Zing Vision

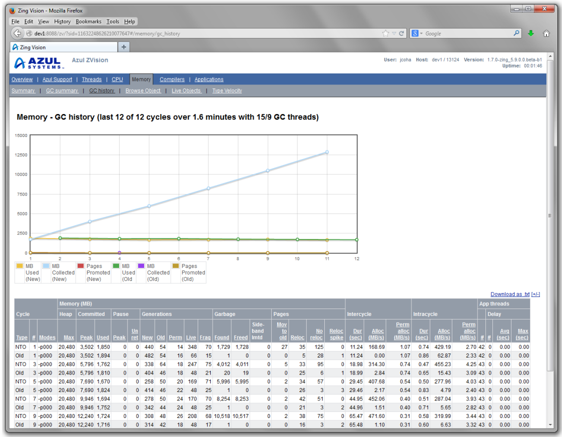
Answering your toughest production Java performance questions

Outline

- What is Zing Vision?
- Where does Zing Vision fit in your Java environment?
- Key features
- How it works
- Using ZVRobot
- Q & A

What is Zing Vision?

- Zing Vision is a browser-based, visual window into the Zing VM
  - – Hyperlinked display provides drill-down to root cause
  - – Both JVM internal and Java program information
  - – No additional performance overhead
  - – Nothing special to install or configure
  - – Easy to get started as a user
  - – ZVRobot collects and stores monitoring data
    - Collected data is same as ZVision!



Zing Vision - Mozilla Firefox
File Edit View History Bookmarks Tools Help
dev1:8088/zv/?sid=11632248626210077647#/memory/gc_history
Google
Zing Vision
AZUL SYSTEMS
Azul ZVision
Overview | Azul Support | Threads | CPU | Memory | Compilers | Applications
Summary | GC summary | GC history | Browse Object | Live Objects | Type Velocity
15000
12500
10000
7500
5000
2500
0
1
2
3
4
5
6
7
8
9
10
11
12
Download as .txt [+/-]

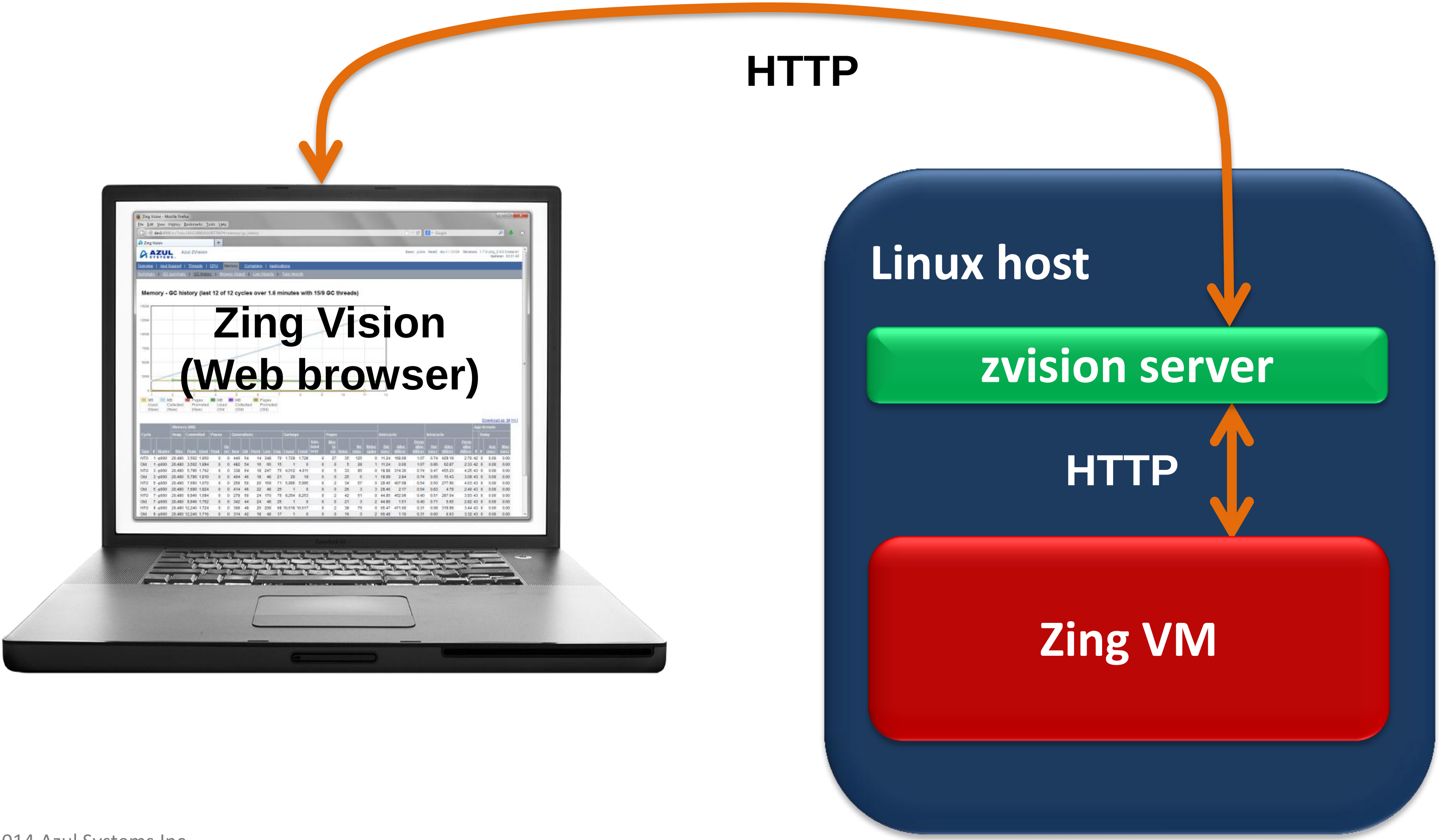
User: jcoha Host: dev1 / 13124 Version: 1.7.0-zing_5.9.0.0.beta-b1
Uptime: 00:01:46

Memory - GC history (last 12 of 12 cycles over 1.6 minutes with 15/9 GC threads)

▇	MB Used (New)	▇	MB Collected (New)	🟥	Pages Promoted (New)	🟩	MB Used (Old	▇	MB Collected (Old)	▇	Pages Promoted (Old)

Cycle			Memory (MB)																	Intercycle			Intracycle			App threads			
			Heap	Committed		Pause		Generations					Garbage			Pages											Delay		
Type	#	Modes	Max	Peak	Used	Peak	Un ret	New	Old	Perm	Live	Frag	Found	Freed	Side-band lmtd	Mov to old	Reloc	No reloc	Reloc spike	Dur (sec)	Alloc (MB/s)	Perm alloc (MB/s)	Dur (sec)	Alloc (MB/s)	Perm alloc (MB/s)	#	#	Avg (sec)	Max (sec)
NTO	1	-p000	20,480	3,502	1,850	0	0	440	54	14	348	70	1,729	1,728	0	27	35	125	0	11.24	168.69	1.07	0.74	429.19	2.70	42	0	0.00	0.00
Old	1	-p000	20,480	3,502	1,894	0	0	482	54	16	66	15	1	0	0	0	5	28	1	11.24	0.00	1.07	0.86	62.87	2.33	42	0	0.00	0.00
NTO	3	-p000	20,480	5,796	1,762	0	0	338	64	18	247	75	4,012	4,011	0	5	33	95	0	18.98	314.30	0.74	0.47	455.23	4.25	43	0	0.00	0.00
Old	3	-p000	20,480	5,796	1,810	0	0	404	46	18	48	21	20	19	0	0	25	6	1	18.99	2.84	0.74	0.65	15.43	3.09	43	0	0.00	0.00
NTO	5	-p000	20,480	7,690	1,670	0	0	258	50	20	169	71	5,996	5,995	0	2	34	57	0	29.45	407.68	0.54	0.50	277.96	4.03	43	0	0.00	0.00
Old	5	-p000	20,480	7,690	1,824	0	0	414	46	22	48	25	1	0	0	0	26	3	3	29.46	2.17	0.54	0.83	4.79	2.40	43	0	0.00	0.00
NTO	7	-p000	20,480	9,946	1,694	0	0	278	50	24	170	70	8,254	8,253	0	2	42	51	0	44.95	452.06	0.40	0.51	287.04	3.93	43	0	0.00	0.00
Old	7	-p000	20,480	9,946	1,752	0	0	342	44	24	48	25	1	0	0	0	21	3	2	44.96	1.51	0.40	0.71	5.65	2.82	43	0	0.00	0.00
NTO	9	-p000	20,480	12,240	1,724	0	0	308	48	26	208	68	10,518	10,517	0	2	38	75	0	65.47	471.60	0.31	0.58	319.99	3.44	43	0	0.00	0.00
Old	9	-p000	20,480	12,240	1,716	0	0	314	42	18	48	17	1	0	0	0	16	3	2	65.48	1.10	0.31	0.60	6.63	3.32	43	0	0.00	0.00

Example Zing Vision deployment



HTTP

Zing Vision (Web browser)

Linux host

zvision server

HTTP

Zing VM

©2014 Azul Systems Inc.

Fitting in: Production focus

- Most tools suitable for developers, but too ‘heavy’ for production
  - Use JVMTI (JVM Tools Interface) and BCI (Byte Code Instrumentation)
  - High overhead, so must be used carefully in production
  - May require configuration to lower the overhead
  - Not practical for Operations teams
- Zing Vision is designed for safe use on production systems
  - Ideal diagnostic tool to use when you know you have a problem in production (“a flashlight in a dark place”)
  - Integrated with the JVM, so it won’t add overhead or perturb the running Java program
  - It’s meant to be used - click on items in the web UI to explore!

Key features

- Tick profiler
- Thread-level views
- Lock contention view
- Garbage collection views
- Java heap object views
    - Live objects
    - Object types that increase in number as application runs

Tick Profiler: Find "hot" code

- Goal: Determine what code is hot (consuming most CPU time)

- Approach used by other profiling tools:
  - Method tracing
  - Byte code instrumentation – heavy weight
  - Solution: Profile only a user-selected portion of the application

  - Drawback: You need to know the location in the code where the performance bottleneck occurs to select the area of the code to profile!

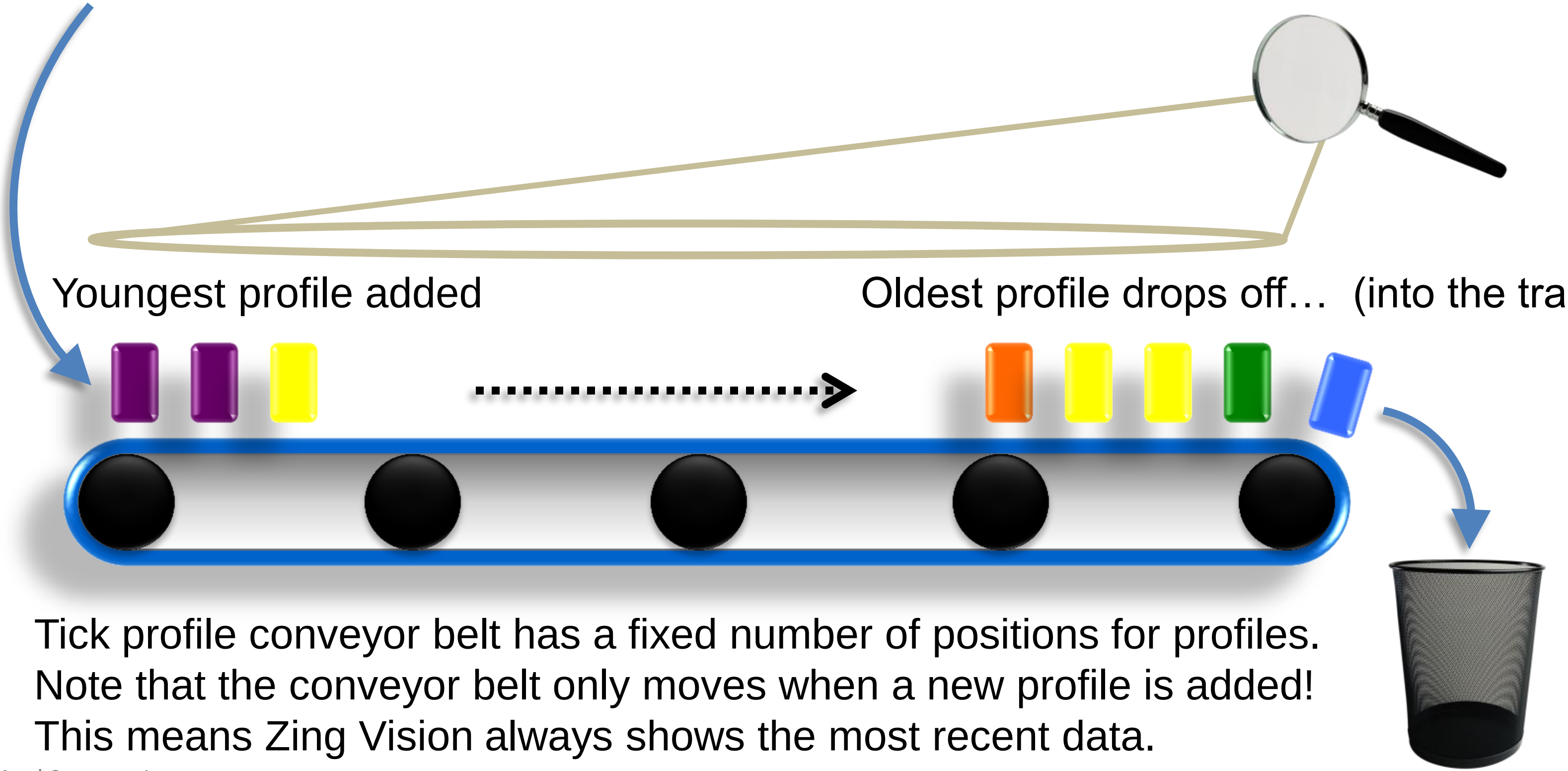
Tick Profiler: Find “hot” code

Zing Vision uses runtime thread sampling:

- Fast snapshots of the code running in executing Java threads
- Identifies where the work is done in the application
- Lightweight, low overhead
  - Always on
  - Even when you’re not looking at the generated metrics
- Instruction-level granularity
  - JVM internal threads, too!
    - Detailed information about the entire process
  - Interpretation sometimes requires understanding of the JVM runtime

Tick Profiler: How it works

1. Thread registers SIG61 signal handler with kernel at intervals 1 ms (1000 times per second, configurable)
2. Thread runs
3. Kernel delivers SIG61
4. Thread is interrupted on its normal stack
5. Signal handler runs, creates a new (youngest) profile
6. Zing Vision aggregates all of the profiles on the belt in the Timer Tick Profile view

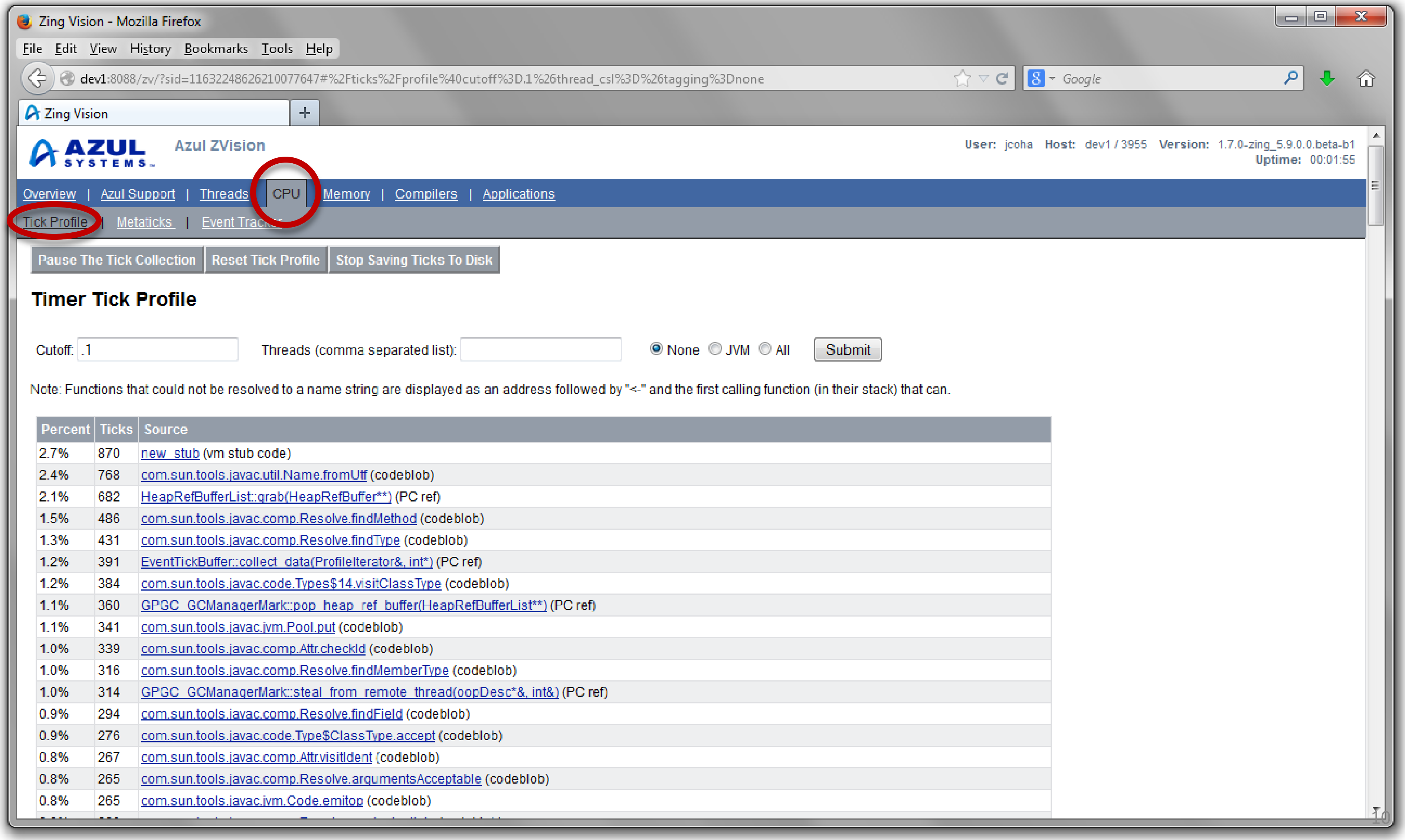


Youngest profile added

Oldest profile drops off… (into the trash)

Tick profile conveyor belt has a fixed number of positions for profiles.
Note that the conveyor belt only moves when a new profile is added!
This means Zing Vision always shows the most recent data.

Tick Profiler: Where is work done?


Zing Vision - Mozilla Firefox
File Edit View History Bookmarks Tools Help
dev1:8088/zv/?sid=11632248626210077647#%2Fticks%2Fprofile%40cutoff%3D.1%26thread_csl%3D%26tagging%3Dnone
Google
Zing Vision
AZUL SYSTEMS
Azul ZVision
Overview | Azul Support | Threads | CPU | Memory | Compilers | Applications
Tick Profile | Metaticks | Event Tracker
Pause The Tick Collection
Reset Tick Profile
Stop Saving Ticks To Disk

User: jcoha Host: dev1 / 3955 Version: 1.7.0-zing_5.9.0.0.beta-b1
Uptime: 00:01:55

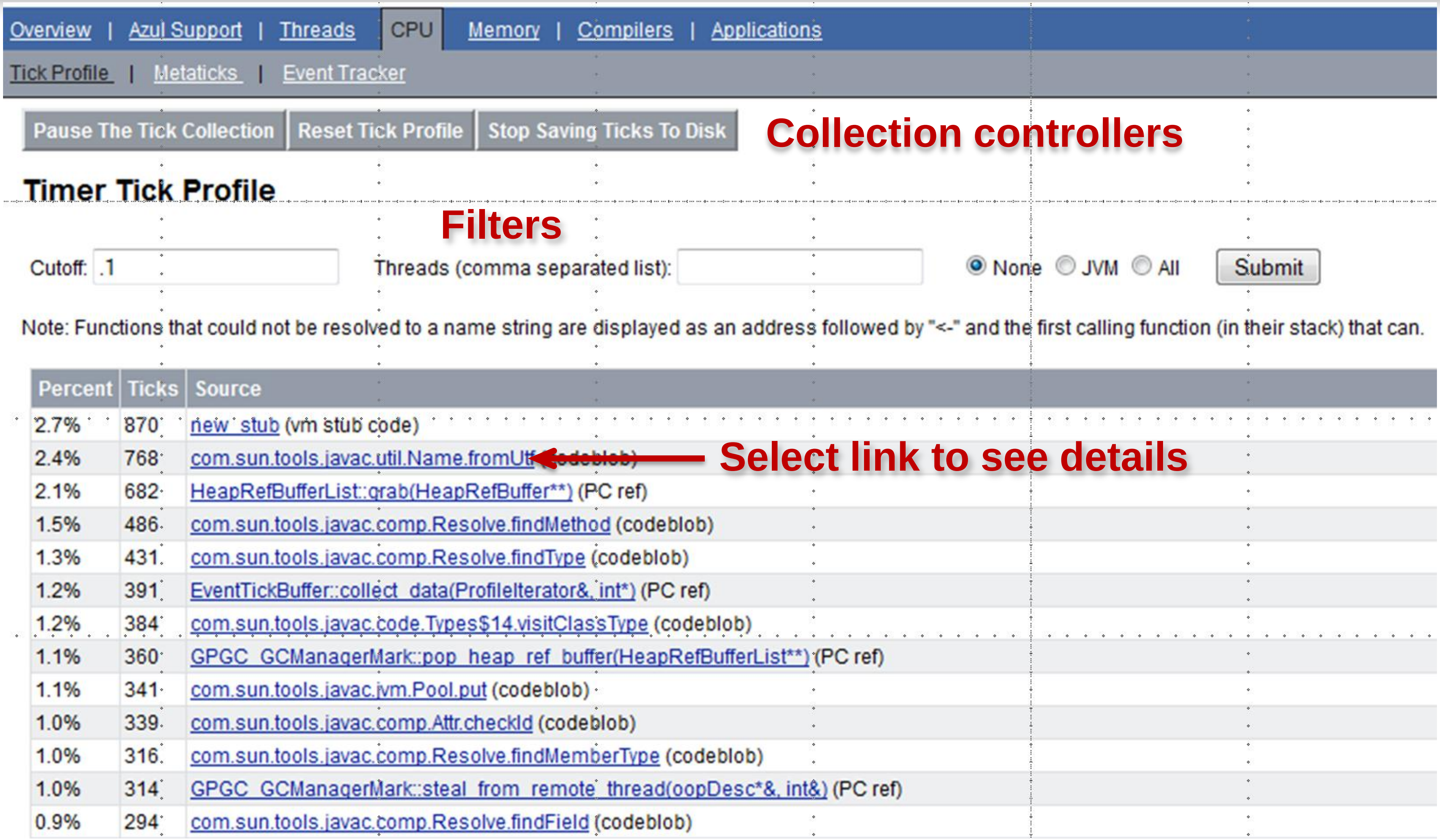
Timer Tick Profile

Cutoff: .1 Threads (comma separated list): ● None ○ JVM ○ All Submit

Note: Functions that could not be resolved to a name string are displayed as an address followed by "<-" and the first calling function (in their stack) that can.

Percent	Ticks	Source
2.7%	870	new_stub (vm stub code)
2.4%	768	com.sun.tools.javac.util.Name.fromUtf (codeblob)
2.1%	682	HeapRefBufferList::grab(HeapRefBuffer**) (PC ref)
1.5%	486	com.sun.tools.javac.comp.Resolve.findMethod (codeblob)
1.3%	431	com.sun.tools.javac.comp.Resolve.findType (codeblob)
1.2%	391	EventTickBuffer::collect_data(ProfileIterator&, int*) (PC ref)
1.2%	384	com.sun.tools.javac.code.Types$14.visitClassType (codeblob)
1.1%	360	GPGC_GCManagerMark::pop_heap_ref_buffer(HeapRefBufferList**) (PC ref)
1.1%	341	com.sun.tools.javac.jvm.Pool.put (codeblob)
1.0%	339	com.sun.tools.javac.comp.Attr.checkId (codeblob)
1.0%	316	com.sun.tools.javac.comp.Resolve.findMemberType (codeblob)
1.0%	314	GPGC_GCManagerMark::steal_from_remote_thread(oopDesc*&, int&) (PC ref)
0.9%	294	com.sun.tools.javac.comp.Resolve.findField (codeblob)
0.9%	276	com.sun.tools.javac.code.Type$ClassType.accept (codeblob)
0.8%	267	com.sun.tools.javac.comp.Attr.visitIdent (codeblob)
0.8%	265	com.sun.tools.javac.comp.Resolve.argumentsAcceptable (codeblob)
0.8%	265	com.sun.tools.javac.jvm.Code.emitop (codeblob)

Tick Profiler: Where is work done?



Overview | Azul Support | Threads | CPU | Memory | Compilers | Applications

Tick Profile | Metaticks | Event Tracker

Pause The Tick Collection | Reset Tick Profile | Stop Saving Ticks To Disk

Collection controllers

Timer Tick Profile

Filters

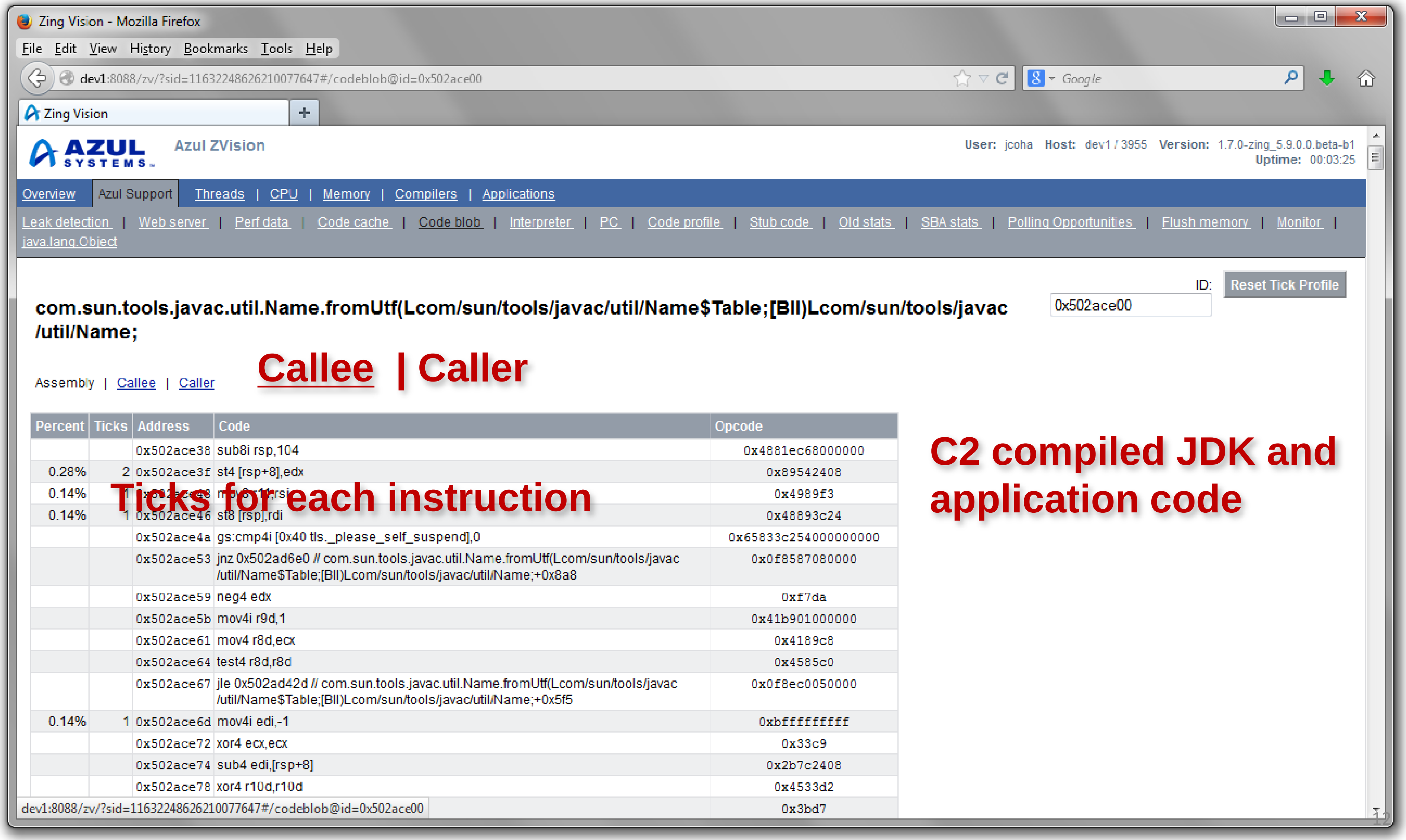
Cutoff: .1 Threads (comma separated list): ● None ○ JVM ○ All Submit

Note: Functions that could not be resolved to a name string are displayed as an address followed by "<-" and the first calling function (in their stack) that can.

Percent	Ticks	Source
2.7%	870	new_stub (vm stub code)
2.4%	768	com.sun.tools.javac.util.Name.fromUtf (codeblob)
2.1%	682	HeapRefBufferList::grab(HeapRefBuffer**) (PC ref)
1.5%	486	com.sun.tools.javac.comp.Resolve.findMethod (codeblob)
1.3%	431	com.sun.tools.javac.comp.Resolve.findType (codeblob)
1.2%	391	EventTickBuffer::collect_data(ProfileIterator&, int*) (PC ref)
1.2%	384	com.sun.tools.javac.code.Types$14.visitClassType (codeblob)
1.1%	360	GPGC_GCManagerMark::pop_heap_ref_buffer(HeapRefBufferList**) (PC ref)
1.1%	341	com.sun.tools.javac.jvm.Pool.put (codeblob)
1.0%	339	com.sun.tools.javac.comp.Attr.checkId (codeblob)
1.0%	316	com.sun.tools.javac.comp.Resolve.findMemberType (codeblob)
1.0%	314	GPGC_GCManagerMark::steal_from_remote_thread(oopDesc*&, int&) (PC ref)
0.9%	294	com.sun.tools.javac.comp.Resolve.findField (codeblob)

Select link to see details

Tick Profiler: Work at instruction level


Zing Vision - Mozilla Firefox
File Edit View History Bookmarks Tools Help
dev1:8088/zv/?sid=11632248626210077647#/codeblob@id=0x502ace00
Google
Zing Vision
Azul ZVision
Overview
Azul Support
Threads | CPU | Memory | Compilers | Applications
Leak detection | Web server | Perf data | Code cache | Code blob | Interpreter | PC | Code profile | Stub code | Old stats | SBA stats | Polling Opportunities | Flush memory | Monitor | java.lang.Object
Reset Tick Profile
Assembly | Callee | Caller

User: jcoha Host: dev1 / 3955 Version: 1.7.0-zing_5.9.0.0.beta-b1
Uptime: 00:03:25

ID: 0x502ace00

com.sun.tools.javac.util.Name.fromUtf(Lcom/sun/tools/javac/util/Name$Table;[BII)Lcom/sun/tools/javac/util/Name;

Percent	Ticks	Address	Code	Opcode
		0x502ace38	sub8i rsp,104	0x4881ec68000000
0.28%	2	0x502ace3f	st4 [rsp+8],edx	0x89542408
0.14%	1	0x502ace43	mov8 r11,rsi	0x4989f3
0.14%	1	0x502ace46	st8 [rsp],rdi	0x48893c24
		0x502ace4a	gs:cmp4i [0x40 tls._please_self_suspend],0	0x65833c2540000000000
		0x502ace53	jnz 0x502ad6e0 // com.sun.tools.javac.util.Name.fromUtf(Lcom/sun/tools/javac/util/Name$Table;[BII)Lcom/sun/tools/javac/util/Name;+0x8a8	0x0f8587080000
		0x502ace59	neg4 edx	0xf7da
		0x502ace5b	mov4i r9d,1	0x41b901000000
		0x502ace61	mov4 r8d,ecx	0x4189c8
		0x502ace64	test4 r8d,r8d	0x4585c0
		0x502ace67	jle 0x502ad42d // com.sun.tools.javac.util.Name.fromUtf(Lcom/sun/tools/javac/util/Name$Table;[BII)Lcom/sun/tools/javac/util/Name;+0x5f5	0x0f8ec0050000
0.14%	1	0x502ace6d	mov4i edi,-1	0xbfffffffff
		0x502ace72	xor4 ecx,ecx	0x33c9
		0x502ace74	sub4 edi,[rsp+8]	0x2b7c2408
		0x502ace78	xor4 r10d,r10d	0x4533d2
				0x3bd7

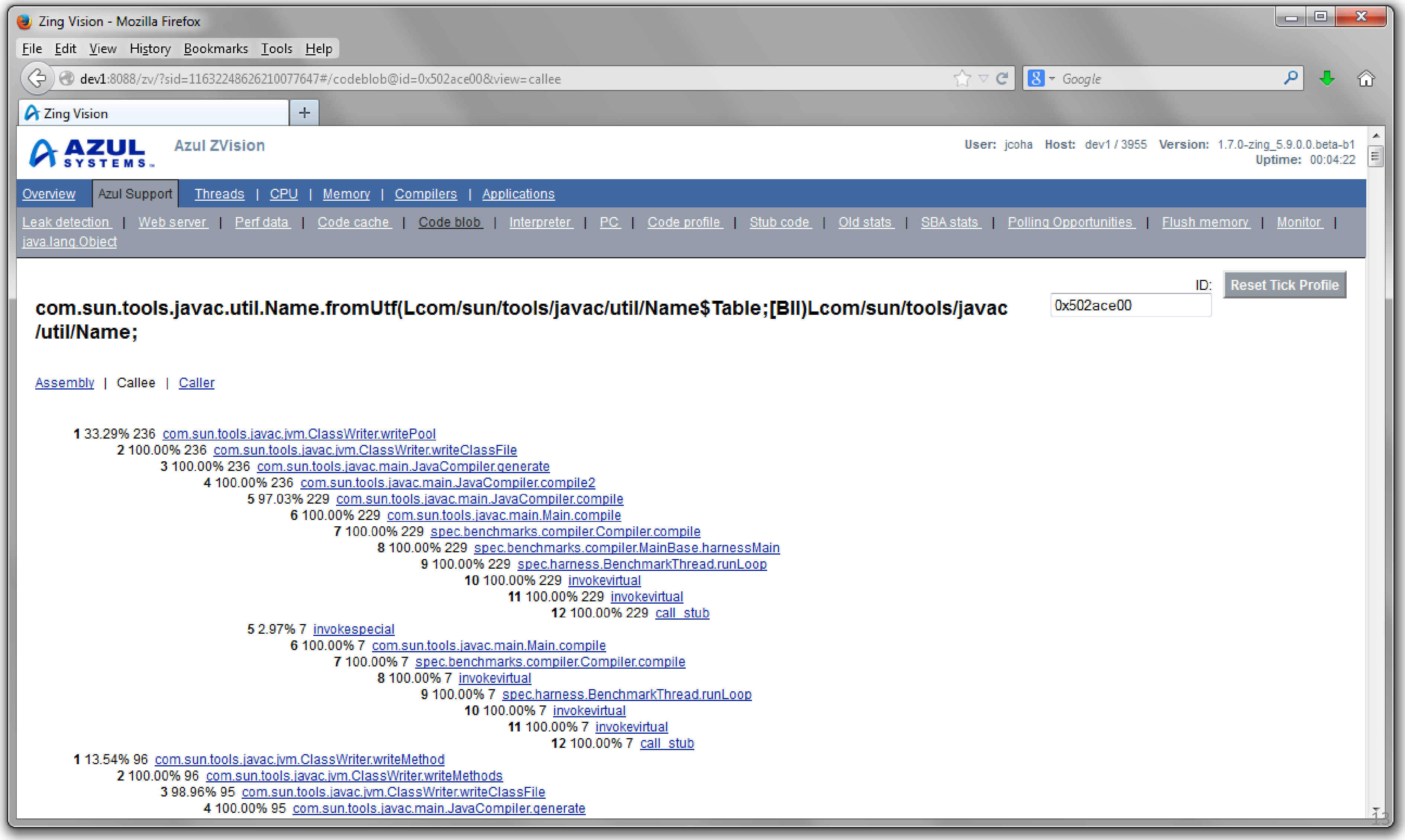
dev1:8088/zv/?sid=11632248626210077647#/codeblob@id=0x502ace00

Callee | Caller

Ticks for each instruction

C2 compiled JDK and application code

Tick Profiler: How did I get here?


Zing Vision - Mozilla Firefox
File Edit View History Bookmarks Tools Help
dev1:8088/zv/?sid=1163224862621007764 7#/codeblob@id=0x502ace00&view=callee
Google
Zing Vision
AZUL SYSTEMS
Azul ZVision
User: jcoha Host: dev1 / 3955 Version: 1.7.0-zing_5.9.0.0.beta-b1
Uptime: 00:04:22
Overview | Azul Support | Threads | CPU | Memory | Compilers | Applications
Leak detection | Web server | Perf data | Code cache | Code blob | Interpreter | PC | Code profile | Stub code | Old stats | SBA stats | Polling Opportunities | Flush memory | Monitor | java.lang.Object
ID:
Reset Tick Profile
Assembly | Callee | Caller

ID: 0x502ace00

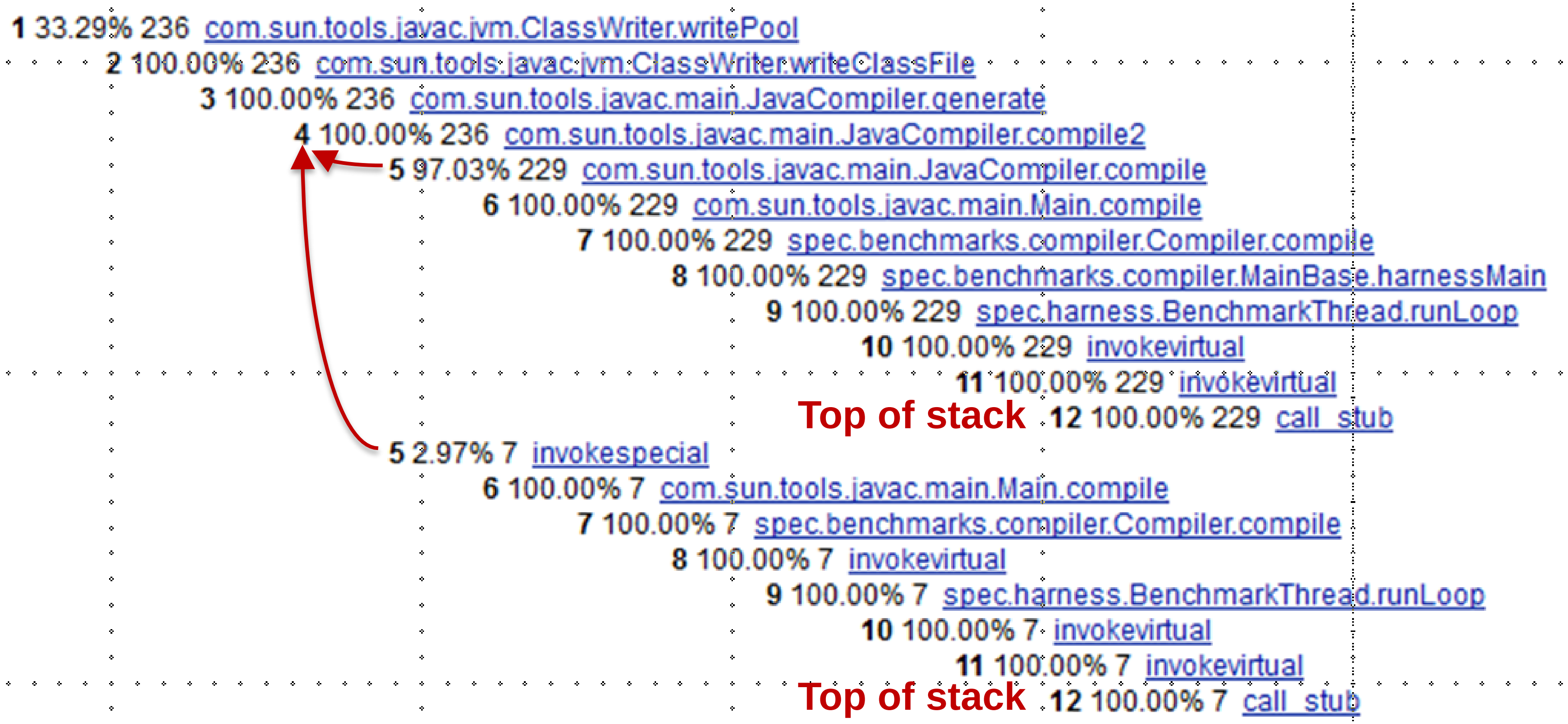
com.sun.tools.javac.util.Name.fromUtf(Lcom/sun/tools/javac/util/Name$Table;[BII)Lcom/sun/tools/javac/util/Name;

```
1 33.29% 236  com.sun.tools.javac.jvm.ClassWriter.writePool
    2 100.00% 236  com.sun.tools.javac.jvm.ClassWriter.writeClassFile
        3 100.00% 236  com.sun.tools.javac.main.JavaCompiler.generate
            4 100.00% 236  com.sun.tools.javac.main.JavaCompiler.compile2
                5 97.03% 229  com.sun.tools.javac.main.JavaCompiler.compile
                    6 100.00% 229  com.sun.tools.javac.main.Main.compile
                        7 100.00% 229  spec.benchmarks.compiler.Compiler.compile
                            8 100.00% 229  spec.benchmarks.compiler.MainBase.harnessMain
                                9 100.00% 229  spec.harness.BenchmarkThread.runLoop
                                    10 100.00% 229  invokevirtual
                                        11 100.00% 229  invokevirtual
                                            12 100.00% 229  call_stub
                5 2.97% 7  invokespecial
                    6 100.00% 7  com.sun.tools.javac.main.Main.compile
                        7 100.00% 7  spec.benchmarks.compiler.Compiler.compile
                            8 100.00% 7  invokevirtual
                                9 100.00% 7  spec.harness.BenchmarkThread.runLoop
                                    10 100.00% 7  invokevirtual
                                        11 100.00% 7  invokevirtual
                                            12 100.00% 7  call_stub
1 13.54% 96  com.sun.tools.javac.jvm.ClassWriter.writeMethod
    2 100.00% 96  com.sun.tools.javac.jvm.ClassWriter.writeMethods
        3 98.96% 95  com.sun.tools.javac.jvm.ClassWriter.writeClassFile
            4 100.00% 95  com.sun.tools.javac.main.JavaCompiler.generate
```

Tick Profiler: How did I get here?

Sorted based on highest CPU consumers

0 com.sun.tools.javac.util.Name.fromUtf



```
1 33.29% 236  com.sun.tools.javac.jvm.ClassWriter.writePool
    2 100.00% 236  com.sun.tools.javac.jvm.ClassWriter.writeClassFile
        3 100.00% 236  com.sun.tools.javac.main.JavaCompiler.generate
            4 100.00% 236  com.sun.tools.javac.main.JavaCompiler.compile2
                5 97.03% 229  com.sun.tools.javac.main.JavaCompiler.compile
                    6 100.00% 229  com.sun.tools.javac.main.Main.compile
                        7 100.00% 229  spec.benchmarks.compiler.Compiler.compile
                            8 100.00% 229  spec.benchmarks.compiler.MainBase.harnessMain
                                9 100.00% 229  spec.harness.BenchmarkThread.runLoop
                                    10 100.00% 229  invokevirtual
                                        11 100.00% 229  invokevirtual
                                            12 100.00% 229  call_stub
                5 2.97% 7  invokespecial
                    6 100.00% 7  com.sun.tools.javac.main.Main.compile
                        7 100.00% 7  spec.benchmarks.compiler.Compiler.compile
                            8 100.00% 7  invokevirtual
                                9 100.00% 7  spec.harness.BenchmarkThread.runLoop
                                    10 100.00% 7  invokevirtual
                                        11 100.00% 7  invokevirtual
                                            12 100.00% 7  call_stub
```

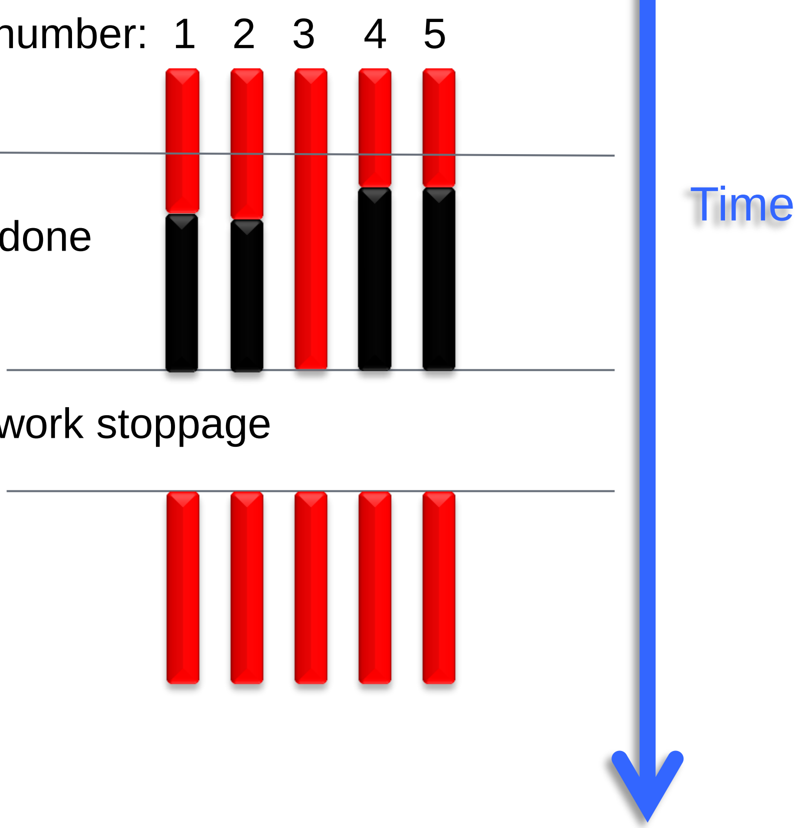
Top of stack

Top of stack

Thread-level questions

- What threads are executing my application?
- What are the housekeeping threads for the JVM?
- Can I examine a thread object?
- What are the threads doing?
  - Profiling information
  - Stack traces
- Where is my application stalled?

Thread-level views: Standard method



Request thread stacks (kill -3)

Some threads blocked – NO work done
Thread 3 continues to run!

Last thread stopped
O(numThreads) delay → application visible work stoppage

Resume threads

Thread-level views: Zing Vision method

Request thread stacks (ZVision)

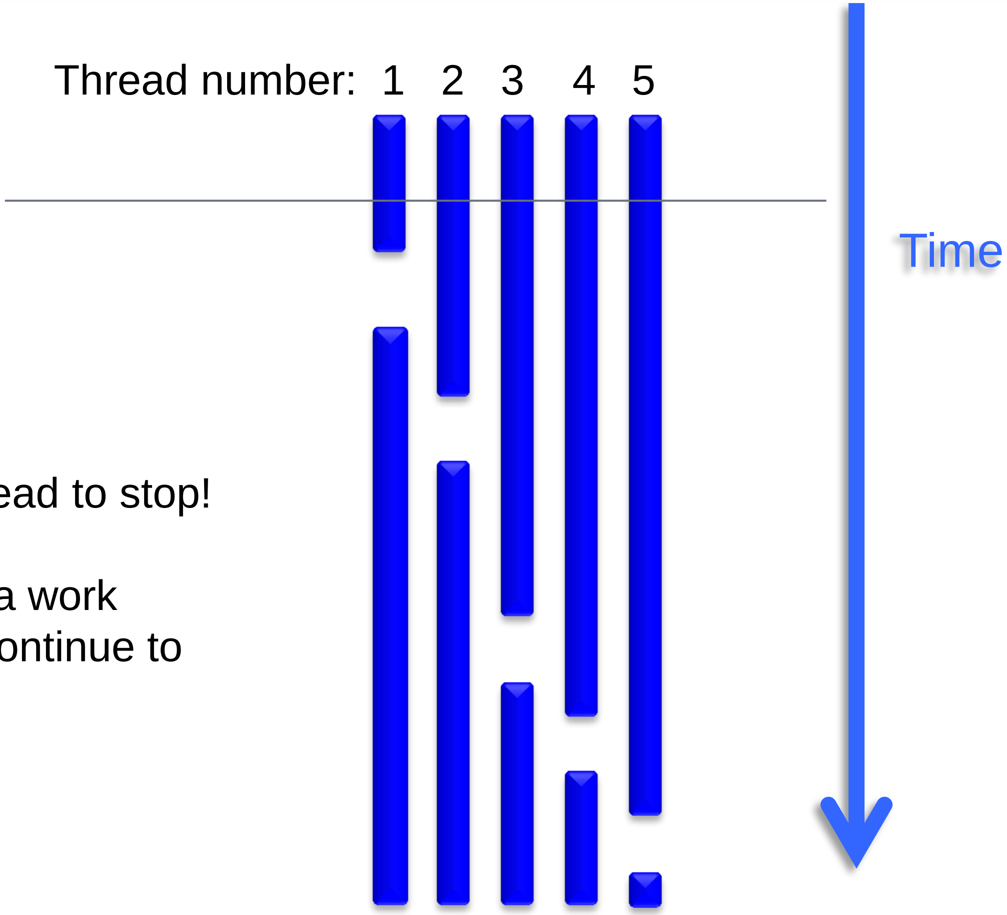
Each thread:

1. Stopped individually
2. Delay order is O(1)

No thread waits for any other thread to stop!

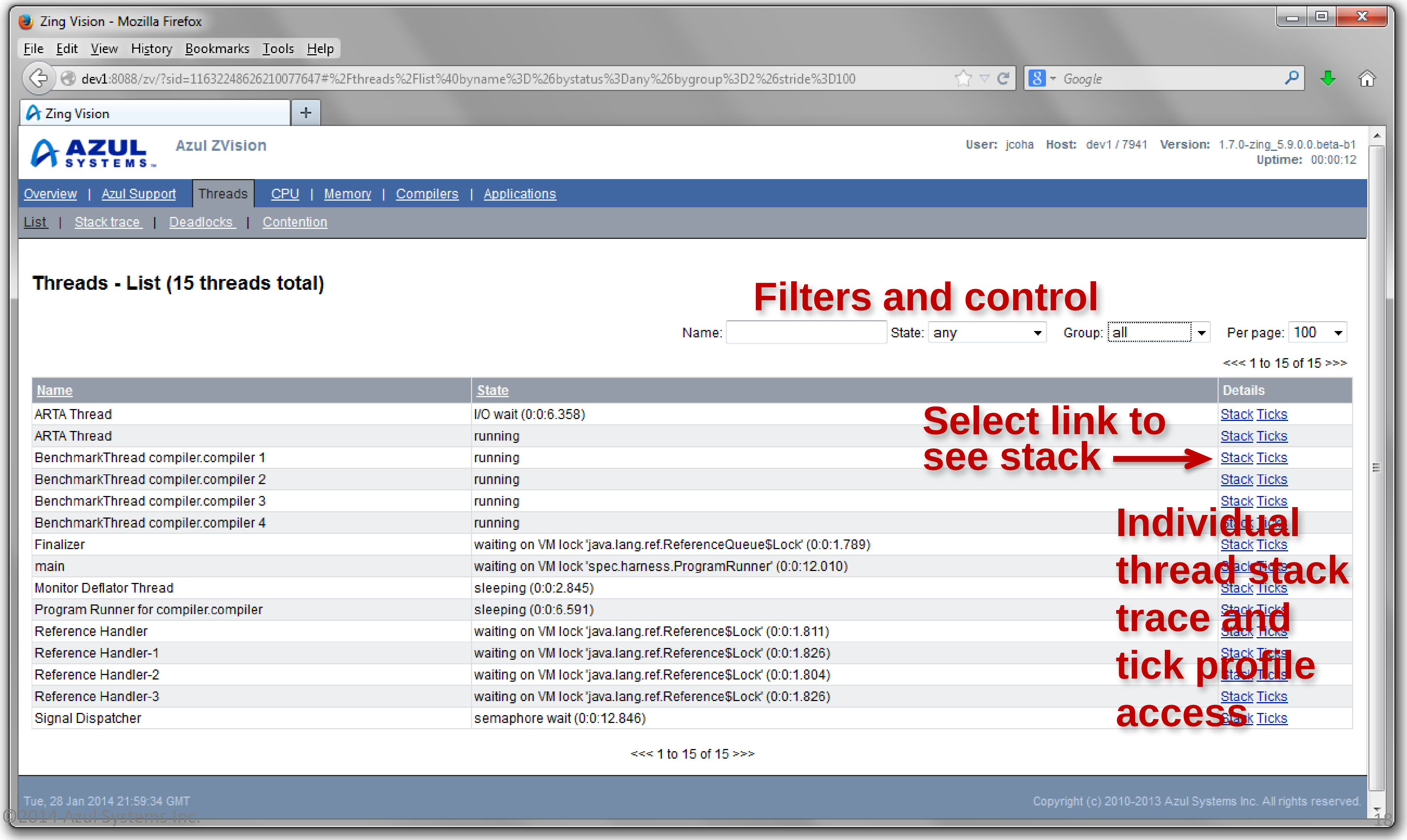
The application never perceives a work stoppage because N-1 threads continue to run at all times

Resume each thread seamlessly



Pro: Safe to use in production (guilt-free clicking)
Con: Not 100% consistent (especially for lock data)

Thread-level view: Application threads



Zing Vision - Mozilla Firefox
File Edit View History Bookmarks Tools Help
dev1:8088/zv/?sid=11632248626210077647#%2Fthreads%2Flist%40byname%3D%26bystatus%3Dany%26bygroup%3D2%26stride%3D100
Google
Zing Vision
Azul ZVision
Overview | Azul Support | Threads | CPU | Memory | Compilers | Applications
List | Stack trace | Deadlocks | Contention

User: jcoha Host: dev1 / 7941 Version: 1.7.0-zing_5.9.0.0.beta-b1
Uptime: 00:00:12

Threads - List (15 threads total)

Filters and control

Name: State: any Group: all Per page: 100

<<< 1 to 15 of 15 >>>

Name	State	Details
ARTA Thread	I/O wait (0:0:6.358)	Stack Ticks
ARTA Thread	running	Stack Ticks
BenchmarkThread compiler.compiler 1	running	Stack Ticks
BenchmarkThread compiler.compiler 2	running	Stack Ticks
BenchmarkThread compiler.compiler 3	running	Stack Ticks
BenchmarkThread compiler.compiler 4	running	Stack Ticks
Finalizer	waiting on VM lock 'java.lang.ref.ReferenceQueue$Lock' (0:0:1.789)	Stack Ticks
main	waiting on VM lock 'spec.harness.ProgramRunner' (0:0:12.010)	Stack Ticks
Monitor Deflator Thread	sleeping (0:0:2.845)	Stack Ticks
Program Runner for compiler.compiler	sleeping (0:0:6.591)	Stack Ticks
Reference Handler	waiting on VM lock 'java.lang.ref.Reference$Lock' (0:0:1.811)	Stack Ticks
Reference Handler-1	waiting on VM lock 'java.lang.ref.Reference$Lock' (0:0:1.826)	Stack Ticks
Reference Handler-2	waiting on VM lock 'java.lang.ref.Reference$Lock' (0:0:1.804)	Stack Ticks
Reference Handler-3	waiting on VM lock 'java.lang.ref.Reference$Lock' (0:0:1.826)	Stack Ticks
Signal Dispatcher	semaphore wait (0:0:12.846)	Stack Ticks

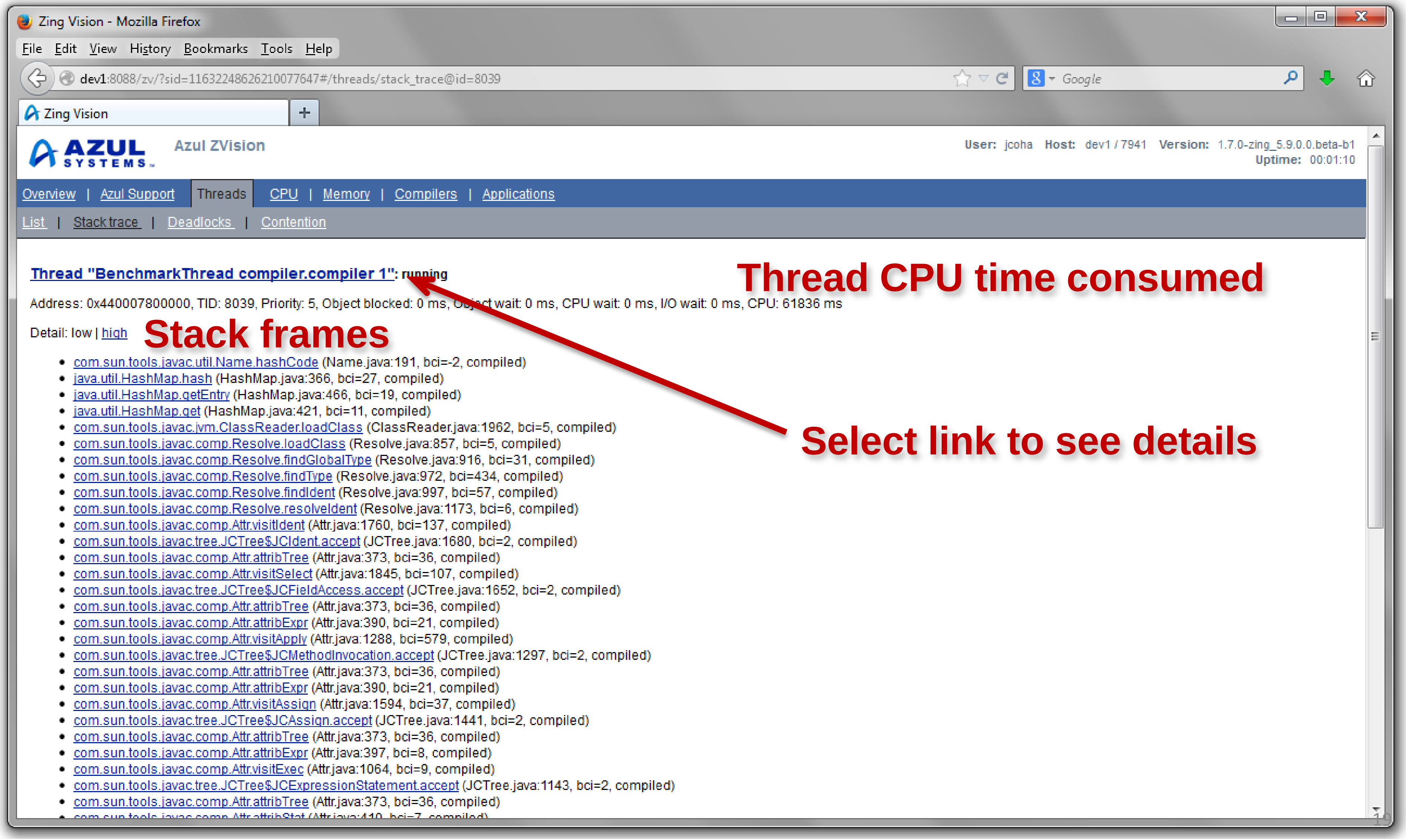
Select link to see stack

Individual thread stack trace and tick profile access

<<< 1 to 15 of 15 >>>

Tue, 28 Jan 2014 21:59:34 GMT

Thread-level view: Thread stack trace


Zing Vision - Mozilla Firefox
File Edit View History Bookmarks Tools Help
dev1:8088/zv/?sid=11632248626210077647#/threads/stack_trace@id=8039
Google
Zing Vision
+
AZUL SYSTEMS
Azul ZVision
Overview | Azul Support | Threads | CPU | Memory | Compilers | Applications
List | Stack trace | Deadlocks | Contention

User: jcoha Host: dev1 / 7941 Version: 1.7.0-zing_5.9.0.0.beta-b1
Uptime: 00:01:10

Thread "BenchmarkThread compiler.compiler 1": running

Address: 0x440007800000, TID: 8039, Priority: 5, Object blocked: 0 ms, Object wait: 0 ms, CPU wait: 0 ms, I/O wait: 0 ms, CPU: 61836 ms

Detail: low | high

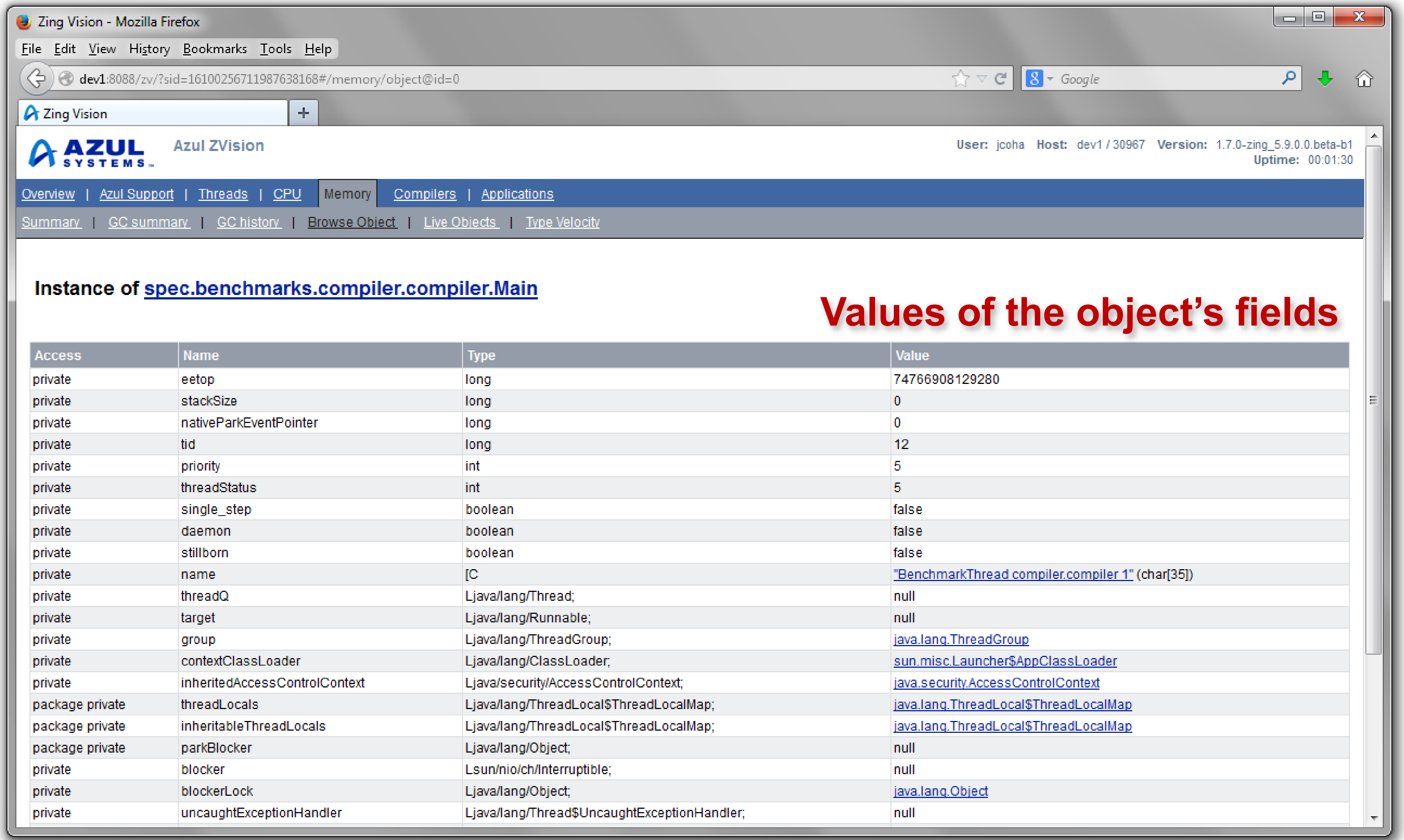
- com.sun.tools.javac.util.Name.hashCode (Name.java:191, bci=-2, compiled)
- java.util.HashMap.hash (HashMap.java:366, bci=27, compiled)
- java.util.HashMap.getEntry (HashMap.java:466, bci=19, compiled)
- java.util.HashMap.get (HashMap.java:421, bci=11, compiled)
- com.sun.tools.javac.jvm.ClassReader.loadClass (ClassReader.java:1962, bci=5, compiled)
- com.sun.tools.javac.comp.Resolve.loadClass (Resolve.java:857, bci=5, compiled)
- com.sun.tools.javac.comp.Resolve.findGlobalType (Resolve.java:916, bci=31, compiled)
- com.sun.tools.javac.comp.Resolve.findType (Resolve.java:972, bci=434, compiled)
- com.sun.tools.javac.comp.Resolve.findIdent (Resolve.java:997, bci=57, compiled)
- com.sun.tools.javac.comp.Resolve.resolveIdent (Resolve.java:1173, bci=6, compiled)
- com.sun.tools.javac.comp.Attr.visitIdent (Attr.java:1760, bci=137, compiled)
- com.sun.tools.javac.tree.JCTree$JCIdent.accept (JCTree.java:1680, bci=2, compiled)
- com.sun.tools.javac.comp.Attr.attribTree (Attr.java:373, bci=36, compiled)
- com.sun.tools.javac.comp.Attr.visitSelect (Attr.java:1845, bci=107, compiled)
- com.sun.tools.javac.tree.JCTree$JCFieldAccess.accept (JCTree.java:1652, bci=2, compiled)
- com.sun.tools.javac.comp.Attr.attribTree (Attr.java:373, bci=36, compiled)
- com.sun.tools.javac.comp.Attr.attribExpr (Attr.java:390, bci=21, compiled)
- com.sun.tools.javac.comp.Attr.visitApply (Attr.java:1288, bci=579, compiled)
- com.sun.tools.javac.tree.JCTree$JCMethodInvocation.accept (JCTree.java:1297, bci=2, compiled)
- com.sun.tools.javac.comp.Attr.attribTree (Attr.java:373, bci=36, compiled)
- com.sun.tools.javac.comp.Attr.attribExpr (Attr.java:390, bci=21, compiled)
- com.sun.tools.javac.comp.Attr.visitAssign (Attr.java:1594, bci=37, compiled)
- com.sun.tools.javac.tree.JCTree$JCAssign.accept (JCTree.java:1441, bci=2, compiled)
- com.sun.tools.javac.comp.Attr.attribTree (Attr.java:373, bci=36, compiled)
- com.sun.tools.javac.comp.Attr.attribExpr (Attr.java:397, bci=8, compiled)
- com.sun.tools.javac.comp.Attr.visitExec (Attr.java:1064, bci=9, compiled)
- com.sun.tools.javac.tree.JCTree$JCExpressionStatement.accept (JCTree.java:1143, bci=2, compiled)
- com.sun.tools.javac.comp.Attr.attribTree (Attr.java:373, bci=36, compiled)
- com.sun.tools.javac.comp.Attr.attribStat (Attr.java:410, bci=7, compiled)

Thread CPU time consumed

Stack frames

Select link to see details

Thread-level view: Thread object details



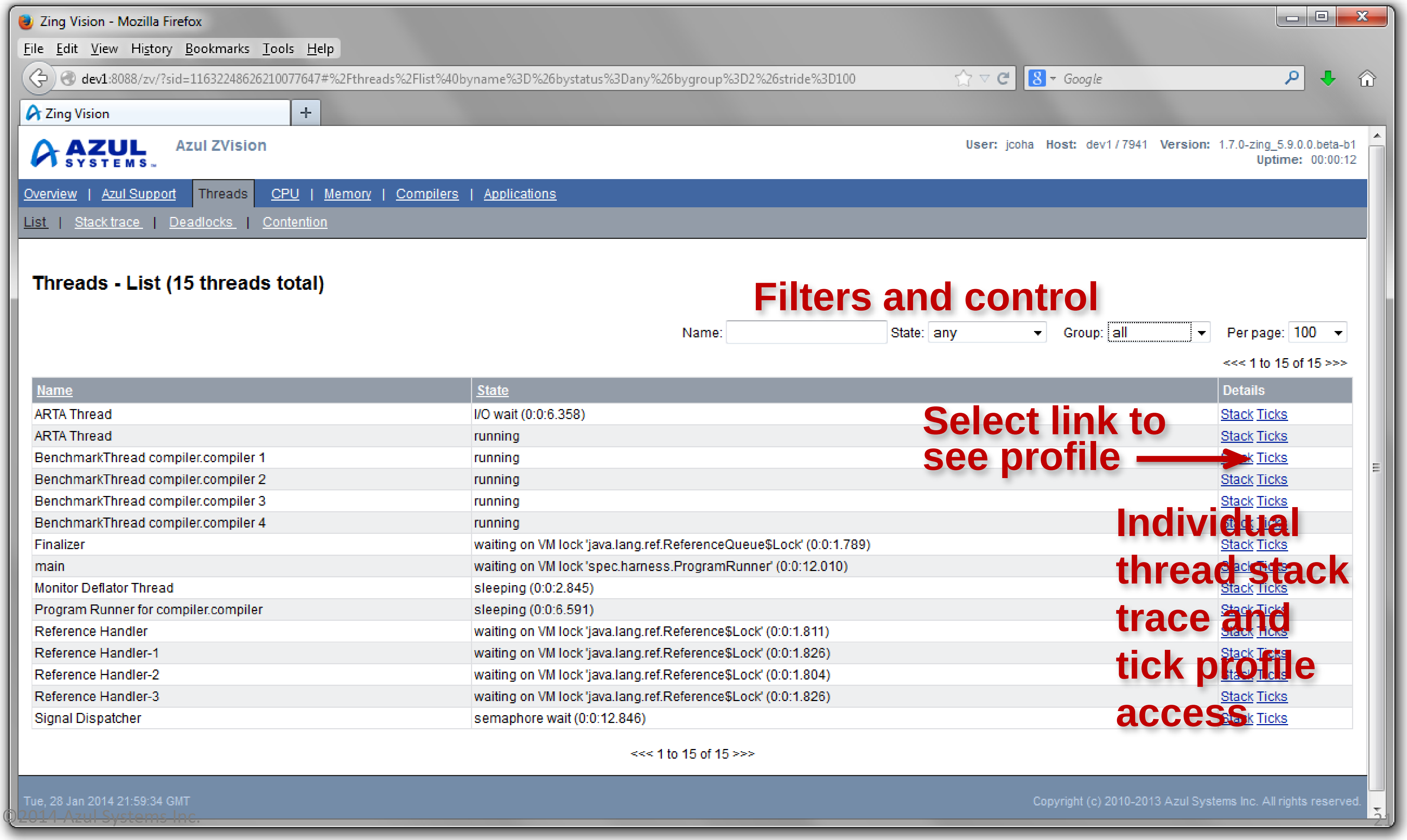
Zing Vision - Mozilla Firefox
File Edit View History Bookmarks Tools Help
dev1:8088/zv/?sid=16100256711987638168#/memory/object@id=0
Google
Zing Vision
Azul ZVision
User: jcoha Host: dev1 / 30967 Version: 1.7.0-zing_5.9.0.0.beta-b1
Uptime: 00:01:30
Overview | Azul Support | Threads | CPU | Memory | Compilers | Applications
Summary | GC summary | GC history | Browse Object | Live Objects | Type Velocity

Instance of spec.benchmarks.compiler.compiler.Main

Values of the object's fields

Access	Name	Type	Value
private	eetop	long	74766908129280
private	stackSize	long	0
private	nativeParkEventPointer	long	0
private	tid	long	12
private	priority	int	5
private	threadStatus	int	5
private	single_step	boolean	false
private	daemon	boolean	false
private	stillborn	boolean	false
private	name	[C	"BenchmarkThread compiler.compiler 1" (char[35])
private	threadQ	Ljava/lang/Thread;	null
private	target	Ljava/lang/Runnable;	null
private	group	Ljava/lang/ThreadGroup;	java.lang.ThreadGroup
private	contextClassLoader	Ljava/lang/ClassLoader;	sun.misc.Launcher$AppClassLoader
private	inheritedAccessControlContext	Ljava/security/AccessControlContext;	java.security.AccessControlContext
package private	threadLocals	Ljava/lang/ThreadLocal$ThreadLocalMap;	java.lang.ThreadLocal$ThreadLocalMap
package private	inheritableThreadLocals	Ljava/lang/ThreadLocal$ThreadLocalMap;	java.lang.ThreadLocal$ThreadLocalMap
package private	parkBlocker	Ljava/lang/Object;	null
private	blocker	Lsun/nio/ch/Interruptible;	null
private	blockerLock	Ljava/lang/Object;	java.lang.Object
private	uncaughtExceptionHandler	Ljava/lang/Thread$UncaughtExceptionHandler;	null

Thread-level view: Application threads



Zing Vision - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Google
Zing Vision
+

dev1:8088/zv/?sid=11632248626210077647#%2Fthreads%2Flist%40byname%3D%26bystatus%3Dany%26bygroup%3D2%26stride%3D100

Azul ZVision

User: jcoha Host: dev1 / 7941 Version: 1.7.0-zing_5.9.0.0.beta-b1
Uptime: 00:00:12

Overview | Azul Support | Threads | CPU | Memory | Compilers | Applications
List | Stack trace | Deadlocks | Contention

Threads - List (15 threads total)

Filters and control

Name: State: any Group: all Per page: 100

<<< 1 to 15 of 15 >>>

Name	State	Details
ARTA Thread	I/O wait (0:0:6.358)	Stack Ticks
ARTA Thread	running	Stack Ticks
BenchmarkThread compiler.compiler 1	running	Stack Ticks
BenchmarkThread compiler.compiler 2	running	Stack Ticks
BenchmarkThread compiler.compiler 3	running	Stack Ticks
BenchmarkThread compiler.compiler 4	running	Stack Ticks
Finalizer	waiting on VM lock 'java.lang.ref.ReferenceQueue$Lock' (0:0:1.789)	Stack Ticks
main	waiting on VM lock 'spec.harness.ProgramRunner' (0:0:12.010)	Stack Ticks
Monitor Deflator Thread	sleeping (0:0:2.845)	Stack Ticks
Program Runner for compiler.compiler	sleeping (0:0:6.591)	Stack Ticks
Reference Handler	waiting on VM lock 'java.lang.ref.Reference$Lock' (0:0:1.811)	Stack Ticks
Reference Handler-1	waiting on VM lock 'java.lang.ref.Reference$Lock' (0:0:1.826)	Stack Ticks
Reference Handler-2	waiting on VM lock 'java.lang.ref.Reference$Lock' (0:0:1.804)	Stack Ticks
Reference Handler-3	waiting on VM lock 'java.lang.ref.Reference$Lock' (0:0:1.826)	Stack Ticks
Signal Dispatcher	semaphore wait (0:0:12.846)	Stack Ticks

<<< 1 to 15 of 15 >>>

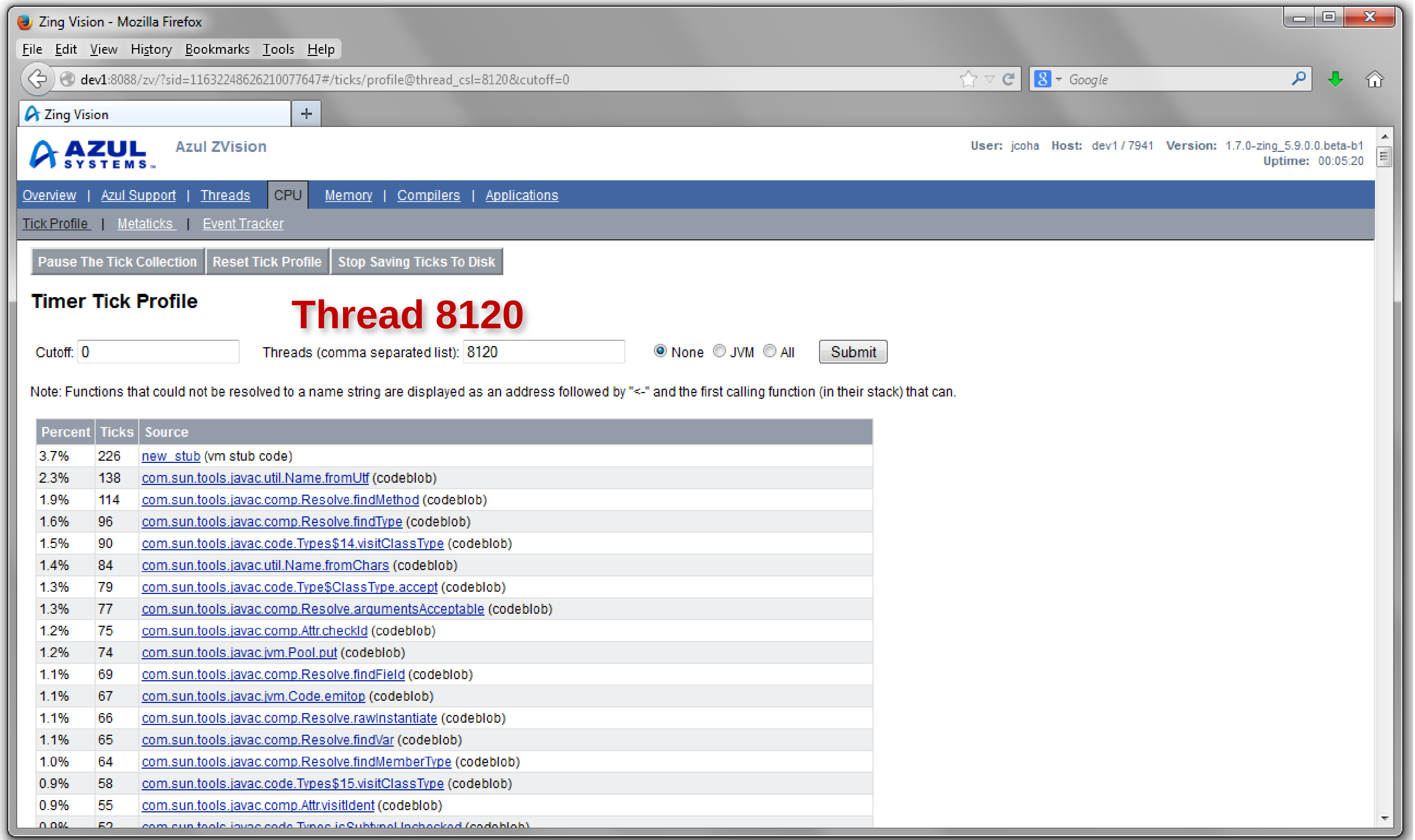
Tue, 28 Jan 2014 21:59:34 GMT

Copyright (c) 2010-2013 Azul Systems Inc. All rights reserved.

Select link to see profile

Individual thread stack trace and tick profile access

Thread-level: Where is the work done?


Zing Vision - Mozilla Firefox
File Edit View History Bookmarks Tools Help
dev1:8088/zv/?sid=1163224862621007647#/ticks/profile@thread_csl=8120&cutoff=0
Google
Zing Vision
Azul ZVision
Overview | Azul Support | Threads | CPU | Memory | Compilers | Applications
Tick Profile | Metaticks | Event Tracker
Pause The Tick Collection
Reset Tick Profile
Stop Saving Ticks To Disk
Submit

User: jcoha Host: dev1 / 7941 Version: 1.7.0-zing_5.9.0.0.beta-b1
Uptime: 00:05:20

Timer Tick Profile

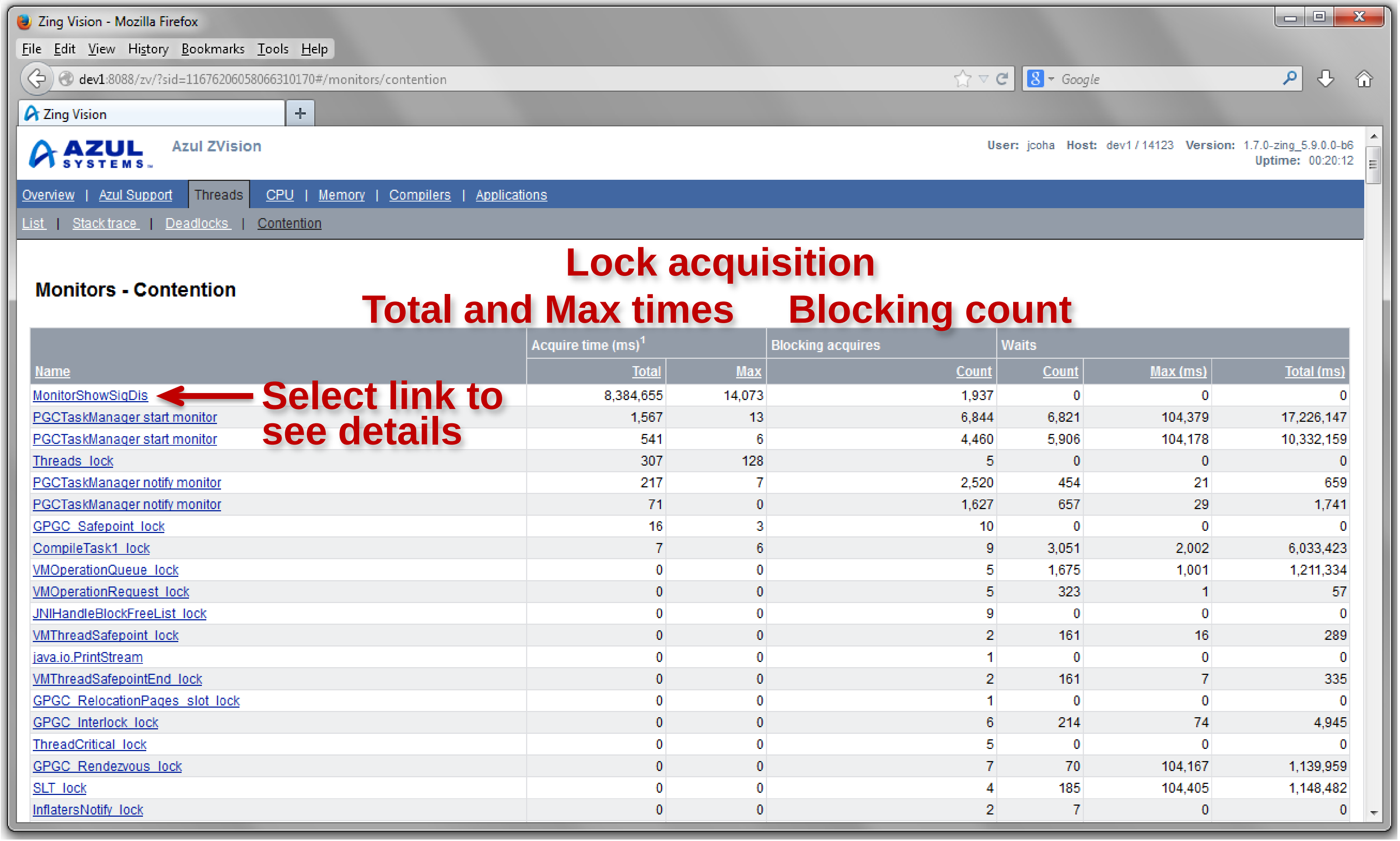
Thread 8120

Cutoff: 0 Threads (comma separated list): 8120 ● None ○ JVM ○ All

Note: Functions that could not be resolved to a name string are displayed as an address followed by "<-" and the first calling function (in their stack) that can.

Percent	Ticks	Source
3.7%	226	new_stub (vm stub code)
2.3%	138	com.sun.tools.javac.util.Name.fromUtf (codeblob)
1.9%	114	com.sun.tools.javac.comp.Resolve.findMethod (codeblob)
1.6%	96	com.sun.tools.javac.comp.Resolve.findType (codeblob)
1.5%	90	com.sun.tools.javac.code.Types$14.visitClassType (codeblob)
1.4%	84	com.sun.tools.javac.util.Name.fromChars (codeblob)
1.3%	79	com.sun.tools.javac.code.Type$ClassType.accept (codeblob)
1.3%	77	com.sun.tools.javac.comp.Resolve.argumentsAcceptable (codeblob)
1.2%	75	com.sun.tools.javac.comp.Attr.checkId (codeblob)
1.2%	74	com.sun.tools.javac.jvm.Pool.put (codeblob)
1.1%	69	com.sun.tools.javac.comp.Resolve.findField (codeblob)
1.1%	67	com.sun.tools.javac.jvm.Code.emitop (codeblob)
1.1%	66	com.sun.tools.javac.comp.Resolve.rawInstantiate (codeblob)
1.1%	65	com.sun.tools.javac.comp.Resolve.findVar (codeblob)
1.0%	64	com.sun.tools.javac.comp.Resolve.findMemberType (codeblob)
0.9%	58	com.sun.tools.javac.code.Types$15.visitClassType (codeblob)
0.9%	55	com.sun.tools.javac.comp.Attr.visitIdent (codeblob)
0.9%	52	com.sun.tools.javac.code.Types.isSubtypeUnchecked (codeblob)

Lock contention: Where is my app stalled?


Zing Vision - Mozilla Firefox
File Edit View History Bookmarks Tools Help
dev1:8088/zv/?sid=11676206058066310170#/monitors/contention
Google
Zing Vision
+
AZUL SYSTEMS
Azul ZVision
Overview | Azul Support | Threads | CPU | Memory | Compilers | Applications
List | Stack trace | Deadlocks | Contention

User: jcoha Host: dev1 / 14123 Version: 1.7.0-zing_5.9.0.0-b6
Uptime: 00:20:12

Lock acquisition

Total and Max times

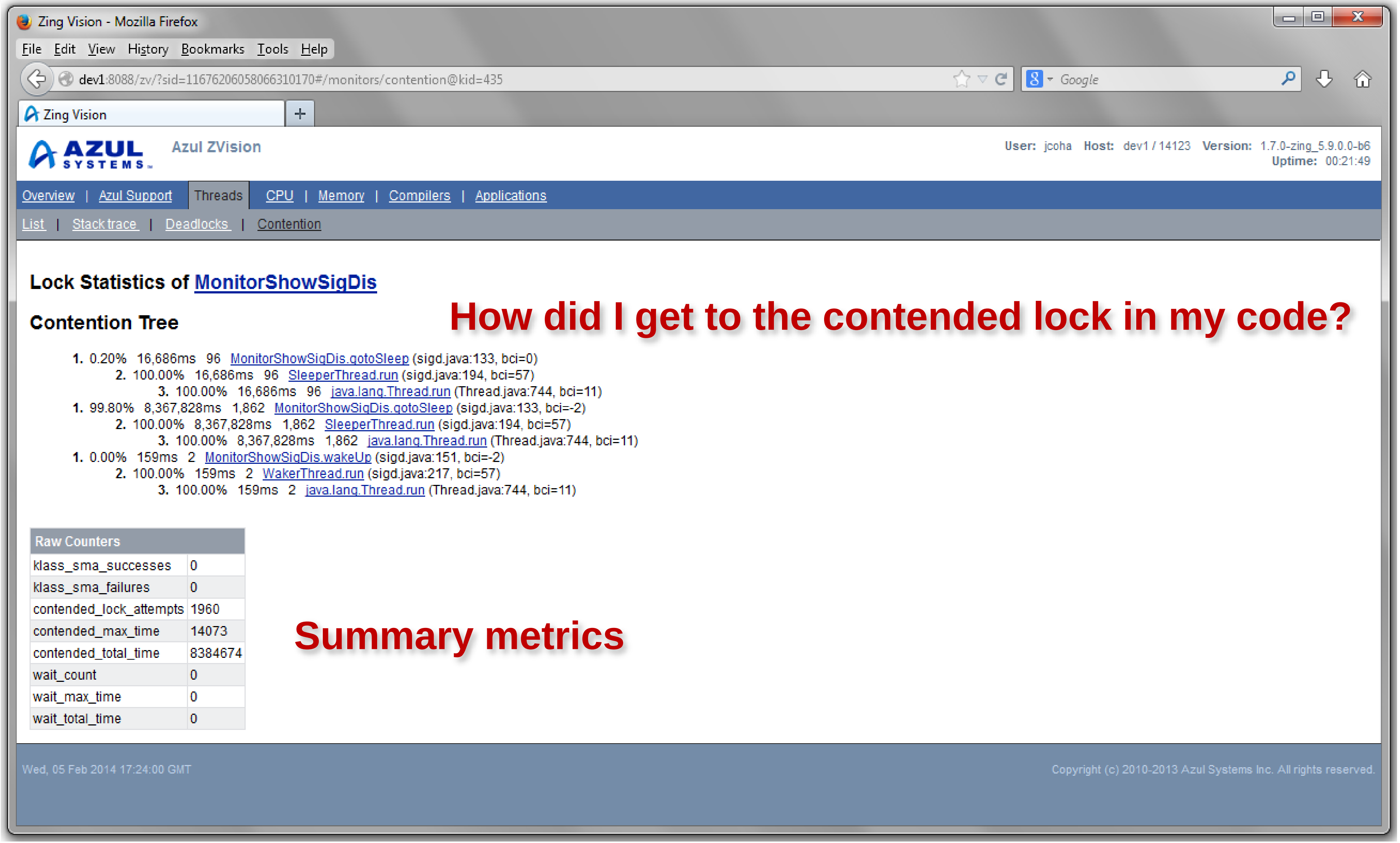
Blocking count

Select link to see details

Monitors - Contention

	Acquire time (ms)[1]		Blocking acquires	Waits		
Name	Total	Max	Count	Count	Max (ms)	Total (ms)
MonitorShowSigDis	8,384,655	14,073	1,937	0	0	0
PGCTaskManager start monitor	1,567	13	6,844	6,821	104,379	17,226,147
PGCTaskManager start monitor	541	6	4,460	5,906	104,178	10,332,159
Threads_lock	307	128	5	0	0	0
PGCTaskManager notify monitor	217	7	2,520	454	21	659
PGCTaskManager notify monitor	71	0	1,627	657	29	1,741
GPGC_Safepoint_lock	16	3	10	0	0	0
CompileTask1_lock	7	6	9	3,051	2,002	6,033,423
VMOperationQueue_lock	0	0	5	1,675	1,001	1,211,334
VMOperationRequest_lock	0	0	5	323	1	57
JNIHandleBlockFreeList_lock	0	0	9	0	0	0
VMThreadSafepoint_lock	0	0	2	161	16	289
java.io.PrintStream	0	0	1	0	0	0
VMThreadSafepointEnd_lock	0	0	2	161	7	335
GPGC_RelocationPages_slot_lock	0	0	1	0	0	0
GPGC_Interlock_lock	0	0	6	214	74	4,945
ThreadCritical_lock	0	0	5	0	0	0
GPGC_Rendezvous_lock	0	0	7	70	104,167	1,139,959
SLT_lock	0	0	4	185	104,405	1,148,482
InflatersNotify_lock	0	0	2	7	0	0

Lock contention: Where is my app stalled?



Zing Vision - Mozilla Firefox
File Edit View History Bookmarks Tools Help
dev1:8088/zv/?sid=1167620605806631017O#/monitors/contention@kid=435
Google
Zing Vision
AZUL SYSTEMS
Azul ZVision
User: jcoha Host: dev1 / 14123 Version: 1.7.0-zing_5.9.0.0-b6
Uptime: 00:21:49
Overview | Azul Support | Threads | CPU | Memory | Compilers | Applications
List | Stack trace | Deadlocks | Contention

Lock Statistics of MonitorShowSigDis

Contention Tree

How did I get to the contended lock in my code?

```
1. 0.20%  16,686ms  96  MonitorShowSigDis.gotoSleep (sigd.java:133, bci=0)
    2. 100.00%  16,686ms  96  SleeperThread.run (sigd.java:194, bci=57)
        3. 100.00%  16,686ms  96  java.lang.Thread.run (Thread.java:744, bci=11)
1. 99.80%  8,367,828ms  1,862  MonitorShowSigDis.gotoSleep (sigd.java:133, bci=-2)
    2. 100.00%  8,367,828ms  1,862  SleeperThread.run (sigd.java:194, bci=57)
        3. 100.00%  8,367,828ms  1,862  java.lang.Thread.run (Thread.java:744, bci=11)
1. 0.00%  159ms  2  MonitorShowSigDis.wakeUp (sigd.java:151, bci=-2)
    2. 100.00%  159ms  2  WakerThread.run (sigd.java:217, bci=57)
        3. 100.00%  159ms  2  java.lang.Thread.run (Thread.java:744, bci=11)
```

Raw Counters	
klass_sma_successes	0
klass_sma_failures	0
contended_lock_attempts	1960
contended_max_time	14073
contended_total_time	8384674
wait_count	0
wait_max_time	0
wait_total_time	0

Summary metrics

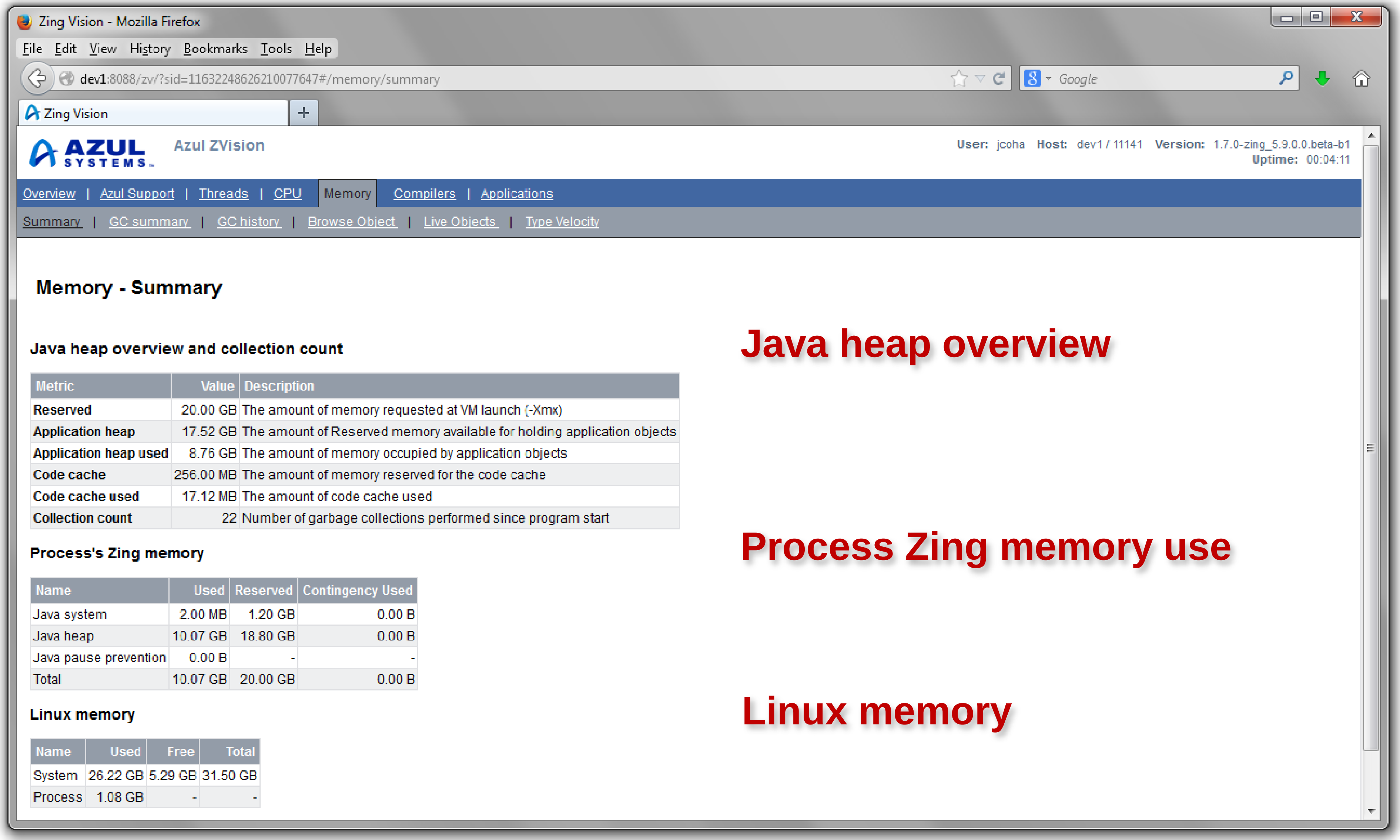
Wed, 05 Feb 2014 17:24:00 GMT

Copyright (c) 2010-2013 Azul Systems Inc. All rights reserved.

Memory: Answering your questions

- What's the collector doing?
- How much memory is the application using?
- What type of objects are in the heap?
  - What's keeping those objects live?
- What object types are increasing in number?

Memory: Resource use summary


Zing Vision - Mozilla Firefox
File Edit View History Bookmarks Tools Help
dev1:8088/zv/?sid=1163224862621007647#/memory/summary
Google
Zing Vision
AZUL SYSTEMS
Azul ZVision
User: jcoha Host: dev1 / 11141 Version: 1.7.0-zing_5.9.0.0.beta-b1
Uptime: 00:04:11
Overview | Azul Support | Threads | CPU | Memory | Compilers | Applications
Summary | GC summary | GC history | Browse Object | Live Objects | Type Velocity

Memory - Summary

Java heap overview and collection count

Metric	Value	Description
Reserved	20.00 GB	The amount of memory requested at VM launch (-Xmx)
Application heap	17.52 GB	The amount of Reserved memory available for holding application objects
Application heap used	8.76 GB	The amount of memory occupied by application objects
Code cache	256.00 MB	The amount of memory reserved for the code cache
Code cache used	17.12 MB	The amount of code cache used
Collection count	22	Number of garbage collections performed since program start

Process's Zing memory

Name	Used	Reserved	Contingency Used
Java system	2.00 MB	1.20 GB	0.00 B
Java heap	10.07 GB	18.80 GB	0.00 B
Java pause prevention	0.00 B	-	-
Total	10.07 GB	20.00 GB	0.00 B

Linux memory

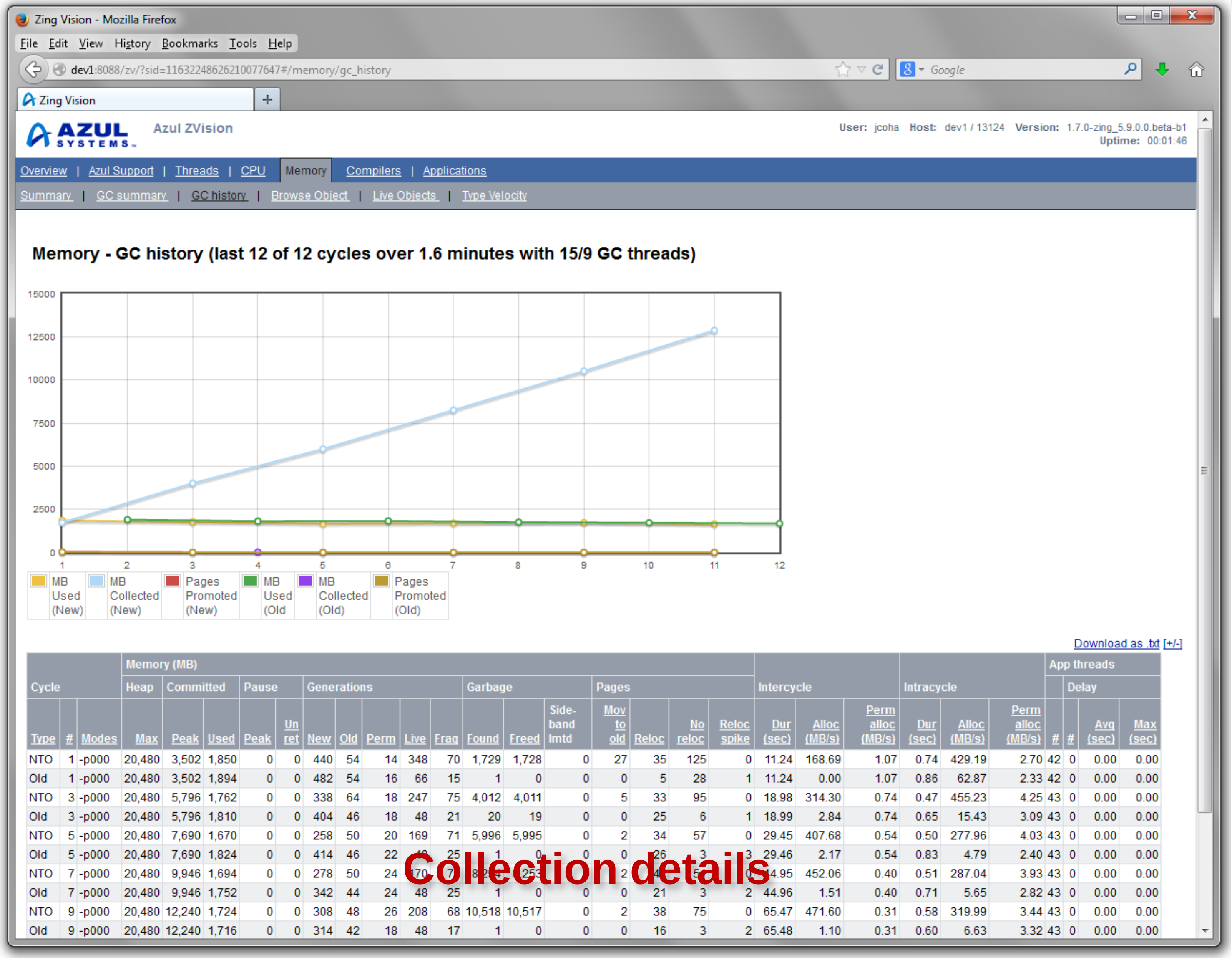
Name	Used	Free	Total
System	26.22 GB	5.29 GB	31.50 GB
Process	1.08 GB	-	-

Java heap overview

Process Zing memory use

Linux memory

Memory: Collection details



Zing Vision - Mozilla Firefox
File Edit View History Bookmarks Tools Help
dev1:8088/zv/?sid=11632248626210077647#/memory/gc_history
Google
Zing Vision
+
AZUL SYSTEMS
Azul ZVision
User: jcoha Host: dev1 / 13124 Version: 1.7.0-zing_5.9.0.0.beta-b1
Uptime: 00:01:46
Overview | Azul Support | Threads | CPU | Memory | Compilers | Applications
Summary | GC summary | GC history | Browse Object | Live Objects | Type Velocity

Memory - GC history (last 12 of 12 cycles over 1.6 minutes with 15/9 GC threads)

15000
12500
10000
7500
5000
2500
0
1
2
3
4
5
6
7
8
9
10
11
12
MB Used (New)
MB Collected (New)
Pages Promoted (New)
MB Used (Old
MB Collected (Old)
Pages Promoted (Old)

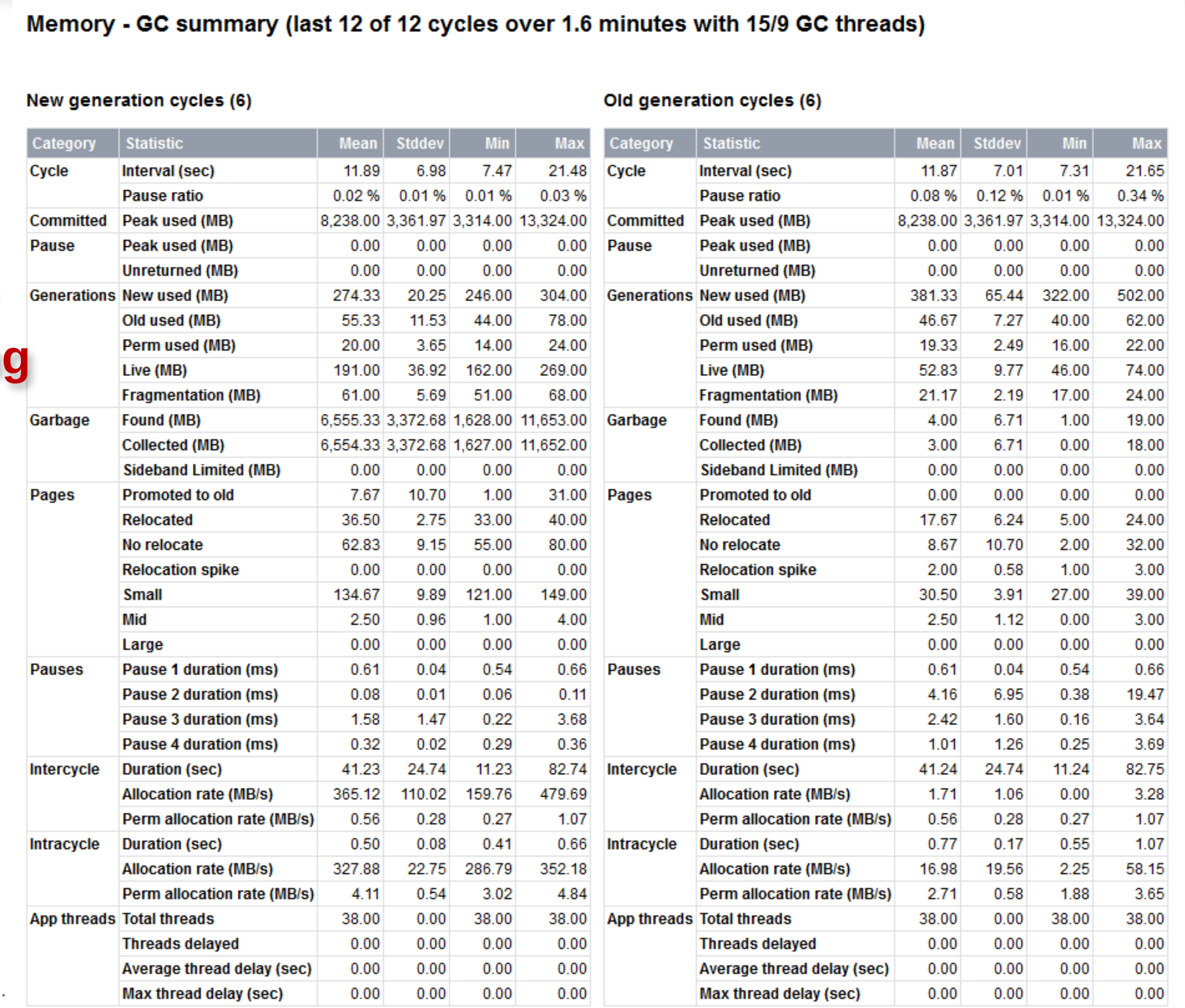
Download as .txt [+/-]

Cycle			Memory (MB)																									App threads			
			Heap	Committed		Pause		Generations					Garbage			Pages					Intercycle			Intracycle				Delay			
Type	#	Modes	Max	Peak	Used	Peak	Un ret	New	Old	Perm	Live	Frag	Found	Freed	Side-band lmtd	Mov to old	Reloc	No reloc	Reloc spike	Dur (sec)	Alloc (MB/s)	Perm alloc (MB/s)	Dur (sec)	Alloc (MB/s)	Perm alloc (MB/s)	#	#	Avg (sec)	Max (sec)
NTO	1	-p000	20,480	3,502	1,850	0	0	440	54	14	348	70	1,729	1,728	0	27	35	125	0	11.24	168.69	1.07	0.74	429.19	2.70	42	0	0.00	0.00
Old	1	-p000	20,480	3,502	1,894	0	0	482	54	16	66	15	1	0	0	0	5	28	1	11.24	0.00	1.07	0.86	62.87	2.33	42	0	0.00	0.00
NTO	3	-p000	20,480	5,796	1,762	0	0	338	64	18	247	75	4,012	4,011	0	5	33	95	0	18.98	314.30	0.74	0.47	455.23	4.25	43	0	0.00	0.00
Old	3	-p000	20,480	5,796	1,810	0	0	404	46	18	48	21	20	19	0	0	25	6	1	18.99	2.84	0.74	0.65	15.43	3.09	43	0	0.00	0.00
NTO	5	-p000	20,480	7,690	1,670	0	0	258	50	20	169	71	5,996	5,995	0	2	34	57	0	29.45	407.68	0.54	0.50	277.96	4.03	43	0	0.00	0.00
Old	5	-p000	20,480	7,690	1,824	0	0	414	46	22	[illegible]	25	1	0	0	0	26	3	3	29.46	2.17	0.54	0.83	4.79	2.40	43	0	0.00	0.00
NTO	7	-p000	20,480	9,946	1,694	0	0	278	50	24	170	[illegible]	[illegible]	[illegible]	0	2	[illegible]	[illegible]	0	[illegible]	452.06	0.40	0.51	287.04	3.93	43	0	0.00	0.00
Old	7	-p000	20,480	9,946	1,752	0	0	342	44	24	48	25	1	0	0	0	21	3	2	44.96	1.51	0.40	0.71	5.65	2.82	43	0	0.00	0.00
NTO	9	-p000	20,480	12,240	1,724	0	0	308	48	26	208	68	10,518	10,517	0	2	38	75	0	65.47	471.60	0.31	0.58	319.99	3.44	43	0	0.00	0.00
Old	9	-p000	20,480	12,240	1,716	0	0	314	42	18	48	17	1	0	0	0	16	3	2	65.48	1.10	0.31	0.60	6.63	3.32	43	0	0.00	0.00

Collection details

Memory: Collection details summary

Detailed summary calculating metric averages



Memory - GC summary (last 12 of 12 cycles over 1.6 minutes with 15/9 GC threads)

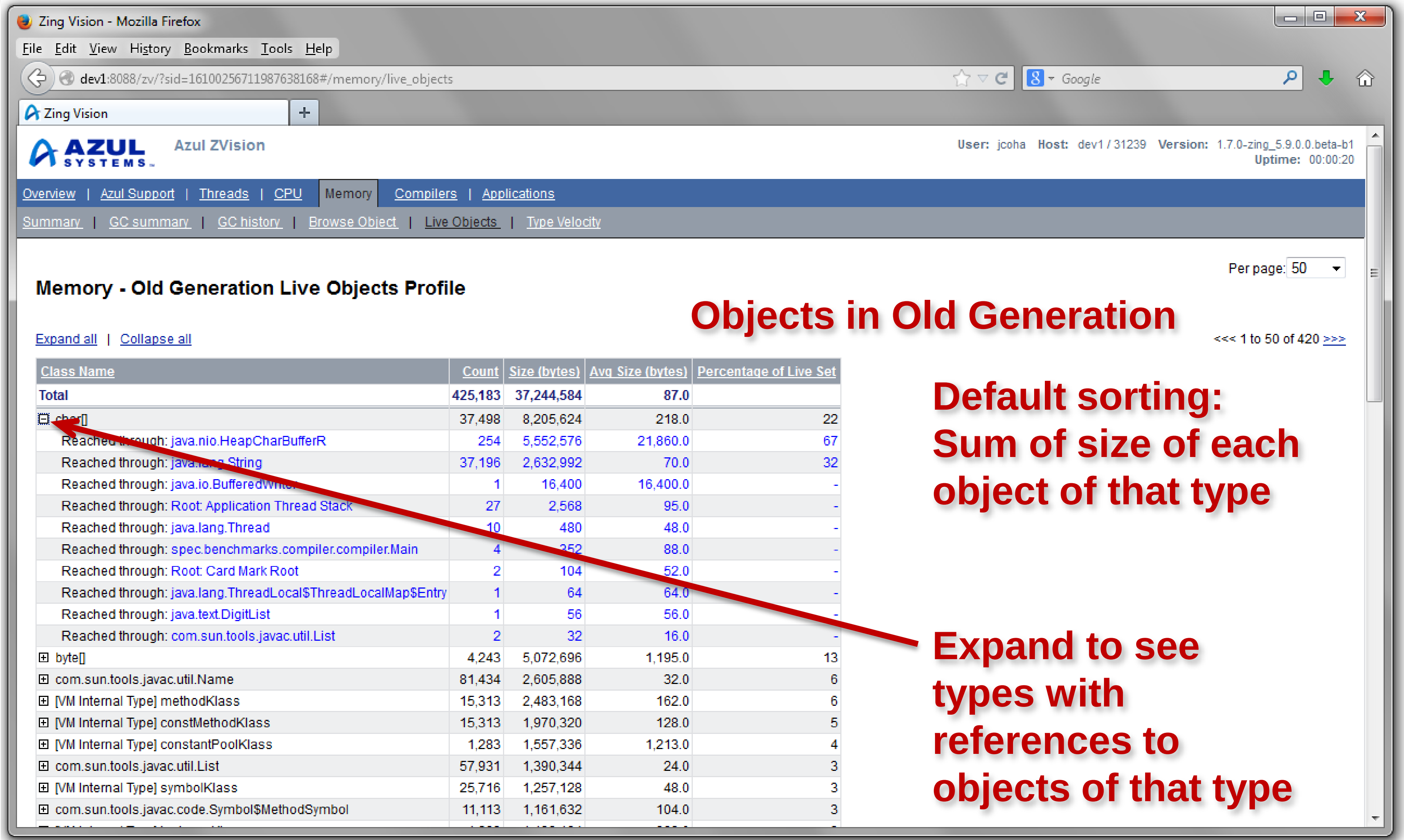
New generation cycles (6)

Category	Statistic	Mean	Stddev	Min	Max
Cycle	Interval (sec)	11.89	6.98	7.47	21.48
	Pause ratio	0.02 %	0.01 %	0.01 %	0.03 %
Committed	Peak used (MB)	8,238.00	3,361.97	3,314.00	13,324.00
Pause	Peak used (MB)	0.00	0.00	0.00	0.00
	Unreturned (MB)	0.00	0.00	0.00	0.00
Generations	New used (MB)	274.33	20.25	246.00	304.00
	Old used (MB)	55.33	11.53	44.00	78.00
	Perm used (MB)	20.00	3.65	14.00	24.00
	Live (MB)	191.00	36.92	162.00	269.00
	Fragmentation (MB)	61.00	5.69	51.00	68.00
Garbage	Found (MB)	6,555.33	3,372.68	1,628.00	11,653.00
	Collected (MB)	6,554.33	3,372.68	1,627.00	11,652.00
	Sideband Limited (MB)	0.00	0.00	0.00	0.00
Pages	Promoted to old	7.67	10.70	1.00	31.00
	Relocated	36.50	2.75	33.00	40.00
	No relocate	62.83	9.15	55.00	80.00
	Relocation spike	0.00	0.00	0.00	0.00
	Small	134.67	9.89	121.00	149.00
	Mid	2.50	0.96	1.00	4.00
	Large	0.00	0.00	0.00	0.00
Pauses	Pause 1 duration (ms)	0.61	0.04	0.54	0.66
	Pause 2 duration (ms)	0.08	0.01	0.06	0.11
	Pause 3 duration (ms)	1.58	1.47	0.22	3.68
	Pause 4 duration (ms)	0.32	0.02	0.29	0.36
Intercycle	Duration (sec)	41.23	24.74	11.23	82.74
	Allocation rate (MB/s)	365.12	110.02	159.76	479.69
	Perm allocation rate (MB/s)	0.56	0.28	0.27	1.07
Intracycle	Duration (sec)	0.50	0.08	0.41	0.66
	Allocation rate (MB/s)	327.88	22.75	286.79	352.18
	Perm allocation rate (MB/s)	4.11	0.54	3.02	4.84
App threads	Total threads	38.00	0.00	38.00	38.00
	Threads delayed	0.00	0.00	0.00	0.00
	Average thread delay (sec)	0.00	0.00	0.00	0.00
	Max thread delay (sec)	0.00	0.00	0.00	0.00

Old generation cycles (6)

Category	Statistic	Mean	Stddev	Min	Max
Cycle	Interval (sec)	11.87	7.01	7.31	21.65
	Pause ratio	0.08 %	0.12 %	0.01 %	0.34 %
Committed	Peak used (MB)	8,238.00	3,361.97	3,314.00	13,324.00
Pause	Peak used (MB)	0.00	0.00	0.00	0.00
	Unreturned (MB)	0.00	0.00	0.00	0.00
Generations	New used (MB)	381.33	65.44	322.00	502.00
	Old used (MB)	46.67	7.27	40.00	62.00
	Perm used (MB)	19.33	2.49	16.00	22.00
	Live (MB)	52.83	9.77	46.00	74.00
	Fragmentation (MB)	21.17	2.19	17.00	24.00
Garbage	Found (MB)	4.00	6.71	1.00	19.00
	Collected (MB)	3.00	6.71	0.00	18.00
	Sideband Limited (MB)	0.00	0.00	0.00	0.00
Pages	Promoted to old	0.00	0.00	0.00	0.00
	Relocated	17.67	6.24	5.00	24.00
	No relocate	8.67	10.70	2.00	32.00
	Relocation spike	2.00	0.58	1.00	3.00
	Small	30.50	3.91	27.00	39.00
	Mid	2.50	1.12	0.00	3.00
	Large	0.00	0.00	0.00	0.00
Pauses	Pause 1 duration (ms)	0.61	0.04	0.54	0.66
	Pause 2 duration (ms)	4.16	6.95	0.38	19.47
	Pause 3 duration (ms)	2.42	1.60	0.16	3.64
	Pause 4 duration (ms)	1.01	1.26	0.25	3.69
Intercycle	Duration (sec)	41.24	24.74	11.24	82.75
	Allocation rate (MB/s)	1.71	1.06	0.00	3.28
	Perm allocation rate (MB/s)	0.56	0.28	0.27	1.07
Intracycle	Duration (sec)	0.77	0.17	0.55	1.07
	Allocation rate (MB/s)	16.98	19.56	2.25	58.15
	Perm allocation rate (MB/s)	2.71	0.58	1.88	3.65
App threads	Total threads	38.00	0.00	38.00	38.00
	Threads delayed	0.00	0.00	0.00	0.00
	Average thread delay (sec)	0.00	0.00	0.00	0.00
	Max thread delay (sec)	0.00	0.00	0.00	0.00

Memory: What objects are in the heap?



Zing Vision - Mozilla Firefox
File Edit View History Bookmarks Tools Help
dev1:8088/zv/?sid=16100256711987638168#/memory/live_objects
Google
Zing Vision
AZUL SYSTEMS
Azul ZVision

User: jcoha Host: dev1 / 31239 Version: 1.7.0-zing_5.9.0.0.beta-b1
Uptime: 00:00:20

Overview | Azul Support | Threads | CPU | Memory | Compilers | Applications
Summary | GC summary | GC history | Browse Object | Live Objects | Type Velocity

Per page: 50

Memory - Old Generation Live Objects Profile

Expand all | Collapse all

<<< 1 to 50 of 420 >>>

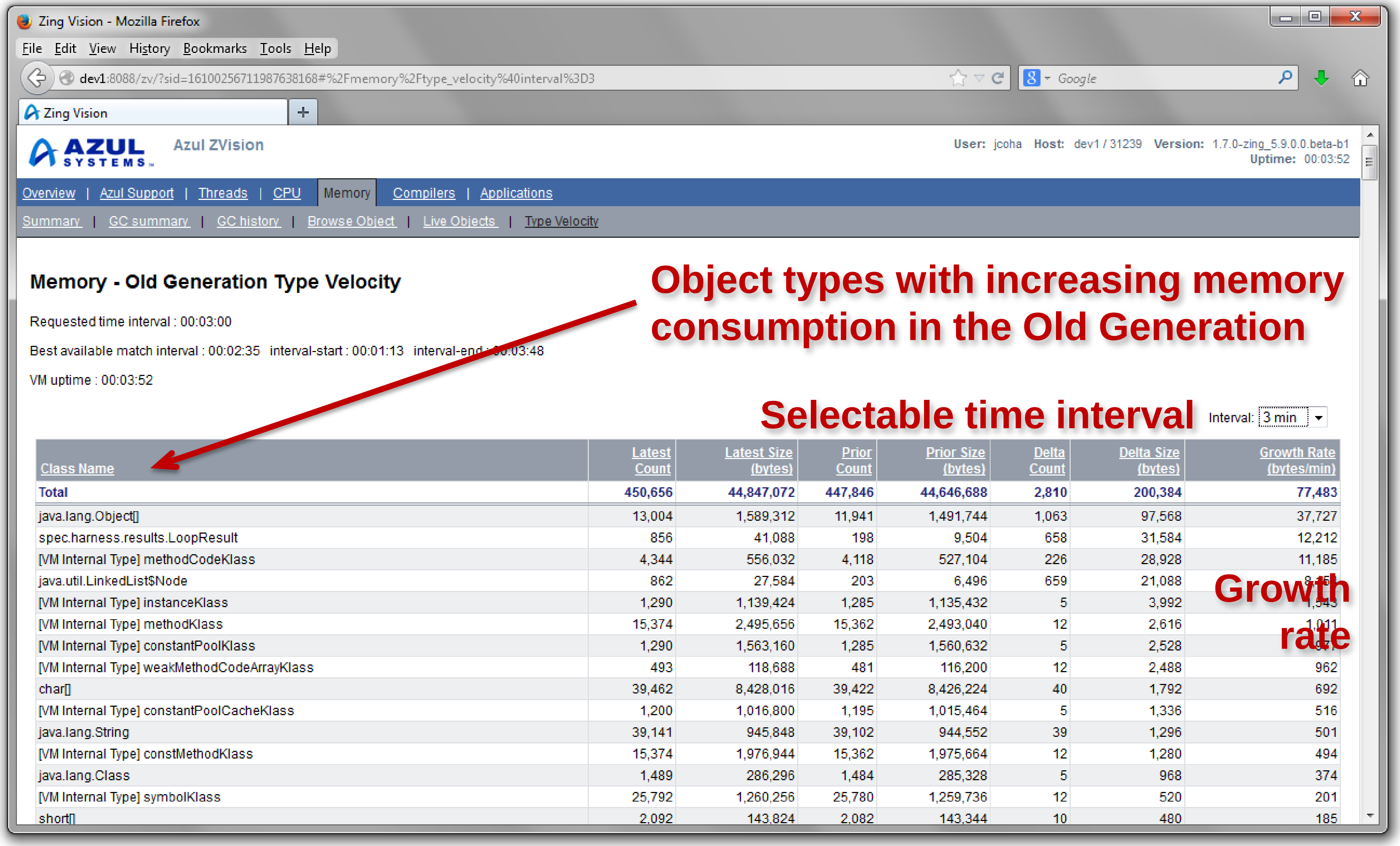
Class Name	Count	Size (bytes)	Avg Size (bytes)	Percentage of Live Set
Total	425,183	37,244,584	87.0	
⊟ char[]	37,498	8,205,624	218.0	22
Reached through: java.nio.HeapCharBufferR	254	5,552,576	21,860.0	67
Reached through: java.lang.String	37,196	2,632,992	70.0	32
Reached through: java.io.BufferedWriter	1	16,400	16,400.0	-
Reached through: Root: Application Thread Stack	27	2,568	95.0	-
Reached through: java.lang.Thread	10	480	48.0	-
Reached through: spec.benchmarks.compiler.compiler.Main	4	352	88.0	-
Reached through: Root: Card Mark Root	2	104	52.0	-
Reached through: java.lang.ThreadLocal$ThreadLocalMap$Entry	1	64	64.0	-
Reached through: java.text.DigitList	1	56	56.0	-
Reached through: com.sun.tools.javac.util.List	2	32	16.0	-
⊞ byte[]	4,243	5,072,696	1,195.0	13
⊞ com.sun.tools.javac.util.Name	81,434	2,605,888	32.0	6
⊞ [VM Internal Type] methodKlass	15,313	2,483,168	162.0	6
⊞ [VM Internal Type] constMethodKlass	15,313	1,970,320	128.0	5
⊞ [VM Internal Type] constantPoolKlass	1,283	1,557,336	1,213.0	4
⊞ com.sun.tools.javac.util.List	57,931	1,390,344	24.0	3
⊞ [VM Internal Type] symbolKlass	25,716	1,257,128	48.0	3
⊞ com.sun.tools.javac.code.Symbol$MethodSymbol	11,113	1,161,632	104.0	3

Objects in Old Generation

Default sorting: Sum of size of each object of that type

Expand to see types with references to objects of that type

Memory: Which object type is growing?



Zing Vision - Mozilla Firefox
File Edit View History Bookmarks Tools Help
dev1:8088/zv/?sid=16100256711987638168#%2Fmemory%2Ftype_velocity%40interval%3D3
Google
Zing Vision
+
AZUL SYSTEMS
Azul ZVision
Overview | Azul Support | Threads | CPU | Memory | Compilers | Applications
Summary | GC summary | GC history | Browse Object | Live Objects | Type Velocity

User: jcoha   Host: dev1 / 31239   Version: 1.7.0-zing_5.9.0.0.beta-b1
Uptime: 00:03:52

Memory - Old Generation Type Velocity

Requested time interval : 00:03:00

Best available match interval : 00:02:35   interval-start : 00:01:13   interval-end : 00:03:48

VM uptime : 00:03:52

Interval: 3 min

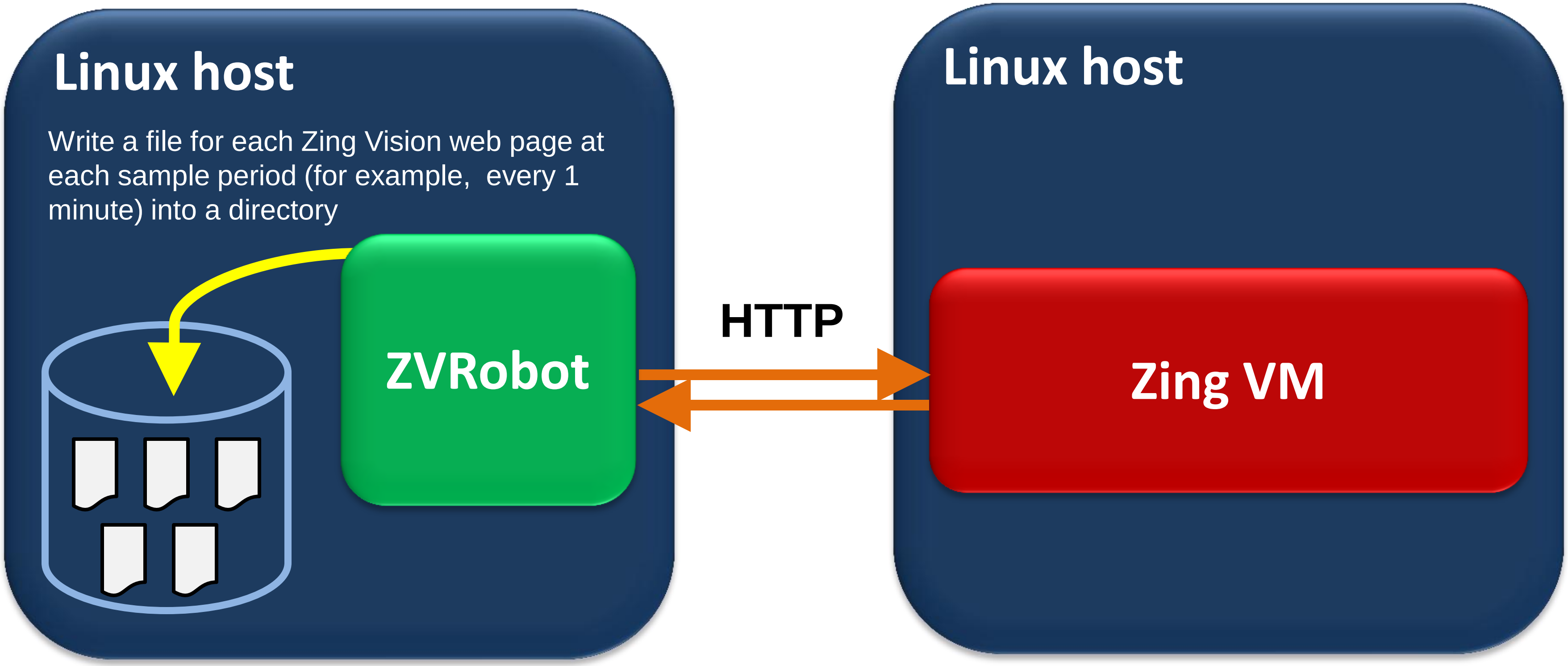
Class Name	Latest Count	Latest Size (bytes)	Prior Count	Prior Size (bytes)	Delta Count	Delta Size (bytes)	Growth Rate (bytes/min)
Total	**450,656**	**44,847,072**	**447,846**	**44,646,688**	**2,810**	**200,384**	**77,483**
java.lang.Object[]	13,004	1,589,312	11,941	1,491,744	1,063	97,568	37,727
spec.harness.results.LoopResult	856	41,088	198	9,504	658	31,584	12,212
[VM Internal Type] methodCodeKlass	4,344	556,032	4,118	527,104	226	28,928	11,185
java.util.LinkedList$Node	862	27,584	203	6,496	659	21,088	8,154
[VM Internal Type] instanceKlass	1,290	1,139,424	1,285	1,135,432	5	3,992	1,543
[VM Internal Type] methodKlass	15,374	2,495,656	15,362	2,493,040	12	2,616	1,011
[VM Internal Type] constantPoolKlass	1,290	1,563,160	1,285	1,560,632	5	2,528	977
[VM Internal Type] weakMethodCodeArrayKlass	493	118,688	481	116,200	12	2,488	962
char[]	39,462	8,428,016	39,422	8,426,224	40	1,792	692
[VM Internal Type] constantPoolCacheKlass	1,200	1,016,800	1,195	1,015,464	5	1,336	516
java.lang.String	39,141	945,848	39,102	944,552	39	1,296	501
[VM Internal Type] constMethodKlass	15,374	1,976,944	15,362	1,975,664	12	1,280	494
java.lang.Class	1,489	286,296	1,484	285,328	5	968	374
[VM Internal Type] symbolKlass	25,792	1,260,256	25,780	1,259,736	12	520	201
short[]	2,092	143,824	2,082	143,344	10	480	185

Object types with increasing memory consumption in the Old Generation

Selectable time interval

Growth rate

Example ZVRobot deployment



Write a file for each Zing Vision web page at each sample period (for example, every 1 minute) into a directory

Use a web browser to look at the saved snapshot files anytime after program has run or even during collection

How to Try Zing Vision - Free

- Request a trial copy of Zing http://www.azulsystems.com/trial
- Download and run Azul Inspector to check your system
- Download and install Zing
  - Zing Vision and ZVRobot are included
- Run your application
- Observe your application and the JVM's activity using Zing Vision

Questions?

Zing Vision – providing answers to your Java performance questions

- Tick profiler
- Thread-level views
- Lock contention view
- Garbage collection views
- Java heap object views
- Live objects
- Object types that increase in number as application runs