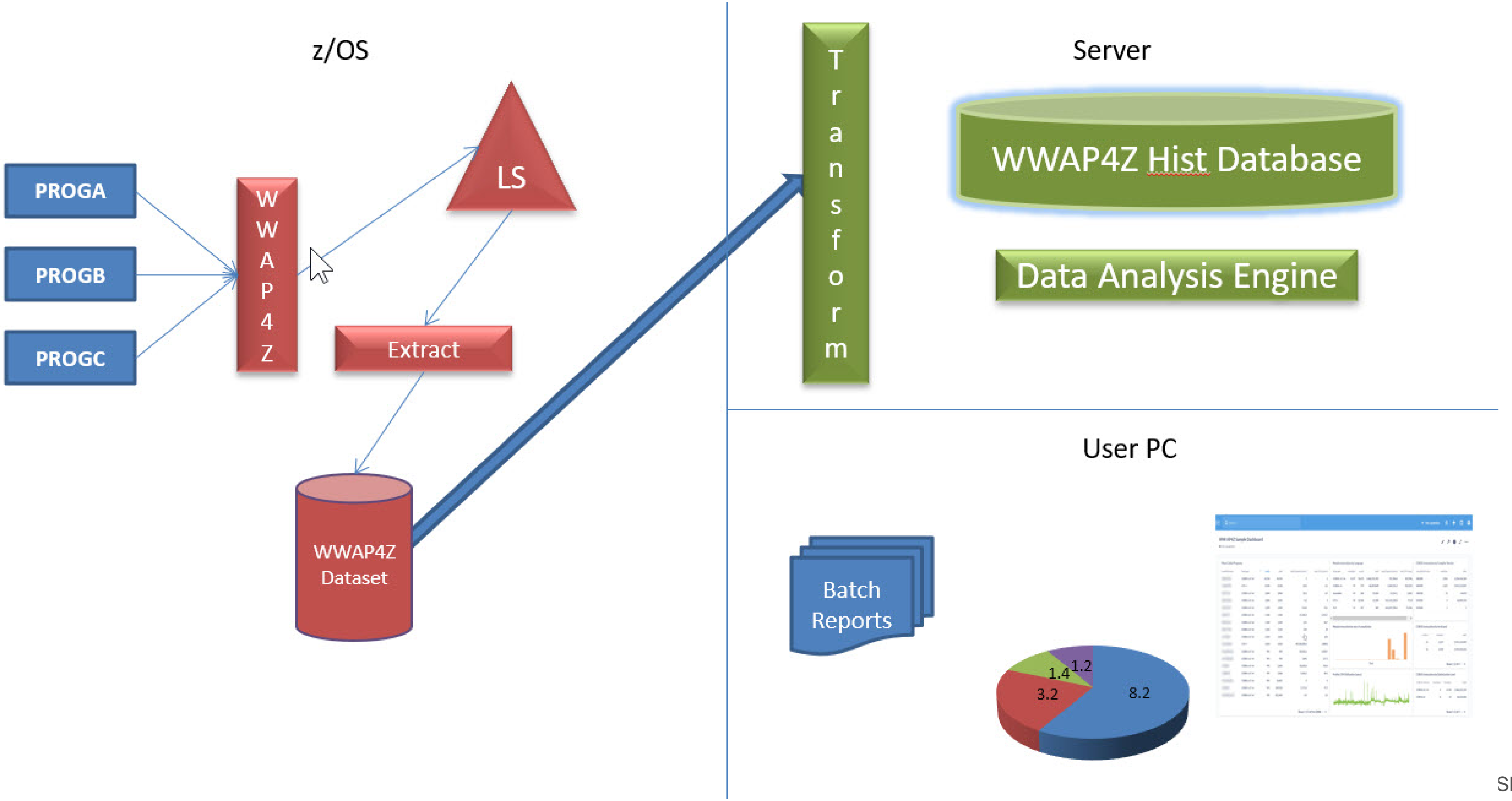
Over time, AP4Z builds a history of which programs are in use, how they perform, how they contribute to the overall elapsed and CPU time, how they relate to each other, when they were recompiled, and what options were used during each compile.

Collected data is loaded into a historical database. Data is analyzed using a browser-based graphical user interface that provides both pre-defined reports / charts and on-demand queries. Using SQL is an option, but not required.

Based on our early measurements the profiling overhead should be below 0.5% for batch and below 1% for CICS

AP4Z – Application Performance Profiling Tool

Watson Walker



Sample AP4Z Dashboard - PGMLOADs

Application Profiler Summary Data

Most used Modules

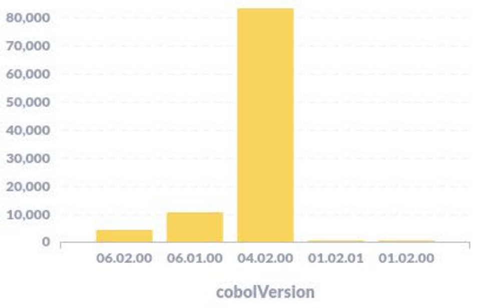
moduleName	language	archLvl	optimizeOption	loadCount
[illegible]	COBOL	NA	N	37,634
[illegible]	COBOL	NA	N	37,282
[illegible]	C/C++	N/A	N/A	11,079
[illegible]	COBOL v5/v6	7	0	916
[illegible]	COBOL	NA	N	894
[illegible]	COBOL	NA	N	775
[illegible]	COBOL v5/v6	7	0	710
[illegible]	COBOL v5/v6	7	0	710
[illegible]	COBOL	NA	N	710
[illegible]	COBOL	NA	N	710
[illegible]	COBOL	NA	N	463
[illegible]	COBOL v5/v6	7	0	443
[illegible]	C/C++	N/A	N/A	441
[illegible]	COBOL	NA	N	436
[illegible]	C/C++	N/A	N/A	386
[illegible]	COBOL	NA	N	330
[illegible]	COBOL	NA	N	330
[illegible]	COBOL v5/v6	12	0	330
[illegible]	COBOL	NA	N	328
[illegible]	COBOL v5/v6	11	2	316
[illegible]	COBOL v5/v6	11	2	316

Rows 1-21 of 1611

COBOL loads by year of compilation

Year	loadCount
2021	5,356
2020	600
2019	578
2018	6,446
2017	4,515
2016	517
2015	35,822
2014	44,641
2009	1
2000	151
1998	3

COBOL loads by compiler version



Module loads by Language

language	loadCount
COBOL for OS/390 & VM, COBOL for MVS & VM	83,707
COBOL v5 and v6	14,923
OS/390 C/C++, C VM/ESA, XL C/C++	12,603
VisualAge PL/I for OS/390	334
ASSEMBLER	75

COBOL Loads by compiled ARCHLVL

archLvl	loadCount
NA	83,707
12	2,913
11	7,978
7	4,032

COBOL Loads by compiled optimization level

optimizeOption	loadCount
N	83,707
2	10,561
0	4,362

Really old COBOL modules

moduleName	cobolVersion	compilationYear	loadCount
[illegible]	01.02.00	1998	2
[illegible]	01.02.00	1998	1

Profiler CPU Consumption

Powered by Metabase

Sample AP4Z Dashboard - PGMCALLs

WW AP4Z Sample Dashboard

Our analytics

Most Called Programs

moduleName	language	Loads	calls	Avg Elapsed (usecs)	Avg CPU (usecs)
[illegible]	COBOL v5 / v6	18	4,290,642,435	13.5	2.8
[illegible]	COBOL v5 / v6	6	1,444,955,475	53.5	17.8
[illegible]	COBOL v5 / v6	58	263,158,716	8,304.5	139.8
[illegible]	COBOL v5 / v6	11	257,997,146	6.5	1.8
[illegible]	COBOL v5 / v6	9	145,774,606	31,586.8	251.1
[illegible]	COBOL v5 / v6	7	109,076,525	34	7.1
[illegible]	COBOL v5 / v6	26	83,932,126	167.2	17.8
[illegible]	COBOL v5 / v6	2	66,592,004	1.5	0
[illegible]	COBOL v5 / v6	3	61,869,628	0	0
[illegible]	COBOL v5 / v6	4	58,309,842	45,643.3	424
[illegible]	COBOL v5 / v6	3	58,309,839	38,305.3	304
[illegible]	COBOL v4	656	46,806,938	2,020.8	108.3
[illegible]	COBOL v5 / v6	3	44,649,694	1.3	0
[illegible]	COBOL v5 / v6	11	44,277,277	44.2	13.4
[illegible]	COBOL v5 / v6	5	42,111,352	0	0
[illegible]	COBOL v5 / v6	7	34,267,580	5.6	2.6
[illegible]	COBOL v5 / v6	20	24,147,791	48.9	15.4
[illegible]	COBOL v5 / v6	5	20,749,370	82.8	23.2
[illegible]	COBOL v5 / v6	663	15,743,782	2,118.9	102.4
[illegible]	COBOL v5 / v6	663	15,743,782	14,122.2	404.6
[illegible]	COBOL v5 / v6	662	15,743,775	4,103.1	120.3

Rows 1-21 of first 2000

Module invocations by Language

language	modules	Loads	calls	Avg Elapsed (usecs)	Avg CPU (usecs)
COBOL v5 / v6	4,137	78,672	7,466,531,395	702,304.6	103,996.2
COBOL v4	33	773	46,853,848	5,384,515.3	333,252.9
Assembler	49	260	25,654	14,214.1	2,062.5
C/C++	58	13,544	12,580	123,142,240.8	77,251
PL/I	15	257	489	161,899,700.2	73,344.6

COBOL invocations by Compiler Version

compilerVersion	modules	calls
060200	1,816	3,554,416,304
060100	2,327	3,912,115,091
040200	23	44,431
010201	9	46,809,414
010200	1	3

Module invocation by year of compilation

COBOL invocations by ArchLevel

archLvl	modules	calls
11	2,327	3,912,115,091
12	1,810	3,554,416,262
NA	33	46,853,848
7	6	42

Profiler CPU Utilization (usecs)

COBOL invocations by Optimization Level

COBOL Version	Optimize	Modules	Calls
COBOL v5 / v6	2	4,130	7,466,531,352
COBOL v4	N	25	46,853,066
COBOL v4	Y	8	782
COBOL v5 / v6	0	7	43

Powered by Metabase

Use Case – Which programs are being used?

Watson Walker

- Identify modules that have the highest load count.
- Identify modules consuming the most CPU time (average and total).
- Identify modules responsible for the elapsed time (average and total).
- Identify modules that are called the most often.
- Sort on any column.

Frank's Dashboard

Frank Kyne's Personal Collection

Top Programs by CPU Time (mics)

moduleName	Language	Loads	Calls	Avg Elapsed (usecs)	Avg CPU (usecs)	Total Elapsed (usecs)	Total CPU (usecs)
[illegible]	COBOL v5 / v6	18	4,290,642,435	13.5	2.8	61,200,526,547	8,074,390,394
[illegible]	COBOL v5 / v6	3	3	512,269,282	428,143,868.7	1,536,807,846	1,284,431,606
[illegible]	COBOL v5 / v6	1	1	619,696,671	422,923,248	619,696,671	422,923,248
[illegible]	COBOL v5 / v6	4	67,540	1,632,339.2	819,482	2,033,687,611	583,615,341
[illegible]	COBOL v5 / v6	3	3	112,621,855.7	98,277,432	337,865,567	294,832,296
[illegible]	COBOL v5 / v6	1	1	10,491,652,429	128,235,199	10,491,652,429	128,235,199
[illegible]	COBOL v5 / v6	1	1	202,506,238	127,717,974	202,506,238	127,717,974
[illegible]	COBOL v5 / v6	6	1,444,955,475	53.5	17.8	2,494,565,302	518,911,776
[illegible]	COBOL v5 / v6	2	2	90,947,828.5	81,785,474.5	181,895,657	163,570,949
[illegible]	COBOL v5 / v6	1	7,216,346	1,601	11	11,559,368,316	85,928,642
[illegible]	COBOL v5 / v6	4	4	50,949,689.3	45,567,962.3	203,798,757	182,271,849
[illegible]	COBOL v5 / v6	2	2	88,344,298	79,128,212	176,688,596	158,256,424
[illegible]	COBOL v5 / v6	2	2	114,667,432	70,395,701	229,334,864	140,791,402
[illegible]	COBOL v5 / v6	1	1	285,441,764	49,194,397	285,441,764	49,194,397
[illegible]	COBOL v4	3	3	499,167,423.3	72,708,269.7	1,497,502,270	218,124,809
[illegible]	COBOL v5 / v6	1	1	50,780,564	44,899,398	50,780,564	44,899,398
[illegible]	COBOL v5 / v6	1	152,548	3,959	289	603,965,122	44,162,606
[illegible]	COBOL v5 / v6	1	1	145,899,402	40,816,966	145,899,402	40,816,966
[illegible]	COBOL v5 / v6	1	152,548	278	246	42,510,181	37,671,889

Powered by Metabase Rows 1-19 of first 2000

Use Case – Deep dive into active COBOL modules

- Identify COBOL modules that have the highest load count (prior slide).
- Identify COBOL modules consuming the most CPU time (prior slide).
- Identify COBOL modules by compiler version.
- Identify COBOL modules by optimization level.
- Identify COBOL modules by ARCHLVL.

COBOL invocations by Compiler Version

compilerVersion	modules	calls
060200	1,816	3,554,416,304
060100	2,327	3,912,115,091
040200	23	44,431
010201	9	46,809,414
010200	1	3

Powered by Metabase

COBOL Loads by compiled optimization level

optimizeOption	loadCount
N	83,707
2	10,561
0	4,362

Powered by Metabase

COBOL Invocations by ARCHLVL

Frank Kyne's Personal Collection • WW AP4Z [redacted] D210715

This question is written in SQL.

archLvl	Modules	Calls	Total_CPU_Time_(usecs)
11	2,327	3,912,115,091	7,392,800,917
12	1,810	3,554,416,262	10,537,021,766
NA	33	46,853,848	266,954,271
7	6	42	4,134

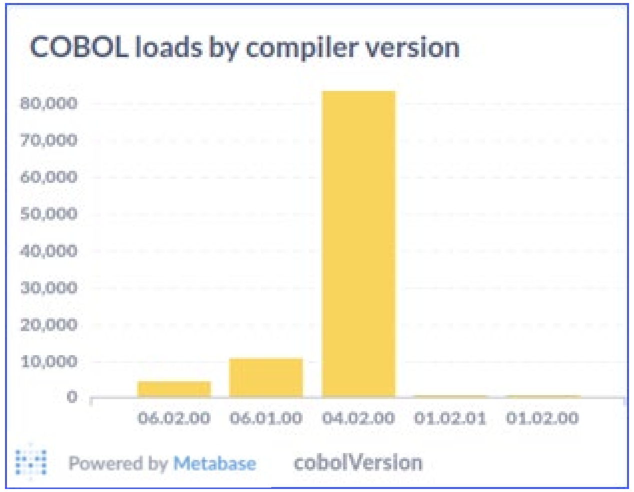
Powered by Metabase

Use Case – Deep dive into active COBOL modules

COBOL loads by year of compilation

Year	loadCount
2021	5,356
2020	600
2019	578
2018	6,446
2017	4,515
2016	517
2015	35,822
2014	44,641
2009	1
2000	151
1998	3

Powered by Metabase



Really old COBOL modules

Powered by Metabase

moduleName	cobolVersion	compilationYear	loadCount
[illegible]	01.02.00	1998	2
[illegible]	01.02.00	1998	1

- Identify COBOL modules by year of compilation.
- Display in tabular or graph format.
- Identify really, really old COBOL modules.

Use Case – Change control options

Watson Walker

- Identify all changed modules during the last week.

Modules Sorted by Compile Date and Time

Module Name	Ppa2time Stamp	Compiler Version	Arch Lvl	Load Count	Call Count
[illegible]	202107151844420	060200	12	1	48,564,250
[illegible]	202107151844420	060200	12	1	49,752,656
[illegible]	202107151842420	060200	12	1	1
[illegible]	202107151838190	060200	12	1	1
[illegible]	202107151824420	060200	12	1	1
[illegible]	202107151818420	060200	12	1	1
[illegible]	202107151805590	060200	12	1	1
[illegible]	202107151751320	060200	12	1	1
[illegible]	202107151724220	060200	12	1	1
[illegible]	202107151719210	060200	12	1	1

Powered by Metabase

Rows 1-10 of first 2000

- We are after a specific module..

CPU Time and Invocation Count for Selected Program

Powered by Metabase

Profiler RunNo.	No. of Invocations	Avg CPU Time
30,319	404,510,976	1.53
39,079	389,048,136	1.52
46,245	46,402,615	5.65
49,831	48,564,250	5.53
49,864	37,900,459	5.56
50,039	49,752,656	5.84

- This module changed its behavior, when and why did it change?

Use Case – Interesting statistics

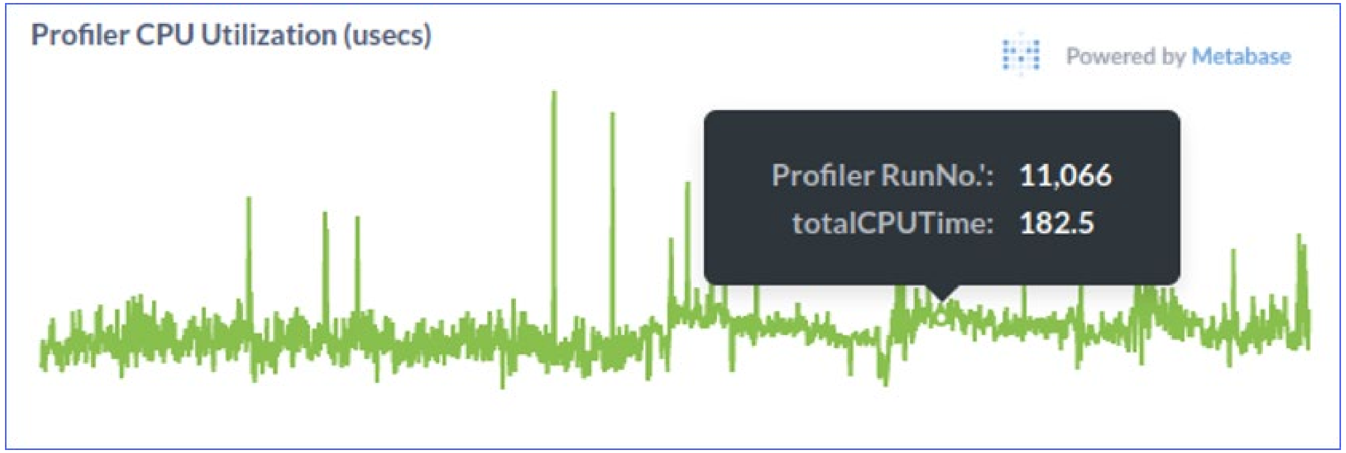
Watson Walker

- Identify module loads by language.

Module loads by Language — Powered by Metabase

language	loadCount
COBOL for OS/390 & VM, COBOL for MVS & VM	83,707
COBOL v5 and v6	14,923
OS/390 C/C++, C VM/ESA, XL C/C++	12,603
VisualAge PL/I for OS/390	334
ASSEMBLER	75

- Quantify overhead for AP4Z.



Other use cases

Watson Walker

Understand relationships between different programs and applications.

Detect unexpected or non-compliant relationships.

Generate inventories of programs related to applications subject to transformation projects.

Identify sub-optimal compile options or programs running with less efficient runtime options.

Prepare, support, and track the migration to new compiler version and measure the benefits or lack of benefits.

Increase accuracy of Quality Assurance test by determining in advance what should be tested after a change.

Help speed up root cause analysis on performance slow-downs.

Detect handled conditions which may impact program efficiency.

Data we collect (same slide as before)

Watson Walker

Job Step Level	
CPC model	z/OS level
Sysplex name	System name
Userid	Job name
Job-id	Job start date-time
Step name	Step program name
Step start date-time	# created enclaves
# created threads	#/type memory req
# handled conditions	

Module Level	
Module name	Load count
Call count	Elapsed time
CPU times	Amode
Routine type[1]	Compiler version
Compile date	Code page option
All compile options	
[1] LE conforming, Fastlink, IEEE floating point, XPLINK, DLL	

Summary of Application Profiling Capabilities

Watson Walker

Capability	AP4Z	Execution Samplers	Load Module Analysers	Software Asset Management Tools	SMF Step Data
Run a continuous dynamic data collector	Y	N	N	Y	Y
Provide a dynamic view of programs actually in use	Y	Y	N	Y	N
Provide module compiling information	Y	Y	Y	N	N
Only track application programs load requests	Y	N	N	N	N
Track the actual relationships between modules	Y	Y	N	N	N
Can create a call tree	Y	Y	N	N	N
Report execution times at the module level	Y	Y	N	N	N
Able to trace static calls	N	Y	N	N	N
Report execution time at the instruction level	N	Y	N	N	N
Report I/O at the step level	N	N	N	N	Y
Install system wide hooks, requires APF authorization	N	Y	N	Y	N
Has a noticeable impact on system wide performance	N	Y	N	Y	N
Provide a static view of modules in a load library	N	N	Y	Y	N
Track all module load requests	N	Y	N	Y	N

Good from an application profiling perspective

Not as good from an application profiling perspective

AP4Z Future Direction

- Support for CICS transactions.
- Additional canned reports.
- Ability to create call graphs.
- Support for IMS transactions.
- Java program profiling.
- Ability to run under zCX for reporting.

Is AP4Z right for you?

Watson Walker

We designed this to fill an important gap in the marketplace.

No other product provides as much insight into applications as AP4Z brings.

If you're under pressure for identifying and recompiling your critical COBOL V4 programs and dynamically-called programs, this is the very best tool to identify them.

The negligible CPU cost allows you to always keep it running for continuous tracking.

Many other IT departments, especially in operations and change control, will find this product invaluable.

The easy-to-use graphical interface to the offline database provides the ability to use the past to understand patterns, as well as to identify important trends.

For more information…

- Email to: ap4z@watsonwalker.com

- See our website at: www.watsonwalker.com/software/ap4z
  - This points to a recording of this presentation, a PDF of the presentation, and a *Tuning Letter* article about AP4Z.
  - You can also sign up to receive more information or a live demo.

- If you have questions or want to comment on this presentation, please contact us at ap4z@watsonwalker.com.

Thank you!