Java 5.0 Reliability, Availability and Serviceability

Java 5 Diagnostic Tools and Capabilities

Chris Bailey baileyc@uk.ibm.com
Trent Gray-Donald trent@ca.ibm.com

Agenda

- History
- 5.0 VM Problem Determination recap
- Strategic direction
- Tools dive
- Education / IBM Support Assistant
- Discussion / Questions

History

- Fragmented tooling story
  - Different tools for different folks, find tools in various places
    - In the JDK itself
    - alphaWorks
    - developerWorks
    - From Java|WAS support
  - Tools JVM level specific
- Substantial technology changes in underlying JVM implementation between 1.4.2 and 5.0
  - Significant robustness improvements (better compaction / fragmentation support, enhanced FFDC)
  - Fundamental PD data produced in same format

Java Problem Determination Strategy

- Centralization of Tools
  - Central customer visible repository of supported, maintained tools
  - Extensible, open tools, with programmable extension interfaces
- Tools must be usable everywhere
  - GUI mode for interactive use
  - Report generation for headless environments
- Documentation
  - Improvements to problem determination doc
  - Aggregated search in IBM Support Assistant
- Iterate!
  - Tools being deployed very regularly – looking for customer feedback.

Recap of JDK 5.0 Problem Determination facilities

- -Xtrace
  - Always on FFDC trace for major components
  - Separate buffers for GC – allows ‘flight recorder’
  - Method parameters available (if enabled)
- -Xdump
  - New triggers (thread start/stop, GC, heap expand, etc..)
  - More naming configuration allowed (date/time/etc..)
- java.lang.Management
  - Java level APIs to introspect into running system

Java Trace Engine

- Thousands of tracepoints throughout VM native code

- Can capture Java Methods data also

- Many different operation modes to aid debugging and problem diagnosis

- Limited “Flight Recorder” trace set is always on in Java 5.0
  - Key vm tracepoints constantly traced into per thread wrapping memory buffers
  - GCLogger tracepoints are stored in separate buffer to ensure they are not overwritten by the high frequency tracepoints

Controlling Trace

- The Trace Engine can be controlled through a number of mechanisms:
  - Through the -Xtrace command-line option
  - Using a trace properties file
  - Dynamically using Java code through the com.ibm.jvm.Trace API
  - Using trace trigger events
  - From an external agent using the C-based JVM RAS Interface (JVMRI)
- Primary way is via the -Xtrace option on the command line.

Basics of Activating Trace

- The first thing that you need to determine is the destination to which the trace output should be directed
  - `minimal` Trace identifier and timestamp only to in-core buffer.
  - `maximal` Trace identifier, timestamp and associated data to in-core buffer.
  - `count` Report the number of times a selected tracepoint is called
  - `print` Trace selected tracepoints to stderr with no indentation.
  - `iprint` Trace selected tracepoints to stderr with indentation.
  - `external` Route selected tracepoints to a JVMRI listener.
  - `exception` Trace selected tracepoints to an in-core buffer reserved for exceptions.

- The value of each keyword is then set to the trace points required
  - `-Xtrace:maximal=all` traces all of the information available from all JVM trace points to internal wrapping buffers.
  - `-Xtrace:iprint=awt` traces all of the JVM internal AWT trace points to stderr, with Indentations on entry and exit.
  - `-Xtrace:iprint=mt` activates method trace and set the output to stderr with indentations

Putting Trace Into Files

- Data can be written to file as an extension to in-storage trace

- To specify that the output of `minimal` or `maximal` trace options should be written to a file, the `output` keyword should be used
  - `-Xtrace:maximal=all,output=trace.out`
    traces into a file called trace.out.
  - `-Xtrace:maximal=all,output={trace.out,5m}`
    traces into a file called trace.out and wraps within the file once it has reached 5MB in size.
  - `-Xtrace:maximal=all,output={trace#.out,5m,5}`
    traces sequentially into five files, each 5MB in size, with # substituted for the file iteration number.

- It is also possible to put the following substitutions into the file name:
  - `%p:` The ID for the Java process.
  - `%d:` The current date, in yyyymmdd format.
  - `%t:` The current time, in hhmmss format.

Method Trace

- Instrumentation free tracing of:
  - Any Java Methods:
    - Core Java API, Middleware, 3rd party code, Customer code
  - Detailed information:
    - Entry and Exit points, with thread information and microsecond time stamps

- Two part invocation:
  - 1) Add `methods` keyword as a token to –Xtrace
  - 2) Use of `mt` as value to destination keyword (eg. maximal, print)

- Method selection by class name or method name
  - use of wildcards, along with the not operator, !, allowing for complex selection criteria.
    - `-Xtrace:print=mt,methods={*.*,!java/lang/*.*}`
      Write method trace to stderr for all methods and for all classes except those in the java.lang package.
    - `-Xtrace:maximal=mt,output=trace.out,methods={tests/mytest/*.*}`
      Write method trace to file for all methods in the tests.mytest package. (Note that this option selects only the methods that are to be traced.)

Triggering

- Trace can produce significant amounts of data
- The trace engine provides the ability to trigger on trace events
  - Provides ability to create targeted trace output
    - Reduces volume of data and performance impact
  - Provides ability to also generate dumps on trace points
- Following actions available as value of `trigger` keyword
  - `suspend` Suspend *all* tracing
  - `resume` Resume *all* tracing (except for threads suspended by resumecount property and Trace.suspendThis() calls).
  - `suspendthis` Increment the suspend count for this thread. A non-zero suspend count prevents tracing for the thread.
  - `resumethis` Decrement the suspend count for this thread if it is greater than zero. If the suspend count reaches zero, tracing for this thread will be resumed.
  - `sysdump` Produce a non-destructive system dump.
  - `javadump` Produce a Java dump.
  - `heapdump` Produce a heap dump.
  - `snap` Snap all active trace buffers to a file in the current working directory.

Triggering Examples

- Following format used to specify triggers in method events:
- `trigger=method{method spec,entry action,exit action,delay count,match count}`
  - On entering any method that matches the method spec indicated, the entry action is executed.
  - When exiting the method, the exit action is performed.
  - If the delay count is specified, the entry and exit actions are only carried out when entry and exit have occurred more times than the delay count.
  - If the match count is specified, the actions are only carried out a maximum of that many times.
- Examples:
  - `-Xtrace:trigger=method{java/lang/StackOverflowError*, sysdump}`
    create a system dump on the first (and only the first) instance of a StackOverflowError method being called - which is the <clinit> method.
  - `-Xtrace:resumecount=1`
  - `-Xtrace:trigger=method{HelloWorld.main,resume,suspend}`
    trace all threads once HelloWorld.main() is called and stop tracing when HelloWorld.main() returns.

Triggering and Method Trace in Action

-Xtrace:print=mt,methods={myapp/MyTime*},resumecount=1,trigger=method{myapp/MyTime.main,resume,suspend}

```
21:05:47.992*0x806cb00      mt.3        > myapp/MyTime.main([Ljava/lang/String;)V Bytecode static method
21:05:47.994 0x806cb00      mt.19        - Static method arguments: ([L@55D8CB98)
21:05:47.994 0x806cb00      mt.0         > myapp/MyTime.<init>()V Bytecode method, This = 809baec
21:05:47.994 0x806cb00      mt.18         - Instance method receiver: myapp/MyTime@55D8CBA8 arguments: ()
21:05:47.994 0x806cb00      mt.6         < myapp/MyTime.<init>()V Bytecode method
21:05:47.994 0x806cb00      mt.0         > myapp/MyTime.test()V Bytecode method, This = 809baf0
21:05:47.994 0x806cb00      mt.18         - Instance method receiver: myapp/MyTime@55D8CBA8 arguments: ()
21:05:48.079 0x806cb00      mt.6         < myapp/MyTime.test()V Bytecode method
21:05:48.079 0x806cb00      mt.9        < myapp/MyTime.main([Ljava/lang/String;)V Bytecode static method
```

Only real time (79ms) is in the call to MyTime.test()

Could now drill down into MyTime.test():
extend scope of methods traced, and reduce scope of tracing into MyTime.test()

-Xtrace:print=mt,methods={myapp/*},resumecount=1,trigger=method{myapp/MyTime.test,resume,suspend}

```
21:07:14.968*0x806cb00      mt.0        > myapp/MyTime.test()V Bytecode method, This = 809baf0
21:07:14.970 0x806cb00      mt.18        - Instance method receiver: myapp/MyTime@55D8CBA8 arguments: ()
21:07:15.067 0x806cb00      mt.3         > myapp/MyTimer.getTime()V Bytecode static method
21:07:15.067 0x806cb00      mt.19         - Static method arguments: ()
21:07:15.067 0x806cb00      mt.9         < myapp/MyTimer.getTime()V Bytecode static method
21:07:15.069 0x806cb00      mt.6        < myapp/MyTime.test()V Bytecode method
```

Java Dump Engine

- 5/6 available dump types:

console	Basic thread dump to stderr
system	Capture raw process image
tool	Run command line program
java	Write application summary
heap	Capture raw heap image
snap	Take a snap of the trace buffers

- Large range of dump triggers
  - 14 available triggers
  - Dump events extended by the use of filters
- User defined dump labels
  - Ability to include: time, date, pid, uid, jre info
- Ability to configure number of dumps generated
- Ability to execute tool on dump event

Trigger Events

gpf	an unexpected crash, such as a SIGSEGV or a SIGILL
user	SIGQUIT signal (Ctrl-Brk on Windows, Ctrl-\ on Linux, Ctrl-V on z/OS)
abort	a controlled crash, as triggered by the abort() system call
vmstart	the VM has finished initialization
vmstop	the VM is about to shutdown
load	a new class has been loaded
unload	a classloader has been unloaded
throw	a Java exception has been thrown
catch	a Java exception has been caught
uncaught	a Java exception was not handled by the application
thrstart	a new thread has started
thrstop	an old thread has stopped
blocked	a thread is blocked entering a monitor
fullgc	garbage collection has started

Dump filters / ranges

- class events (load, throw, catch, uncaught)
  - exact name `filter=java/lang/OutOfMemoryError`
  - prefix `filter=java/lang/Out*`
  - substring `filter=*OutOfMemory*`
- vmstop event
  - exit code(s) `filter=#129..192#-42#255`
- ranges can be used to remove “noise” or save disk space
  - bounded `range=1..4`
  - open-ended `range=8..0`

Dump Labels

- Can use any combination of tokens and text
- Tokens expanded at time of event
  - Usual date and time: %Y, %y, %m, %d, %H, %M, %S
  - High precision time: %tick (msec), %seq (dump counter)
  - Process info: %pid, %uid (plus %job on z/OS)
  - JRE info: %home, %last (last snapped dump label)
- VM will try to create intermediate directories, for example:

/mnt/archive/dumps/%Y%m%d/%pid/javacore.%tick.txt

Dump Tools

- Spawns external command
  - -Xdump:tool:exec=<command string>
- Command string can contain tokens
  - “%home\bin\jextract core.%Y%m%d.%H%M%S.%pid.dmp”
- Default tool attaches platform debugger to VM

– Windows:	windbg
– Linux:	gdb
– AIX, z/OS:	dbx

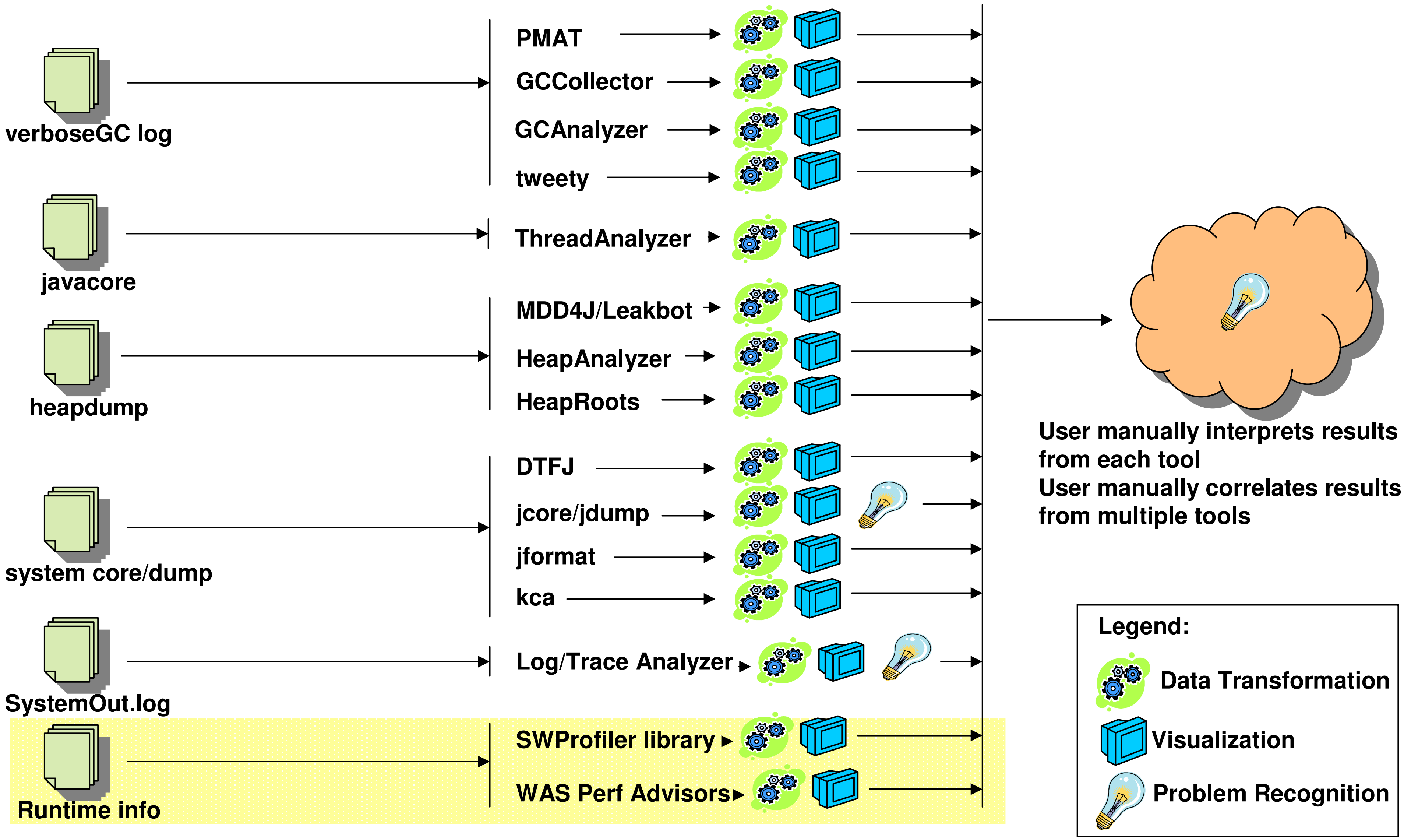
Strategic Direction - DTFJ

DTFJ (Diagnostic Tooling Framework for Java) is a new technology within the IBM JDK to analyze and diagnose problems in Java applications

  - Read RAS artifacts from a JVM (e.g. a core file) and extract all kinds of useful information from that dump
  - Not just one tool: an extensible framework for building many different tools

- Components of the DTFJ family
  - **jextract**: a tool to capture information from a JVM system dump (e.g. core file) and package it into a platform-independent format
  - **DTFJ library proper or core library**: a library that parses the contents of the system dump file packaged by jextract, and provides access to its contents in a standardized manner, through a standard API
  - **DTFJ-based tools**: a collection of end-user tools that call the DTFJ library through the DTFJ API, to present and analyze information in various ways useful to the users

Roadmap: Where we are coming from



verboseGC log

PMAT

GCCollector

GCAnalyzer

tweety

javacore

ThreadAnalyzer

heapdump

MDD4J/Leakbot

HeapAnalyzer

HeapRoots

system core/dump

DTFJ

jcore/jdump

jformat

kca

SystemOut.log

Log/Trace Analyzer

Runtime info

SWProfiler library

WAS Perf Advisors

User manually interprets results from each tool
User manually correlates results from multiple tools

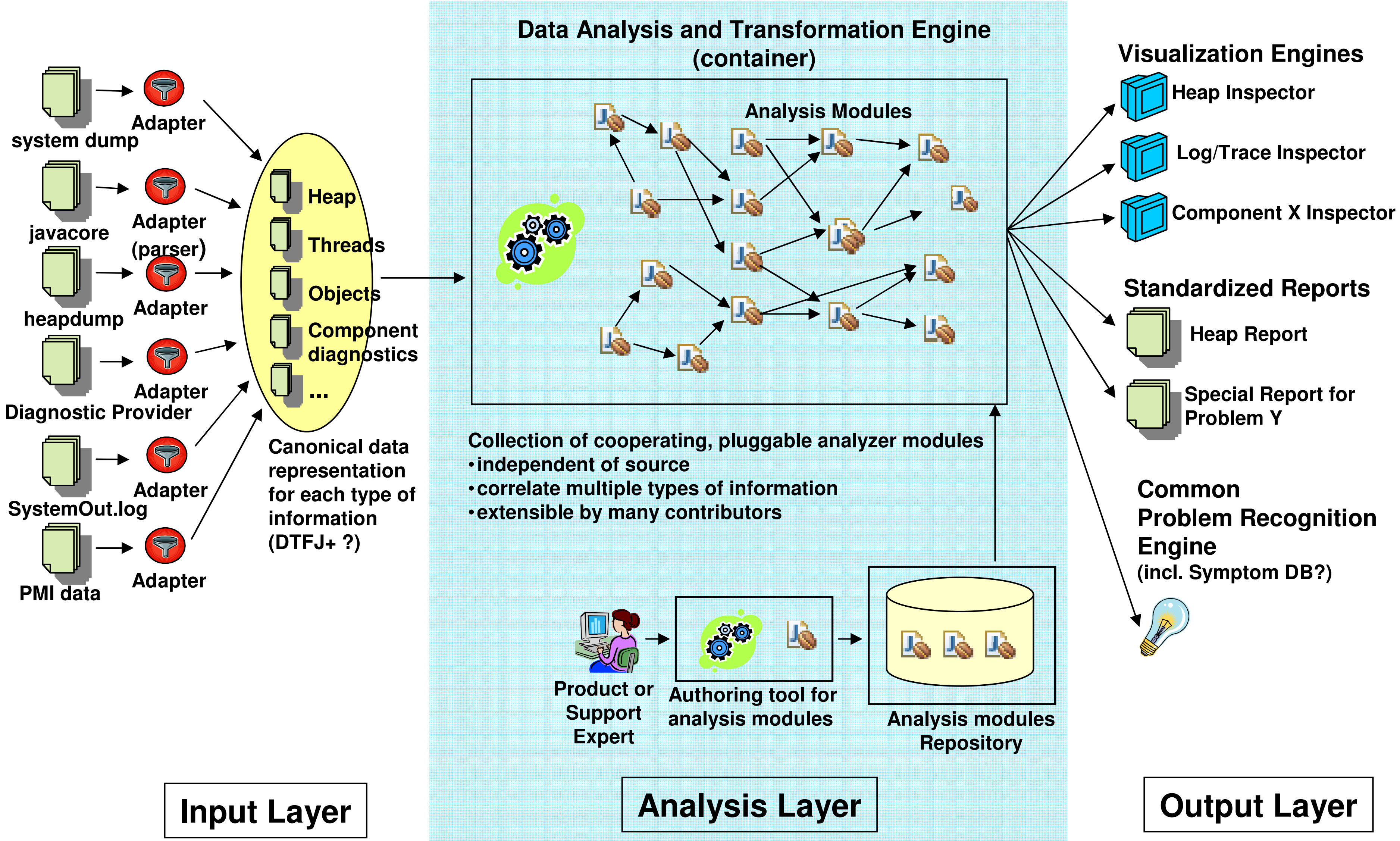
Legend:

Data Transformation

Visualization

Problem Recognition

Roadmap: Composable Tools Architecture – where we are going



system dump

Adapter

javacore

Adapter (parser)

heapdump

Adapter

Diagnostic Provider

Adapter

SystemOut.log

Adapter

PMI data

Adapter

Heap

Threads

Objects

Component diagnostics

...

Canonical data representation for each type of information (DTFJ+ ?)

Data Analysis and Transformation Engine (container)

Analysis Modules

Collection of cooperating, pluggable analyzer modules
- independent of source
- correlate multiple types of information
- extensible by many contributors

Product or Support Expert

Authoring tool for analysis modules

Analysis modules Repository

Visualization Engines

Heap Inspector

Log/Trace Inspector

Component X Inspector

Standardized Reports

Heap Report

Special Report for Problem Y

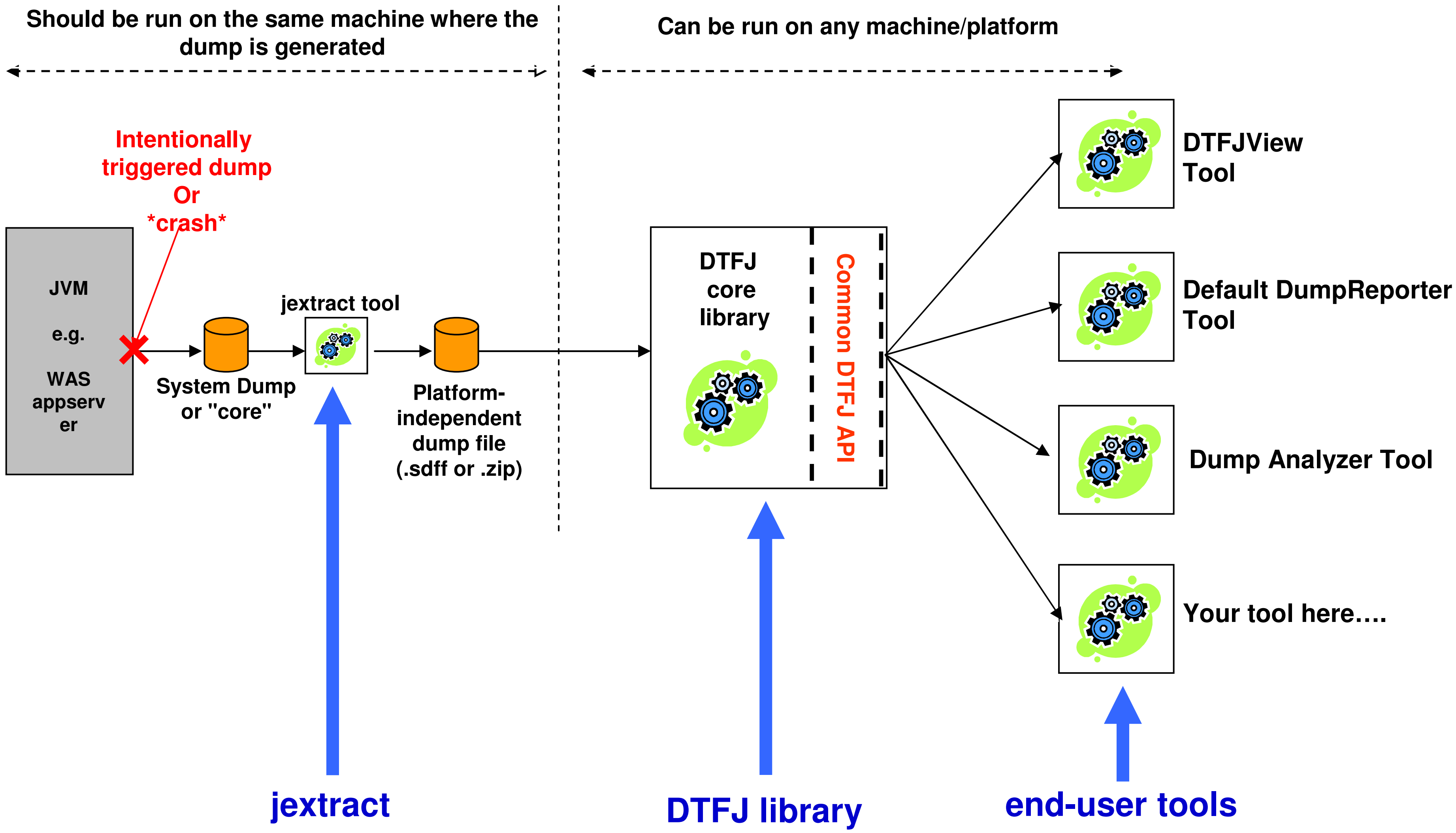
Common Problem Recognition Engine (incl. Symptom DB?)

Input Layer

Analysis Layer

Output Layer

Using the DTFJ components - example

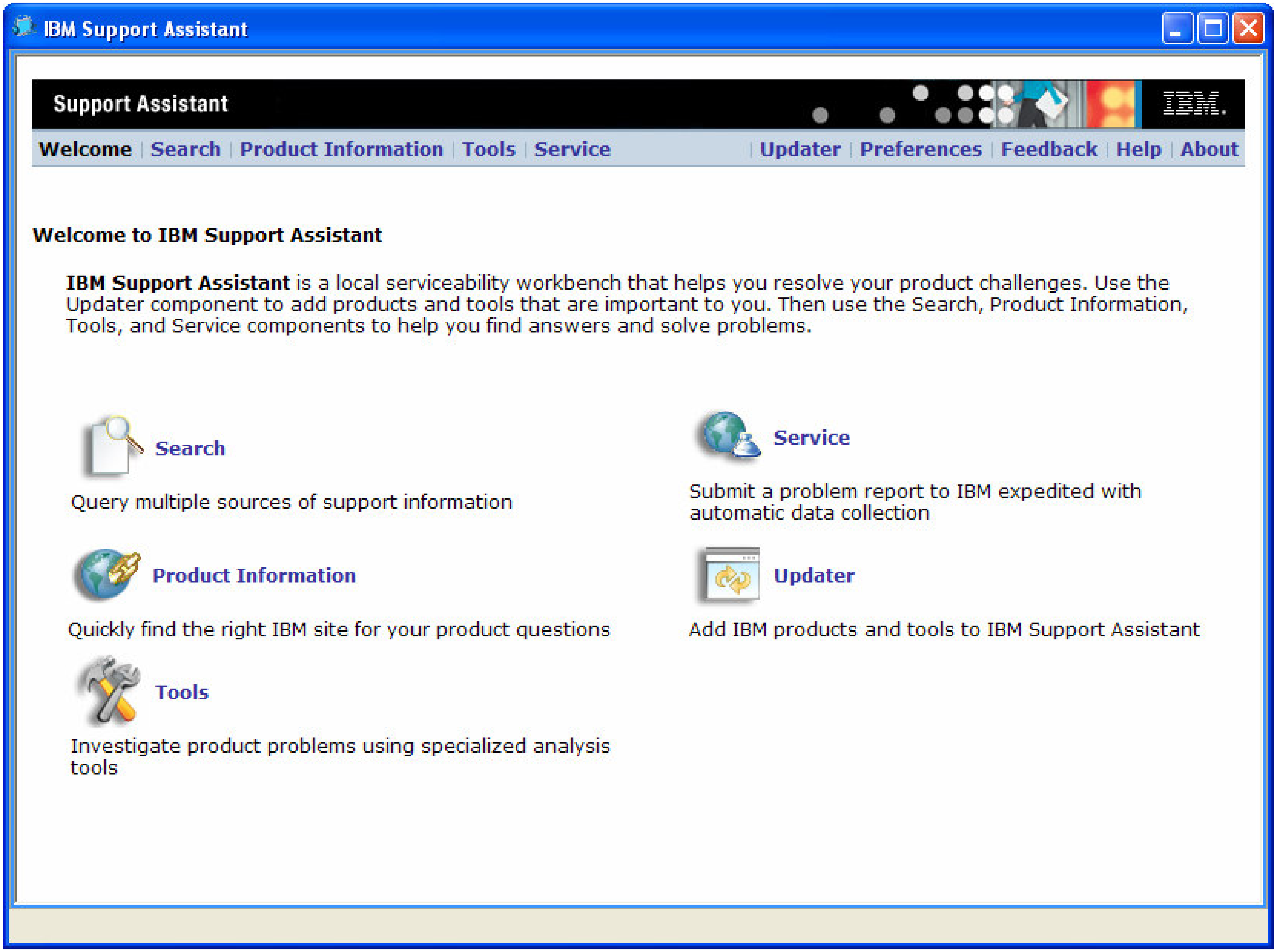

Should be run on the same machine where the dump is generated
Can be run on any machine/platform
Intentionally triggered dump Or *crash*
JVM
e.g.
WAS appserver
System Dump or "core"
jextract tool
Platform-independent dump file (.sdff or .zip)
DTFJ core library
Common DTFJ API
DTFJView Tool
Default DumpReporter Tool
Dump Analyzer Tool
Your tool here….
jextract
DTFJ library
end-user tools

Common Problem Recognition

- Goal: reduce service turnaround time by providing a comprehensive suite of self-help tools and documentation
  - Causal analysis of PMRs in both WAS and JDK to detect common idiom (across all platforms)
  - Documentation revisions and enhancements
    - IBM Education Assistant
    - IBM Guided Activity Assistant

IBM Support Assistant

- Hosting for Serviceability Tools across product families
- Automatic PD data gathering
- Assist with opening PMRs and working with IBM Support
- Documentation
  - Aggregated Search across sources
  - Regular updates to Diagnostics Guide


IBM Support Assistant
Support Assistant
IBM.
Welcome | Search | Product Information | Tools | Service | Updater | Preferences | Feedback | Help | About

Welcome to IBM Support Assistant

IBM Support Assistant is a local serviceability workbench that helps you resolve your product challenges. Use the Updater component to add products and tools that are important to you. Then use the Search, Product Information, Tools, and Service components to help you find answers and solve problems.

Search

Query multiple sources of support information

Service

Submit a problem report to IBM expedited with automatic data collection

Product Information

Quickly find the right IBM site for your product questions

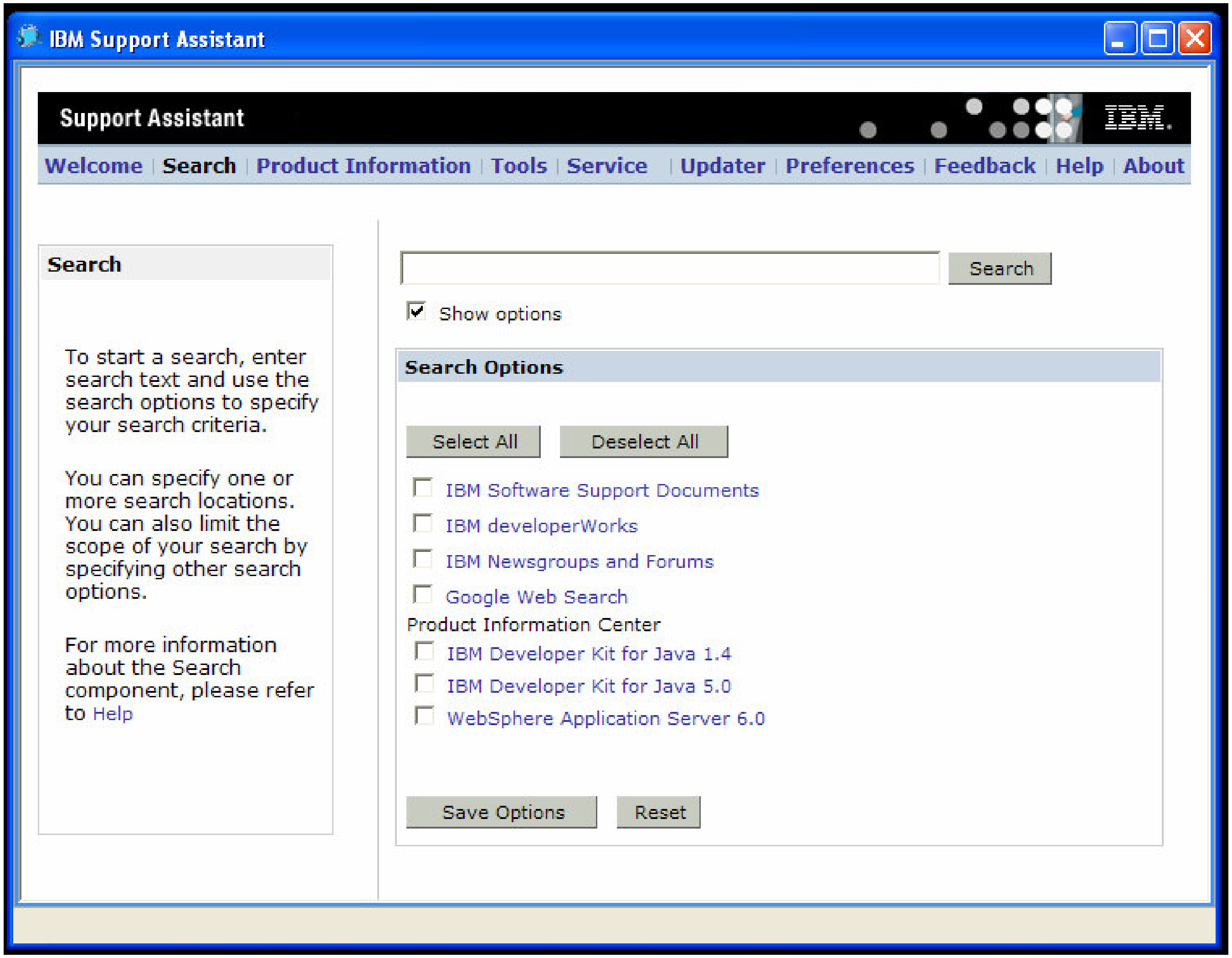
Updater

Add IBM products and tools to IBM Support Assistant

Tools

Investigate product problems using specialized analysis tools

ISA - Search



IBM Support Assistant
Support Assistant
IBM.
Welcome | Search | Product Information | Tools | Service | Updater | Preferences | Feedback | Help | About

Search

To start a search, enter search text and use the search options to specify your search criteria.

You can specify one or more search locations. You can also limit the scope of your search by specifying other search options.

For more information about the Search component, please refer to Help

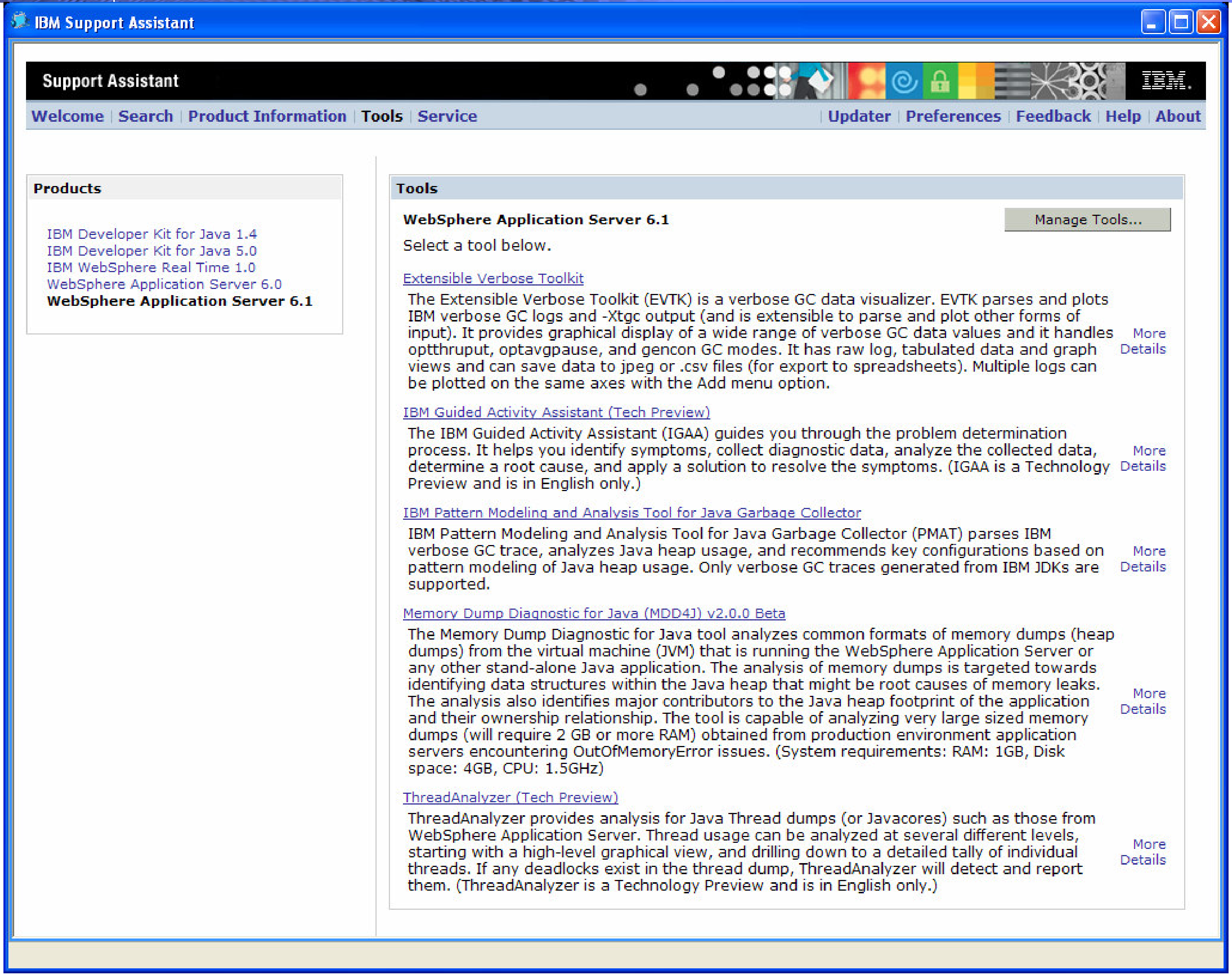
☑ Show options

Search Options

□ IBM Software Support Documents
□ IBM developerWorks
□ IBM Newsgroups and Forums
□ Google Web Search

Product Information Center

□ IBM Developer Kit for Java 1.4
□ IBM Developer Kit for Java 5.0
□ WebSphere Application Server 6.0


IBM Support Assistant
Support Assistant
IBM.
Welcome | Search | Product Information | Tools | Service
Updater | Preferences | Feedback | Help | About
Manage Tools...
More Details

Products

IBM Developer Kit for Java 1.4
IBM Developer Kit for Java 5.0
IBM WebSphere Real Time 1.0
WebSphere Application Server 6.0
WebSphere Application Server 6.1

Tools

WebSphere Application Server 6.1

Select a tool below.

Extensible Verbose Toolkit

The Extensible Verbose Toolkit (EVTK) is a verbose GC data visualizer. EVTK parses and plots IBM verbose GC logs and -Xtgc output (and is extensible to parse and plot other forms of input). It provides graphical display of a wide range of verbose GC data values and it handles optthruput, optavgpause, and gencon GC modes. It has raw log, tabulated data and graph views and can save data to jpeg or .csv files (for export to spreadsheets). Multiple logs can be plotted on the same axes with the Add menu option.

IBM Guided Activity Assistant (Tech Preview)

The IBM Guided Activity Assistant (IGAA) guides you through the problem determination process. It helps you identify symptoms, collect diagnostic data, analyze the collected data, determine a root cause, and apply a solution to resolve the symptoms. (IGAA is a Technology Preview and is in English only.)

IBM Pattern Modeling and Analysis Tool for Java Garbage Collector

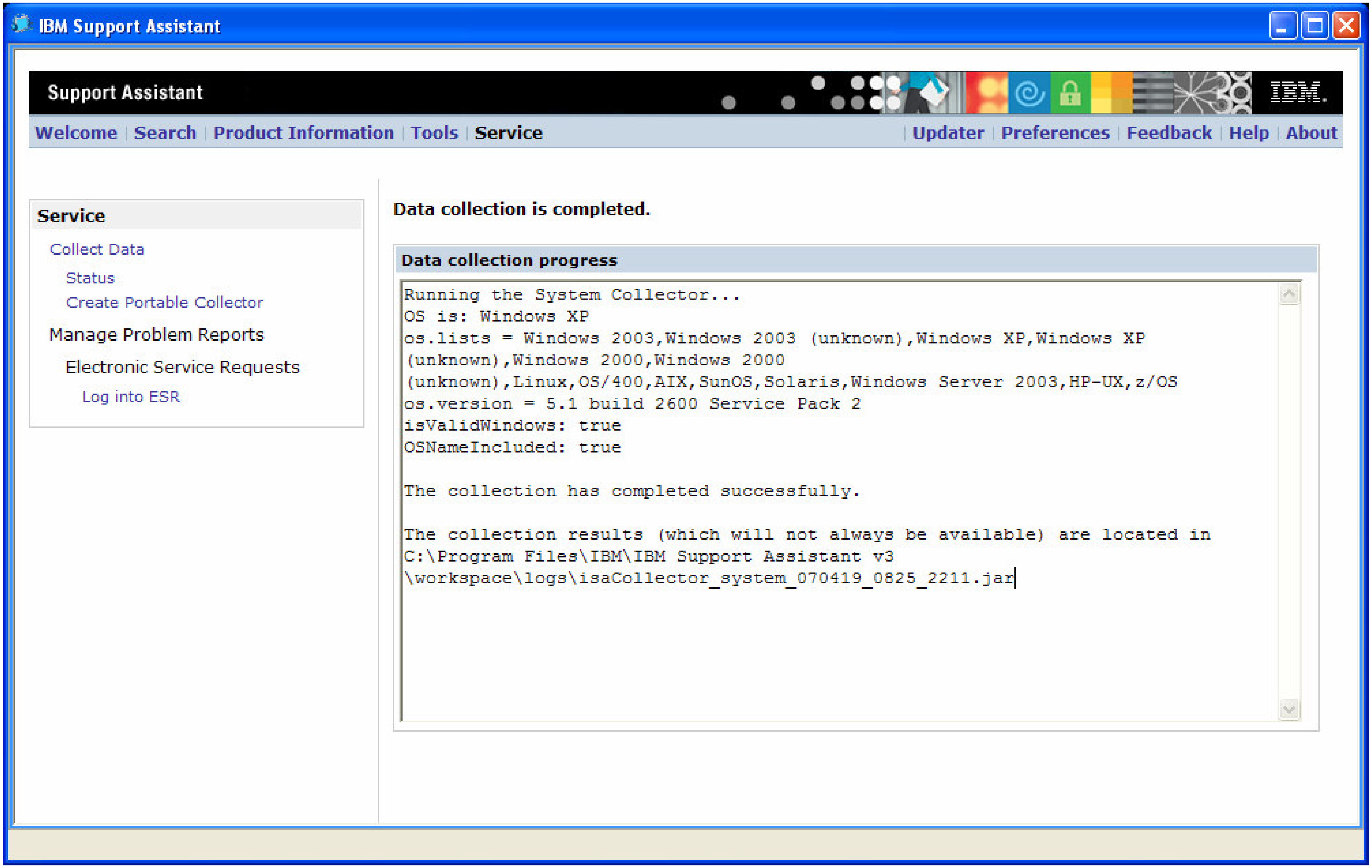
IBM Pattern Modeling and Analysis Tool for Java Garbage Collector (PMAT) parses IBM verbose GC trace, analyzes Java heap usage, and recommends key configurations based on pattern modeling of Java heap usage. Only verbose GC traces generated from IBM JDKs are supported.

Memory Dump Diagnostic for Java (MDD4J) v2.0.0 Beta

The Memory Dump Diagnostic for Java tool analyzes common formats of memory dumps (heap dumps) from the virtual machine (JVM) that is running the WebSphere Application Server or any other stand-alone Java application. The analysis of memory dumps is targeted towards identifying data structures within the Java heap that might be root causes of memory leaks. The analysis also identifies major contributors to the Java heap footprint of the application and their ownership relationship. The tool is capable of analyzing very large sized memory dumps (will require 2 GB or more RAM) obtained from production environment application servers encountering OutOfMemoryError issues. (System requirements: RAM: 1GB, Disk space: 4GB, CPU: 1.5GHz)

ThreadAnalyzer (Tech Preview)

ThreadAnalyzer provides analysis for Java Thread dumps (or Javacores) such as those from WebSphere Application Server. Thread usage can be analyzed at several different levels, starting with a high-level graphical view, and drilling down to a detailed tally of individual threads. If any deadlocks exist in the thread dump, ThreadAnalyzer will detect and report them. (ThreadAnalyzer is a Technology Preview and is in English only.)


IBM Support Assistant
Support Assistant
Welcome | Search | Product Information | Tools | Service
| Updater | Preferences | Feedback | Help | About
Service
Collect Data
Status
Create Portable Collector
Manage Problem Reports
Electronic Service Requests
Log into ESR

Data collection is completed.

Data collection progress

```
Running the System Collector...
OS is: Windows XP
os.lists = Windows 2003,Windows 2003 (unknown),Windows XP,Windows XP
(unknown),Windows 2000,Windows 2000
(unknown),Linux,OS/400,AIX,SunOS,Solaris,Windows Server 2003,HP-UX,z/OS
os.version = 5.1 build 2600 Service Pack 2
isValidWindows: true
OSNameIncluded: true

The collection has completed successfully.

The collection results (which will not always be available) are located in
C:\Program Files\IBM\IBM Support Assistant v3
\workspace\logs\isaCollector_system_070419_0825_2211.jar
```

Common Support Concerns – what we heard

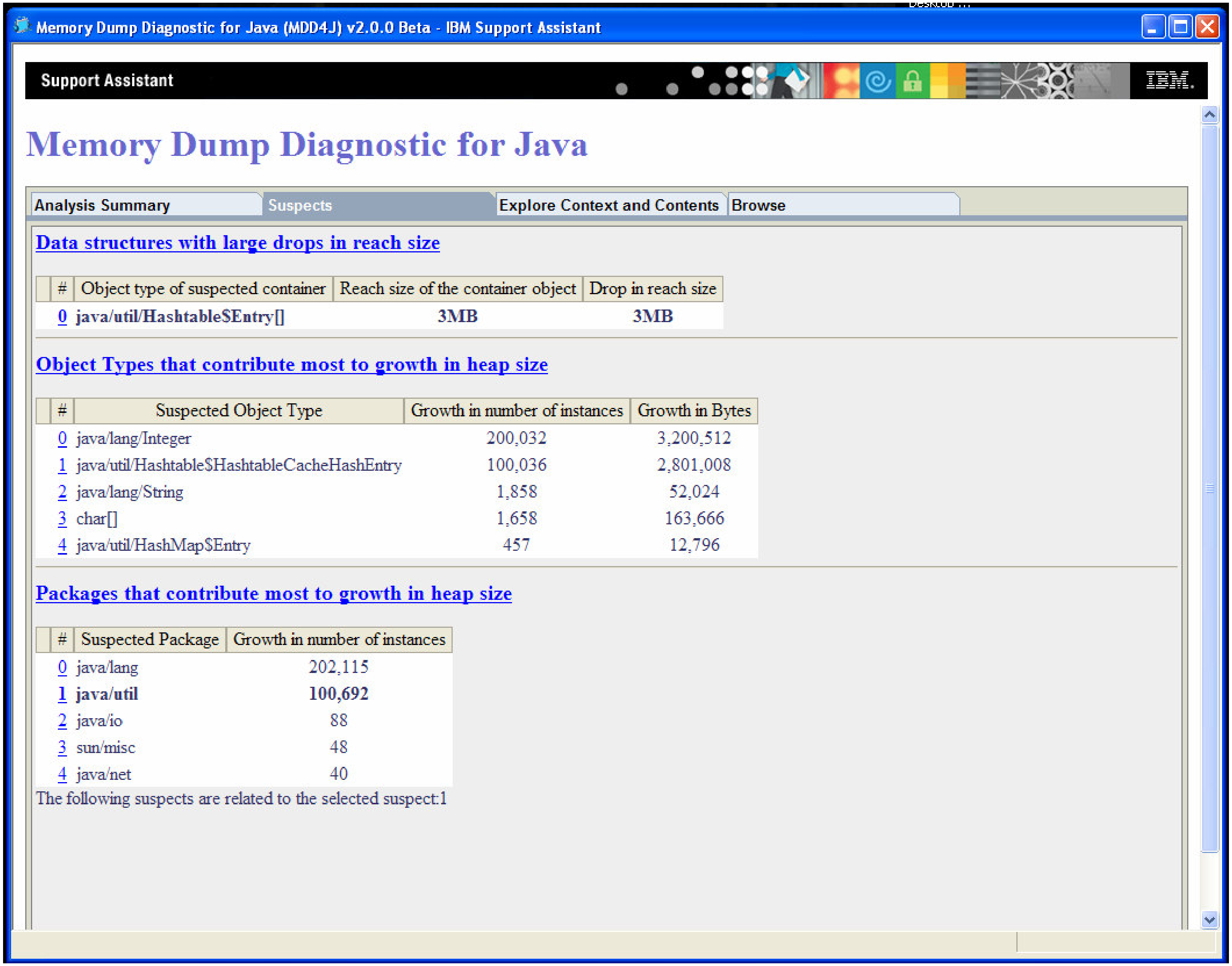
- OutOfMemoryError / heap Size Tuning
  - It's hard to tune the right GC parameters, and figure out where memory leaks come from.
- Deadlocks / hangs / spins
  - Need ability to introspect on a running JVM to determine what's happening at the moment – in a report based way.
- General analysis tools
  - Need ability to examine JVM data – classloaders, threads, monitors, etc.. to do general PD tasks.

Tools

- IBM Support Assistant “umbrella”
  - MDD4J (Memory Dump Diagnostics For Java)
  - EVTK (Extensible Verbose ToolKit)
  - DumpAnalyser
  - ThreadAnalyser
  - Java Lock Monitor (JLM)
  - IBM Guided Activity Assistant (IGAA)
- RAD (Rational Application Developer) & family

MDD4J

- Java Memory Analysis tool
  - Help explain / track down OutOfMemoryError
  - Performance problems when object use
- 2 modes of use
  - Single snapshot – to visualize a given heap
  - Delta mode – to track growth between 2 points in time
- Input data types supported
  - IBM Portable Heap Dump (heapdump.phd)
  - IBM Text heap dump (heapdump.txt)
  - HPROF heap dump format (hprof.txt)
  - 2007 workplan – DTFJ adaptors (thus direct svcdump consumption)
- Aims to replace HeapRoots, FindRoots, etc..
  - Compatible imputs with those tools
  - Full replacement only when true full functional superset


Memory Dump Diagnostic for Java (MDD4J) v2.0.0 Beta - IBM Support Assistant
Support Assistant
IBM.
Analysis Summary
Suspects
Explore Context and Contents
Browse

Memory Dump Diagnostic for Java

Data structures with large drops in reach size

#	Object type of suspected container	Reach size of the container object	Drop in reach size
0	**java/util/Hashtable$Entry[]**	**3MB**	**3MB**

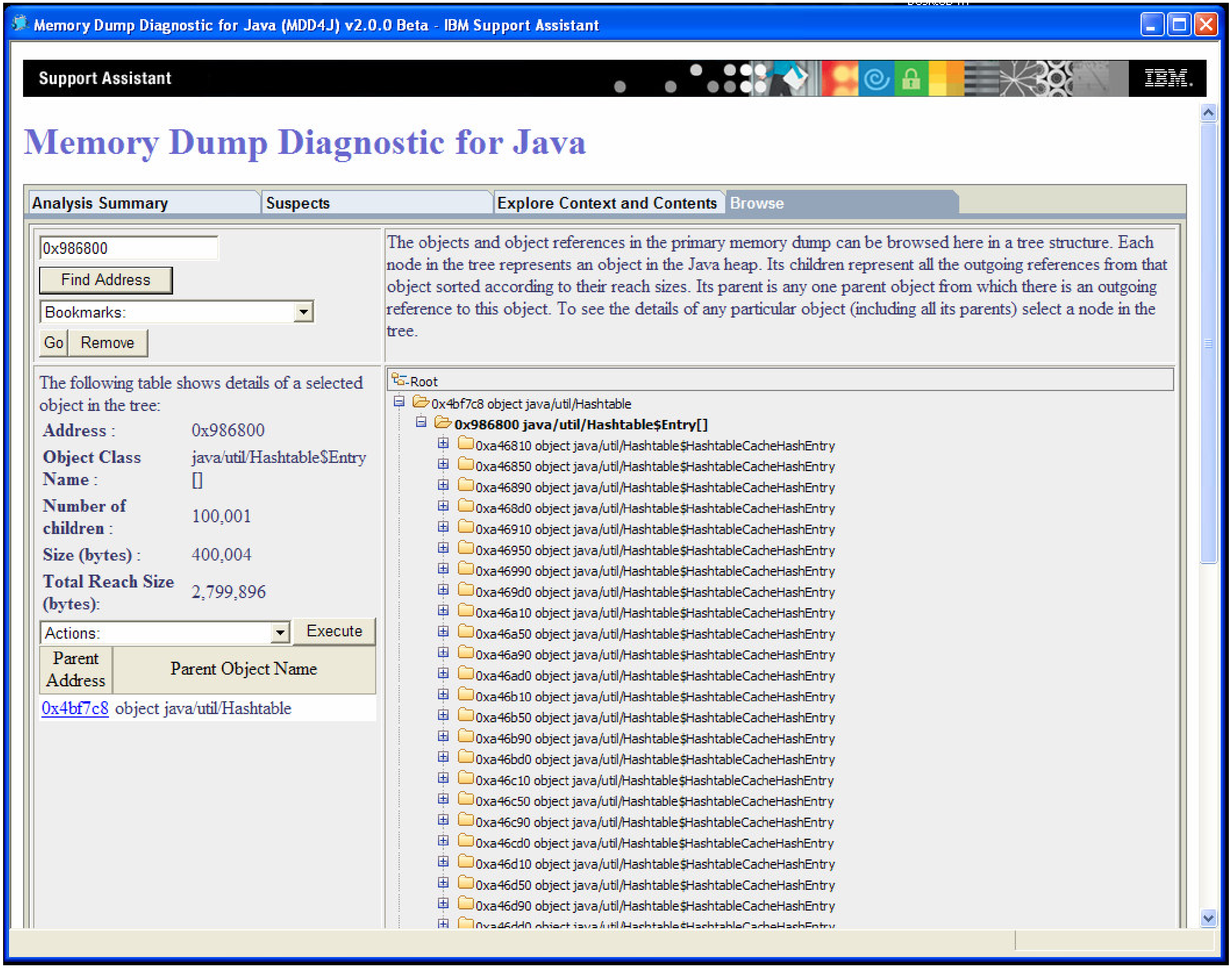
Object Types that contribute most to growth in heap size

#	Suspected Object Type	Growth in number of instances	Growth in Bytes
0	java/lang/Integer	200,032	3,200,512
1	java/util/Hashtable$HashtableCacheHashEntry	100,036	2,801,008
2	java/lang/String	1,858	52,024
3	char[]	1,658	163,666
4	java/util/HashMap$Entry	457	12,796

Packages that contribute most to growth in heap size

#	Suspected Package	Growth in number of instances
0	java/lang	202,115
1	**java/util**	**100,692**
2	java/io	88
3	sun/misc	48
4	java/net	40

The following suspects are related to the selected suspect:1



Memory Dump Diagnostic for Java (MDD4J) v2.0.0 Beta - IBM Support Assistant

Memory Dump Diagnostic for Java

0x986800

The objects and object references in the primary memory dump can be browsed here in a tree structure. Each node in the tree represents an object in the Java heap. Its children represent all the outgoing references from that object sorted according to their reach sizes. Its parent is any one parent object from which there is an outgoing reference to this object. To see the details of any particular object (including all its parents) select a node in the tree.

The following table shows details of a selected object in the tree:

Address :	0x986800
Object Class Name :	java/util/Hashtable$Entry []
Number of children :	100,001
Size (bytes) :	400,004
Total Reach Size (bytes):	2,799,896

Parent Address	Parent Object Name
0x4bf7c8	object java/util/Hashtable

```
Root
⊟ 0x4bf7c8 object java/util/Hashtable
  ⊟ 0x986800 java/util/Hashtable$Entry[]
    ⊞ 0xa46810 object java/util/Hashtable$HashtableCacheHashEntry
    ⊞ 0xa46850 object java/util/Hashtable$HashtableCacheHashEntry
    ⊞ 0xa46890 object java/util/Hashtable$HashtableCacheHashEntry
    ⊞ 0xa468d0 object java/util/Hashtable$HashtableCacheHashEntry
    ⊞ 0xa46910 object java/util/Hashtable$HashtableCacheHashEntry
    ⊞ 0xa46950 object java/util/Hashtable$HashtableCacheHashEntry
    ⊞ 0xa46990 object java/util/Hashtable$HashtableCacheHashEntry
    ⊞ 0xa469d0 object java/util/Hashtable$HashtableCacheHashEntry
    ⊞ 0xa46a10 object java/util/Hashtable$HashtableCacheHashEntry
    ⊞ 0xa46a50 object java/util/Hashtable$HashtableCacheHashEntry
    ⊞ 0xa46a90 object java/util/Hashtable$HashtableCacheHashEntry
    ⊞ 0xa46ad0 object java/util/Hashtable$HashtableCacheHashEntry
    ⊞ 0xa46b10 object java/util/Hashtable$HashtableCacheHashEntry
    ⊞ 0xa46b50 object java/util/Hashtable$HashtableCacheHashEntry
    ⊞ 0xa46b90 object java/util/Hashtable$HashtableCacheHashEntry
    ⊞ 0xa46bd0 object java/util/Hashtable$HashtableCacheHashEntry
    ⊞ 0xa46c10 object java/util/Hashtable$HashtableCacheHashEntry
    ⊞ 0xa46c50 object java/util/Hashtable$HashtableCacheHashEntry
    ⊞ 0xa46c90 object java/util/Hashtable$HashtableCacheHashEntry
    ⊞ 0xa46cd0 object java/util/Hashtable$HashtableCacheHashEntry
    ⊞ 0xa46d10 object java/util/Hashtable$HashtableCacheHashEntry
    ⊞ 0xa46d50 object java/util/Hashtable$HashtableCacheHashEntry
    ⊞ 0xa46d90 object java/util/Hashtable$HashtableCacheHashEntry
    ⊞ 0xa46dd0 object java/util/Hashtable$HashtableCacheHashEntry
```

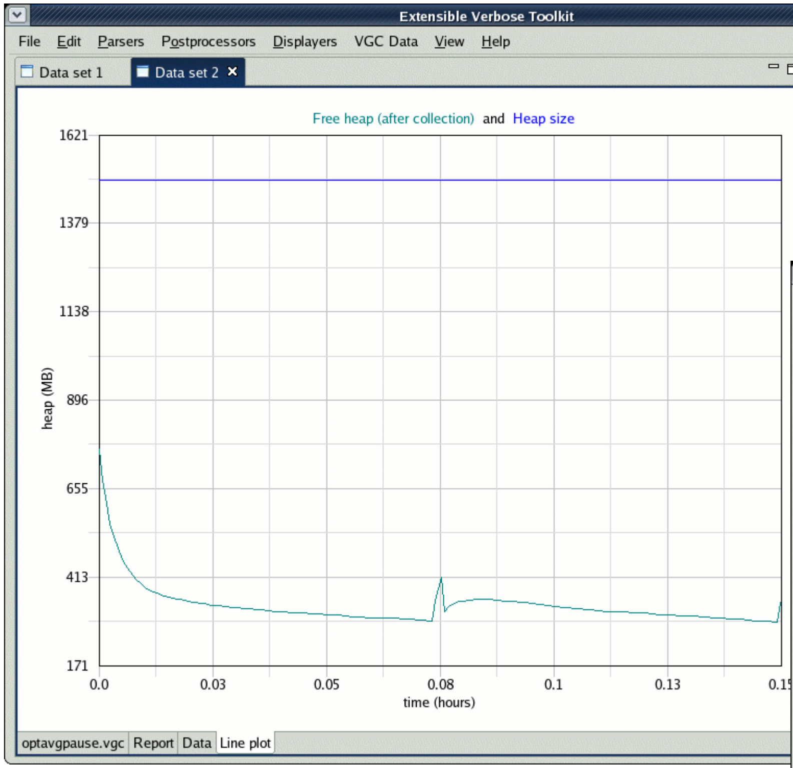
Extensible Verbose Toolkit (eVTK)

- EVTK is a verbose GC analysis tool
- Handles verbose GC from all versions of IBM JVMs
  - 1.4.2 and lower
  - 1.5.0 and higher
  - WebSphere real time
  - Intel, PowerPC, Z-Series,
- … and Solaris platforms

EVTK capabilities

- Analyses heap usage, heap size, pause times, and many other properties
- Compare multiple logs in the same plots and reports
- Many views on data
  - Reports
  - Graphs
  - Tables
- Can save data to
  - HTML reports
  - JPEG pictures
  - CSV files

EVTK – Heap Visualization

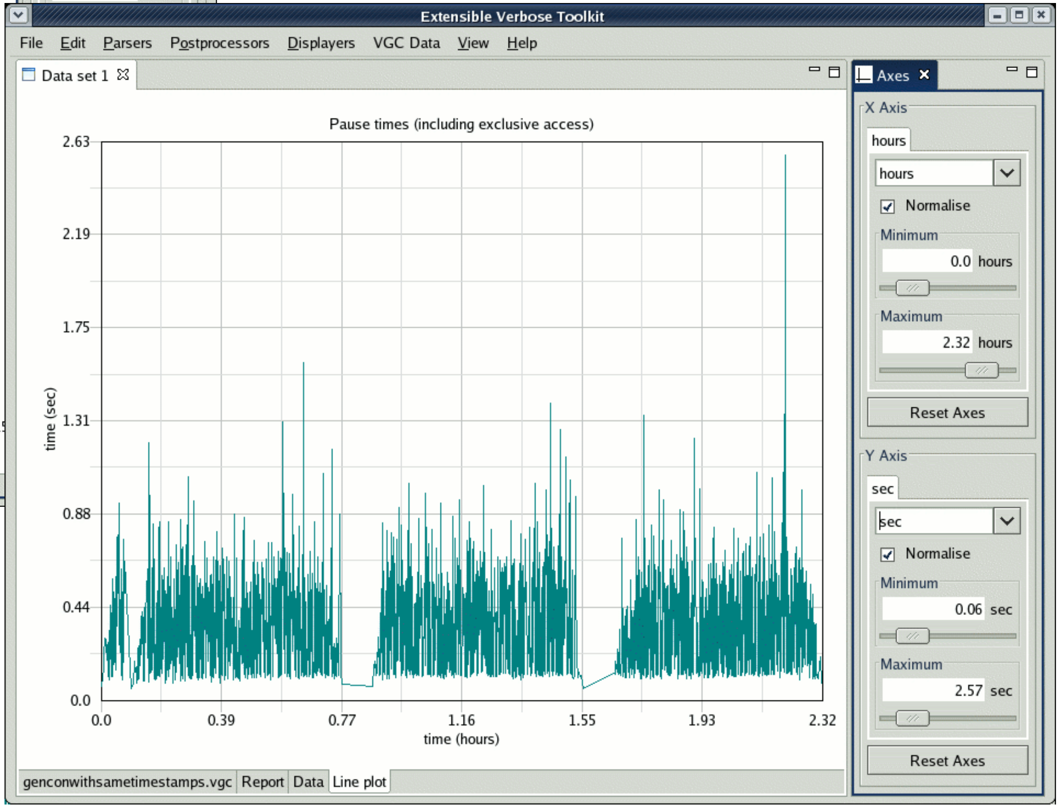


X Axis: hours

☑ Normalise

Minimum 0.0 hours

Heap occupancy


Extensible Verbose Toolkit
File Edit Parsers Postprocessors Displayers VGC Data View Help
Data set 1
Pause times (including exclusive access)
2.63
2.19
1.75
1.31
0.88
0.44
0.0
time (sec)
0.0
0.39
0.77
1.16
1.55
1.93
2.32
time (hours)
genconwithsametimestamps.vgc
Report
Data
Line plot
Axes
X Axis
hours
Reset Axes
Y Axis
sec
Reset Axes

X Axis: hours

☑ Normalise

Minimum 0.0 hours

Maximum 2.32 hours

Y Axis: sec

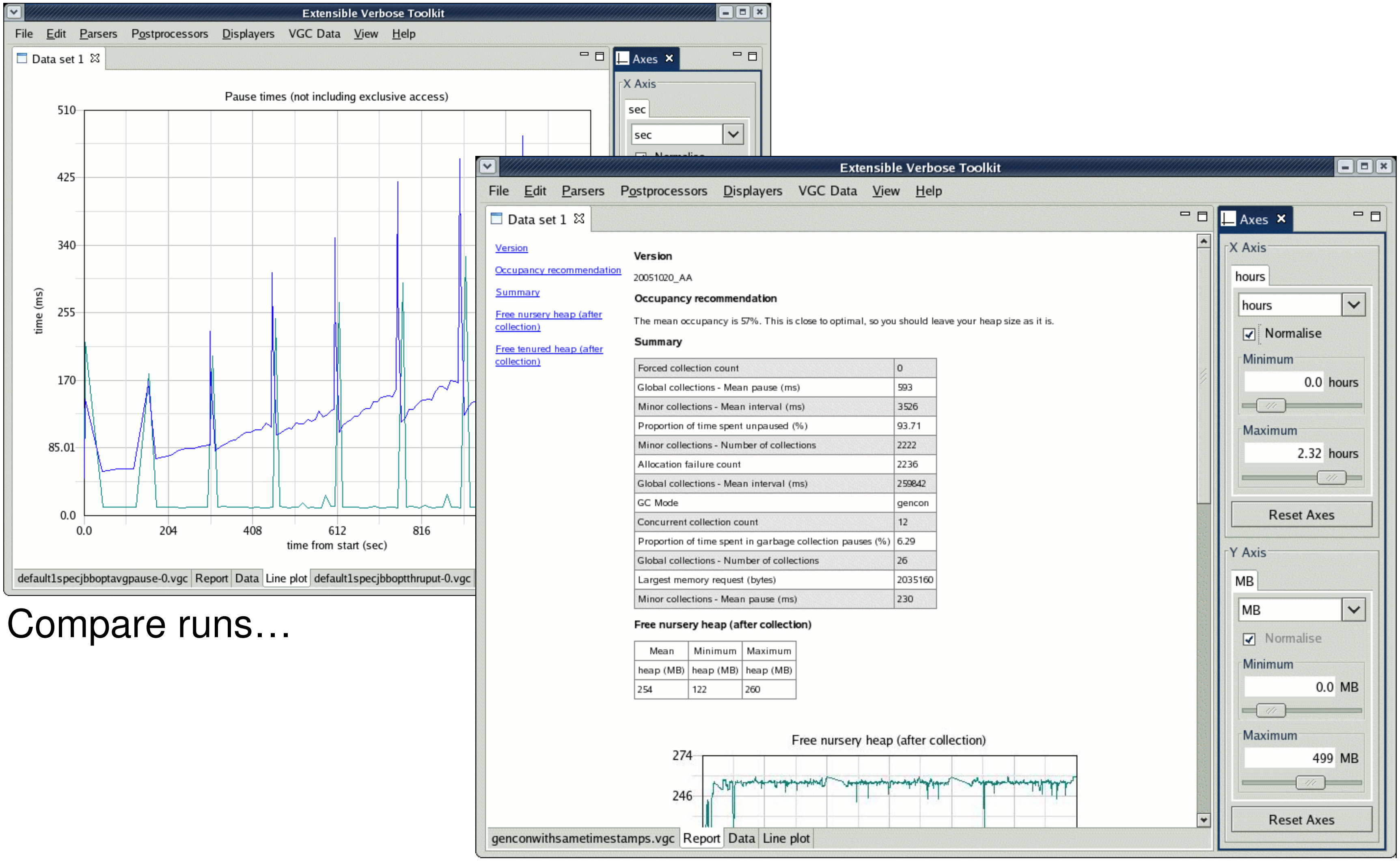
☑ Normalise

Minimum 0.06 sec

Maximum 2.57 sec

Pause times

EVTK - Comparison & Advice


Extensible Verbose Toolkit
File Edit Parsers Postprocessors Displayers VGC Data View Help
Data set 1
Axes
X Axis
sec
Pause times (not including exclusive access)
510
425
340
255
170
85.01
0.0
time (ms)
0.0
204
408
612
816
time from start (sec)
default1specjbboptavgpause-0.vgc
Report
Data
Line plot
default1specjbboptthruput-0.vgc
Extensible Verbose Toolkit
File Edit Parsers Postprocessors Displayers VGC Data View Help
Data set 1
Version
Occupancy recommendation
Summary
Free nursery heap (after collection)
Free tenured heap (after collection)
Axes
X Axis
hours
☑ Normalise
Minimum
0.0 hours
Maximum
2.32 hours
Reset Axes
Y Axis
MB
☑ Normalise
Minimum
0.0 MB
Maximum
499 MB
Reset Axes
Free nursery heap (after collection)
274
246
genconwithsametimestamps.vgc
Report
Data
Line plot

Version

20051020_AA

Occupancy recommendation

The mean occupancy is 57%. This is close to optimal, so you should leave your heap size as it is.

Summary

Forced collection count	0
Global collections - Mean pause (ms)	593
Minor collections - Mean interval (ms)	3526
Proportion of time spent unpaused (%)	93.71
Minor collections - Number of collections	2222
Allocation failure count	2236
Global collections - Mean interval (ms)	259842
GC Mode	gencon
Concurrent collection count	12
Proportion of time spent in garbage collection pauses (%)	6.29
Global collections - Number of collections	26
Largest memory request (bytes)	2035160
Minor collections - Mean pause (ms)	230

Free nursery heap (after collection)

Mean	Minimum	Maximum
heap (MB)	heap (MB)	heap (MB)
254	122	260

Compare runs…

Performance advisor…

DumpAnalyzer **initial** release

- Aim to diagnose
  - Deadlock in Java code
    - Report thread names / locations etc.
  - Out of memory condition
    - Report populations and large collections etc.
    - Summarise the native memory usage
    - Recommend further analysis using eg MDD4J
    - Analysis generally requires multiple heap dumps
  - Internal error (gpf etc.)
    - Is failure in non-IBM native code ?
      - Probably user coding error, report location etc.
      - If on J9 recommend running with -Xcheck:jni
    - Otherwise open PMR (use ISA to assist with data gathering)

DumpAnalyzer deadlock example

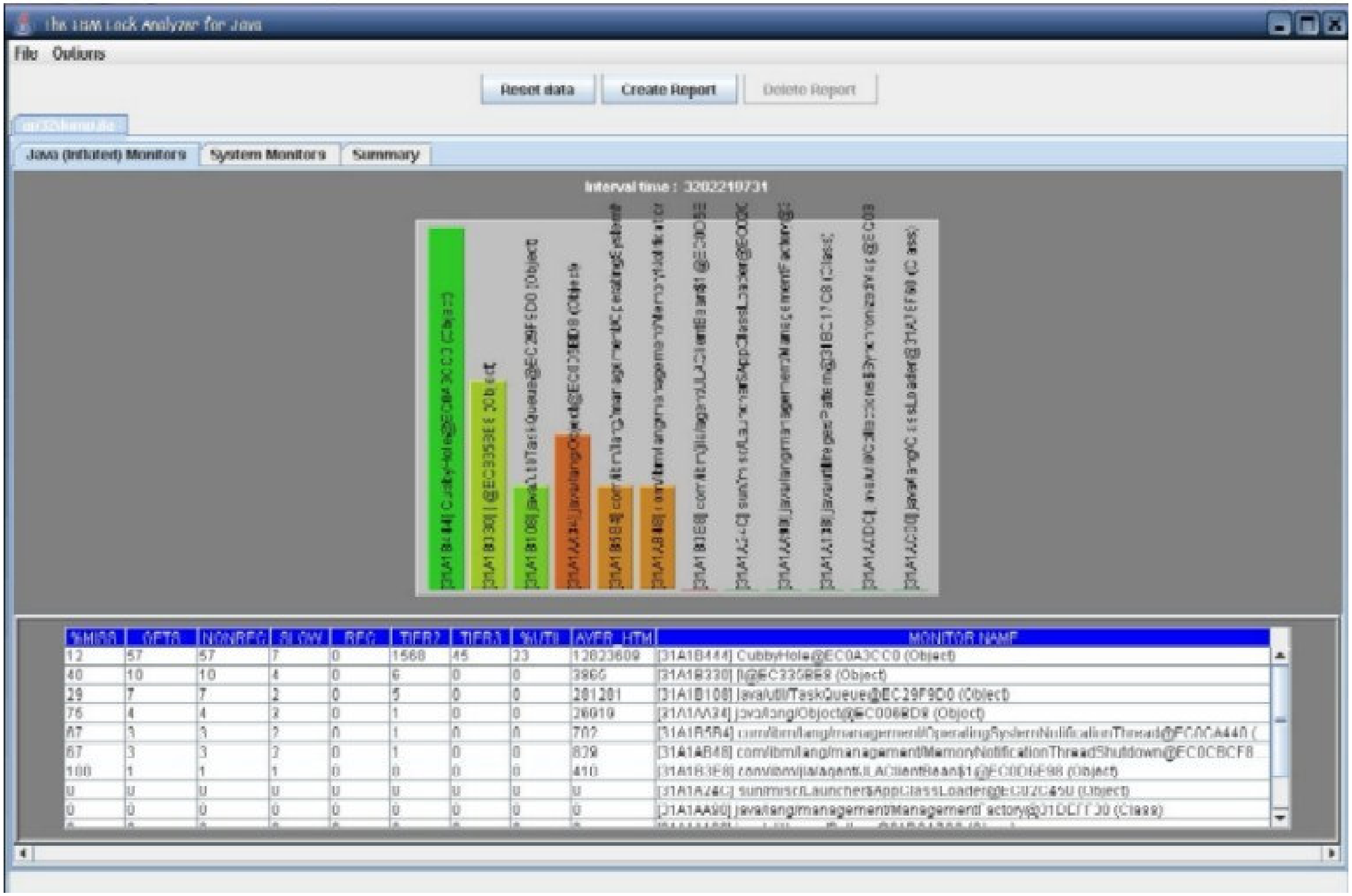
```
Start analysis of C:\a_tsm\workspaces\DumpAnalyzer\bin\core.20070315.101244.5436.dmp.zip
Execution time : 3925ms

Deadlocks detected - now trying to determine cause

Run report on analyzer: com.ibm.dtfj.analyzer.DTFJDeadlock rule: default

Error : Deadlock cycle detected on monitor com.ibm.dtfj.java.j9.JavaMonitor@7b3c48
Error : Deadlock cycle detected on monitor com.ibm.dtfj.java.j9.JavaMonitor@7b3c88
Error : Deadlock cycle detected on monitor com.ibm.dtfj.java.j9.JavaMonitor@7b35a8

Deadlock cycle detected
0 monitor <com.ibm.dtfj.java.j9.JavaMonitor@7b3c48> is owned by thread <Thread 1> which is waiting
    on monitor ...
1 monitor <com.ibm.dtfj.java.j9.JavaMonitor@7b3c88> is owned by thread <Thread 2> which is waiting
    on monitor ...
2 monitor <com.ibm.dtfj.java.j9.JavaMonitor@7b35a8> is owned by thread <Thread 3> which is waiting
    on monitor ...
3 monitor <com.ibm.dtfj.java.j9.JavaMonitor@7b3c48> is owned by thread <Thread 1> which is waiting
    on monitor ...
.
.

Thread Thread 1 : owned monitors and top 2 frames on stack
  owns com.ibm.dtfj.java.j9.JavaMonitor@7b3c48
  frame 1 com/ibm/monitor/test/TestDeadlock$AThread::grabLocks ()V lev 0
  frame 2 com/ibm/monitor/test/TestDeadlock$AThread::run ()V lev 0
Thread Thread 3 : owned monitors and top 2 frames on stack
  owns com.ibm.dtfj.java.j9.JavaMonitor@7b35a8
  frame 1 com/ibm/monitor/test/TestDeadlock$AThread::grabLocks ()V lev 0
  frame 2 com/ibm/monitor/test/TestDeadlock$AThread::run ()V lev 1
Thread Thread 2 : owned monitors and top 2 frames on stack
  owns com.ibm.dtfj.java.j9.JavaMonitor@7b3c88
  frame 1 com/ibm/monitor/test/TestDeadlock$AThread::grabLocks ()V lev 0
  frame 2 com/ibm/monitor/test/TestDeadlock$AThread::run ()V lev 1
```

```
============================ Deployed applications =============================

   DefaultApplication                   STARTED           DefaultApplication.ear/deployments/DefaultApplication
   ivtApp                               STARTED           ivtApp.ear/deployments/ivtApp
   query                                STARTED           query.ear/deployments/query
   SamplesGallery                       STARTED           SamplesGallery.ear/deployments/SamplesGallery
   PlantsByWebSphere                    STARTED           PlantsByWebSphere.ear/deployments/PlantsByWebSphere
   IBMUTC                               STARTED           IBMUTC.ear/deployments/IBMUTC
   ver/systemApp                        STARTED           e:/WAS61/AppServer/systemApps/ManagementEJB.ear
   ver/systemAp                         STARTED           e:/WAS61/AppServer/systemApps/filetransfer.ear
   ver/systemApps/Sch                   STARTED           e:/WAS61/AppServer/systemApps/SchedulerCalendars.ear
   isclite                              STARTED           isclite.ear/deployments/isclite
================================ EJB Container =================================

   Number of EJB modules defined: 6
   Number of EJB homes defined: 20
   EJB Container: com/ibm/ws/runtime/component/EJBContainerImpl@00F1EA88
   Number of EJBs currently in the EJB cache: 0
   Number of EJB wrappers currently in the wrapper cache: 0
   Local JNDI naming context name: ejb
================================ Web Container =================================
   Number of web modules defined: 14
   Number of servlets defined: 130
   Number of servlet requests currently active: 2
   Web Container: com/ibm/ws/webcontainer/component/WebContainerImpl@0104A9C8
   Number of requests: 0
================================= Thread Pools =================================
   Thread Pool:
      Name: PluginConfigService
      Min: 5                 Max: 20                 Grow-as-needed: false
      Current size: 0
      Current active threads: 0
   Thread Pool:
…..
```

Java Lock Analyser (JLA)

- JLA provides profiling data on monitors used in Java applications and the JVM:
  - Counters associated with contended locks
  - Total number of successful acquires
  - Recursive acquires
  - Frequency with which a thread had to block waiting on the monitor
  - Cumulative time the monitor was held.
  - For platforms that support 3 Tier Spin Locking the following are also collected
    - Number of times the requesting thread went through the inner (spin loop) while attempting to acquire the monitor.
    - Number of times the requesting thread went through the outer (thread yield loop) while attempting to acquire the monitor.

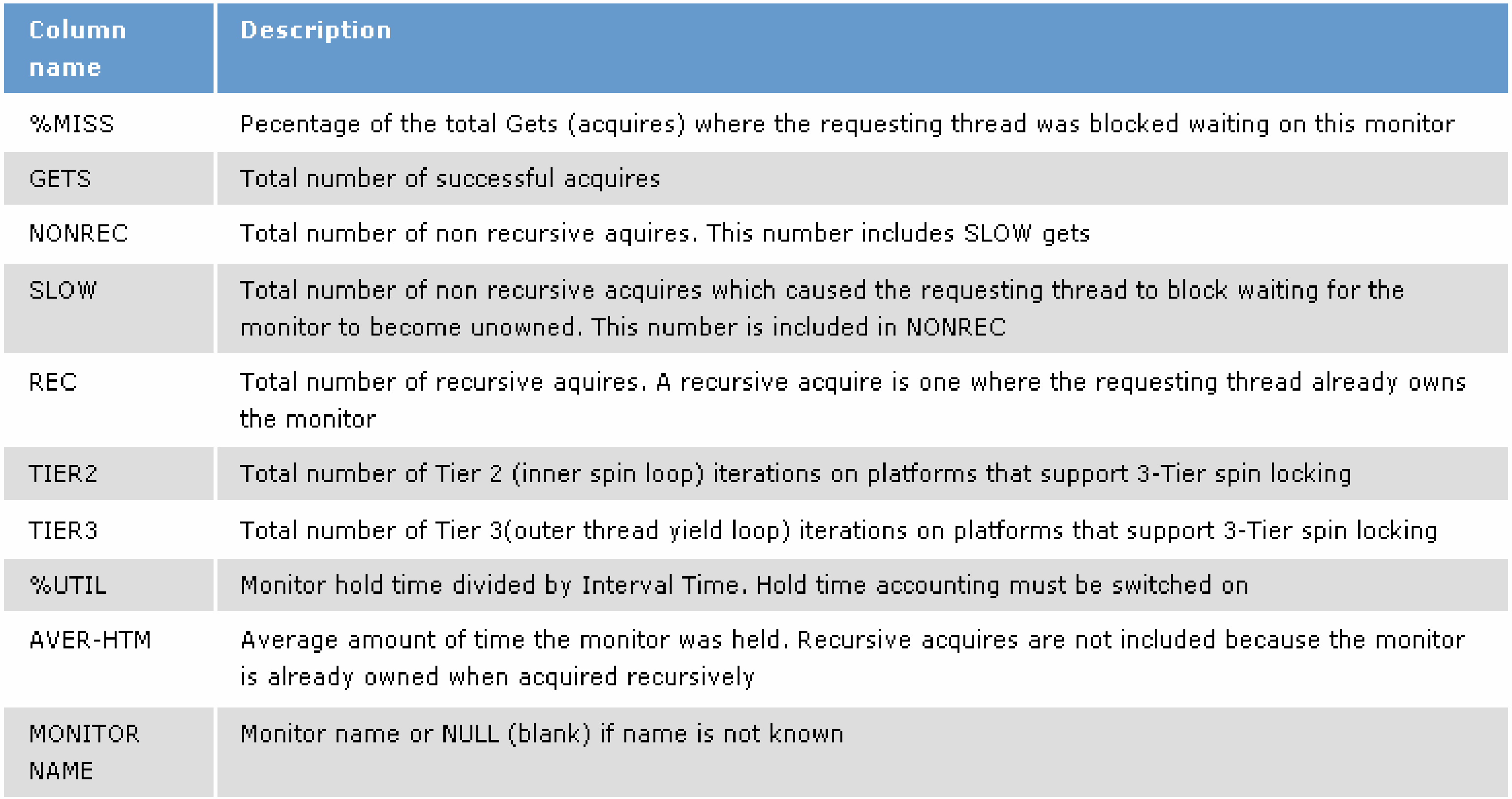
Java Lock Analyser (JLA)



Interval time : 3202210731

%MISS	GETS	NONREC	SLOW	REC	TIER2	TIER3	%UTIL	AVER_HTM	MONITOR NAME
12	57	57	7	0	1560	45	23	12823609	[31A1B444] CubbyHole@EC0A3CC0 (Object)
40	10	10	4	0	6	0	0	3866	[31A1B330] [I@EC335B88 (Object)
29	7	7	2	0	5	0	0	281281	[31A1B108] java/util/TaskQueue@EC29F9D0 (Object)
75	4	4	3	0	1	0	0	26010	[31A1AA34] java/lang/Object@EC008BD8 (Object)
67	3	3	2	0	1	0	0	702	[31A1B5B4] com/ibm/lang/management/OperatingSystemNotificationThread@EC0CA440 (
67	3	3	2	0	1	0	0	829	[31A1AB48] com/ibm/lang/management/MemoryNotificationThreadShutdown@EC0CBCF8
100	1	1	1	0	0	0	0	410	[31A1B3E8] com/ibm/jla/agent/JLAClientBean$1@EC0D6E98 (Object)
0	0	0	0	0	0	0	0	0	[31A1A24C] sun/misc/Launcher$AppClassLoader@EC02C450 (Object)
0	0	0	0	0	0	0	0	0	[31A1AA90] java/lang/management/ManagementFactory@31DE7730 (Class)

JLA Data Provided



Column name	Description
%MISS	Pecentage of the total Gets (acquires) where the requesting thread was blocked waiting on this monitor
GETS	Total number of successful acquires
NONREC	Total number of non recursive aquires. This number includes SLOW gets
SLOW	Total number of non recursive acquires which caused the requesting thread to block waiting for the monitor to become unowned. This number is included in NONREC
REC	Total number of recursive aquires. A recursive acquire is one where the requesting thread already owns the monitor
TIER2	Total number of Tier 2 (inner spin loop) iterations on platforms that support 3-Tier spin locking
TIER3	Total number of Tier 3(outer thread yield loop) iterations on platforms that support 3-Tier spin locking
%UTIL	Monitor hold time divided by Interval Time. Hold time accounting must be switched on
AVER-HTM	Average amount of time the monitor was held. Recursive acquires are not included because the monitor is already owned when acquired recursively
MONITOR NAME	Monitor name or NULL (blank) if name is not known

Discussion / Questions