



Azul Virtual Machine Guide

510-00256

Version 11

Azul Software Compatibility

This product guide is compatible with Azul Virtual Machine distribution 2.5. For more information on obtaining other versions of Azul documentation, see the [Azul Systems Support Site](#) section of this guide.

Azul Virtual Machine Guide, Version 11
Printed in the U.S.A.
510-00256
July 2009

Notice of Rights

No part of this guide may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Azul Systems. Please note that the content in this guide is protected under copyright law even if it is not distributed with software that includes an end user license agreement.

Notice of Liability

Azul Systems reserves the right to make changes, without notice. The content of this guide is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Azul Systems. Azul Systems assumes no responsibility or liability for any errors or inaccuracies that may appear in the informational content contained in this guide.

Trademarks

Copyright © 2005–2009, Azul Systems®, Inc. All rights reserved. Azul and Azul Systems are registered trademarks of Azul Systems, Inc. in the United States and other countries. The Azul arch logo, Compute Pool Manager, and Vega are trademarks of Azul Systems, Inc. in the United States and other countries.

BEA and WebLogic are registered trademarks of BEA Systems, Inc. in the United States and other countries.

IBM and WebSphere are registered trademarks of International Business Machines Corporation in the United States and other countries.

Intel is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

JBoss is a registered trademark of JBoss Inc.

Linux is a registered trademark of Linus Torvalds.

RedHat is the property of Red Hat, Inc.

Sun, Sun Microsystems, Solaris, J2EE, J2SE, Java and all Java based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Other marks are the property of their respective owners and are used here only for identification purposes. Products and specifications discussed in this document may reflect future versions and are subject to change by Azul Systems without notice.

Contents

About This Guide	vii
Purpose.....	vii
Audience.....	vii
Organization	viii
Related Documentation	viii
Conventions.....	viii
Azul Systems Support Site.....	ix
Contacting Azul Systems	ix
Chapter 1: Overview.....	1
Network Attached Processing	1
Azul Virtual Machine Software.....	2
Standards-compliant Multi-platform Support	2
Encrypted Data Path.....	2
Very Large Heap Size Support.....	2
Pauseless Garbage Collection	2
Generational Pauseless Garbage Collection (GPGC)	2
DirectPath	2
Cooperative Memory Management and Memory Resiliency	3
Grant Pool	3
GC Pause Prevention Pool	3
Optimistic Thread Concurrency.....	4
Chapter 2: Installation	5
Azul VM Software Packages	5
Supported Host Platforms.....	6
System Requirements	6
Pre-installation Requirements	7
javax.crypto	7
Install the Azul Virtual Machine Software	7
Common Procedure.....	8
RPM Installations	10
Verify the Azul VM Software Installation.....	10
IP Unicast Discovery	11

IP Unicast Verification	11
IP Multicast Discovery	12
IP Multicast Verification	12
DNS Entry Notes for the Policy Server Virtual IP Address	13
Direct Selection of Discovery Method.....	13
Chapter 3: Command Line Options.....	15
Java Command Line Options for the Azul VM.....	15
Compatibility with HotSpot VM Command Line Options	15
VM Proxy Options	16
Set Application Co-location Group.....	18
DirectPath Options	18
System Call Tracing Options	18
Garbage Collection Options	19
Property Control Options	19
Other Azul VM Options	20
Java Command Line Modifications for Azul Compute Pools.....	20
IP Discovery	20
Committed Memory.....	21
Committed Memory Calculation	21
Launch Parameter Control.....	22
Launch Parameter File Processing.....	23
Specify Default Launch Parameters for all Applications.....	23
Set Launch Parameters for a Single Application Environment.....	23
Rules for Editing the Launch Parameter Files	23
Run Java -version without Additional Command Line Arguments	24
Library Mapping.....	24
Trace Messages	24
Specify Native Libraries.....	25
Library Mapping File Formatting Rules.....	25
Chapter 4: REALTime Performance Monitor (RTPM)	27
About the REALTime Performance Monitor (RTPM).....	27
General Guidelines for Capturing Data	28
Azul Support	28
System Call Tracing.....	28
Statistical Tick Profiler.....	28
Table Sorting	28
Web Browser Support	28
Enable RTPM.....	29
User Authentication and Authorization.....	29
User Databases.....	30
Password Database Utility.....	30
Session Time Out	31
Authorization	31
Authorization Levels and Users	31
Syntax of security.conf	31

Log In to RTPM	32
Certificate Security Warnings.....	32
Provide a Secure User Certificate	33
Login Window.....	33
Graphical User Interface.....	34
Configuration Tab.....	35
Process Window	35
Environmental Variables Window.....	36
HotSpot Flags Window	37
Threads Tab	38
Threads List Window.....	38
Stack Trace Window	39
Deadlocks Window	40
I/O Tab.....	41
System Call Profile Window	41
Examine System Call Profile Data.....	42
Outgoing and Incoming RPC Profile Windows	42
FDC Profile Window	44
FDC Stats Window	45
Transport Profile Window	46
Open Sockets Window	47
Open Files Window	49
Buffers Window.....	50
Trace Window.....	51
Trace Settings Window.....	51
Ticks Tab.....	52
Data Collection after Application Launch.....	53
Monitors Tab.....	53
Types of Monitors	54
Identifying Applications with Locking Problems	54
Calculate Percentage of Time a Typical Thread is Blocked on a Monitor	54
Memory Tab	55
Summary Window.....	55
GC Summary Window	56
GC History Window.....	59
Allocated Objects Window.....	60
Live Objects Window	61
Compiler Tab.....	62
Settings Tab	63
Appendix A: Azul VM Configuration for IBM WebSphere	65
Create a WebSphere-compatible JDK from an Existing Azul JDK.....	65
WebSphere 5.1 installations.....	66
WebSphere 6.0 and 6.1 installations.....	66
Appendix B: Azul VM Configuration for Resin	69
Overview	69

Contents

Script File Syntax.....	70
Running the Script.....	70
Index	73

About This Guide

This chapter provides an overview of the *Azul Virtual Machine Guide*. It covers the following topics:

- [Purpose](#)
- [Audience](#)
- [Organization](#)
- [Related Documentation](#)
- [Conventions](#)
- [Azul Systems Support Site](#)
- [Contacting Azul Systems](#)

Purpose

The *Azul Virtual Machine Guide* explains how to install and manage the Azul Virtual Machine (VM) software.

Audience

This guide is intended for application administrators who install and manage the Azul Virtual Machine software.

Organization

The *Azul Virtual Machine Guide* guide is organized as follows:






- [Chapter 1, “Overview”](#)—provides an overview of network attached processing and the Azul VM.
- [Chapter 2, “Installation”](#)—provides installation requirements and procedures for installing the Azul VM.
- [Chapter 3, “Command Line Options”](#)—describes command line options that can be used with the Azul VM.
- [Chapter 4, “REALTime Performance Monitor \(RTPM\)”](#)—describes the REALTime Performance Monitor profiling and diagnostics tool embedded in the Azul VM.
- [Appendix A, “Azul VM Configuration for IBM WebSphere”](#)—explains how to configure the Azul VM for use in the IBM® WebSphere® environment.
- [Appendix B, “Azul VM Configuration for Resin”](#)—explains how to configure the Azul VM for use in the Caucho Technology Resin® environment.

Related Documentation

- *Compute Appliance Installation and Reference Guide (Model 960)*
- *Compute Appliance Installation and Reference Guide (Models 1920 and 3840)*
- *Compute Appliance Installation and Reference Guide (Models 7240 and 7280)*
- *Azul Command Line Interface Guide*
- *Compute Pool Manager Guide*

Conventions

The following conventions are used in this guide:

WARNING 	Means <i>danger</i> . This situation could cause bodily injury. Before working on any equipment, be aware of the hazards involved and familiar with standard practices for preventing accidents.
CAUTION 	Means <i>be careful</i> . In this situation, something is being done that could result in equipment damage or loss of data.
GO TO 	Points to related topics or reference documentation.
TIP 	Provides additional information to help solve a problem.
NOTE 	Provides additional information or exceptions to rules.

Azul Systems Support Site

Azul documentation is available to registered customers through Azul Systems support sites. To access the Azul support site, go to: <http://support.azulsystems.com>.

This password-protected site requires a user name and login. To obtain a password, contact Azul Systems at 1.800.258.4199. To obtain a login, contact your designated Azul representative, or send a request to: support_admin@azulsystems.com.

Contacting Azul Systems

	Address	Phone	Fax	Email/WWW
General	Azul Systems, Inc. 1600 Plymouth Street Mountain View, CA 94043	650.230.6500	650.230.6500	info@azulsystems.com http://www.azulsystems.com
Support Services	Azul Systems, Inc. 1600 Plymouth Street Mountain View, CA 94043	1.800.258.4199	650.230.6500	support_admin@azulsystems.com http://support.azulsystems.com

Notes

Overview

This chapter provides an overview of network attached processing and the Azul Virtual Machine software. The following topics are covered:

- [Network Attached Processing](#)
- [Azul Virtual Machine Software](#)
- [Standards-compliant Multi-platform Support](#)
- [Encrypted Data Path](#)
- [Very Large Heap Size Support](#)
- [Pauseless Garbage Collection](#)
- [Generational Pauseless Garbage Collection \(GPGC\)](#)
- [DirectPath](#)
- [Cooperative Memory Management and Memory Resiliency](#)
- [Optimistic Thread Concurrency](#)

Network Attached Processing

Azul Systems has introduced the industry's first network attached processing solution to provide massive compute resources for applications built on virtual machine-based technologies, such as the J2EE™ platform.

Network attached processing augments traditional host servers with a compute pool—a group of mountable, ultra high-capacity Azul Compute Appliances that provides virtually unlimited processor and memory resources for Java™ and J2EE applications. In combination with the companion Azul VM software, compute pools deliver massive processor and memory resources for Java-based workloads originating from application host servers.

Network attached processing allows Java applications installed on general-purpose application servers to access the processing power of specialized compute appliances within a compute pool. Instead of running on the host server, the Java workload is redirected to the compute appliance while all interfaces to clients, databases, and host operating system services remain on the host server. By transforming a cluster of individual compute appliances into a flexible pool of compute resources, applications are provided with the necessary processor and memory resources to deliver predictable and reliable response times under any load condition.

Azul Virtual Machine Software

The Azul Virtual Machine (VM) software redirects the Java workload processing from a conventional server to an Azul Compute Appliance, and enables applications to transparently access high-capacity pools of Azul Compute Appliances.

Standards-compliant Multi-platform Support

The Azul VM software is J2SE 1.4.2 and J2SE 5.0 compliant and supports popular application host hardware platforms and operating systems. Popular third-party debugging and profiling tools are supported through the JVMDI debugging interface, the JVMPI profiling interface, and the JVMTI debugging and profiling interface. Additionally, the flexibility of network attached processing allows IT managers to use the same compute pool resources for multiple applications running simultaneously on different application host hardware and operating systems.

Encrypted Data Path

AVM has an option to augment the security of network communication between the VM proxy and the VM engine by encrypting this communication. Customers concerned about the security of the network segment connecting servers and compute appliances have several choices for keeping application information transmitted over internal engine-proxy channels safe through encryption. Encryption can be limited to certain types of proxy-engine communication, or applied to all such communication. Encryption complies with SSL RC440, 128-bit key. Impact on performance is application dependent.

Very Large Heap Size Support

Azul Compute Appliances use 64-bit addressing and support application heap sizes up to 320 GB. The Azul VM allows 32-bit application host environments to use large heap sizes with little or no modification.

Pauseless Garbage Collection

The Azul VM software provides a unique pauseless garbage collection (GC) option that virtually eliminates pauses that are often a source of application response time problems in conventional virtual machines. The Azul pauseless GC option is the default garbage collector.

Generational Pauseless Garbage Collection (GPGC)

The Azul GPGC is a new, enhanced garbage collector for the Azul VM. It combines the benefits of Pauseless GC and generational collectors to drive application response time predictability to new records while reducing memory heap footprint. GPGC is an optional (non-default) collector.

DirectPath

DirectPath is a novel technology from Azul Systems that enhances the Network Attached Processing architecture. DirectPath enables Java applications running on the Azul VM to interact directly with other application components. By shifting the processing load of I/O-heavy applications from servers to the Azul VM, DirectPath has a dramatic effect on server processing capacity, network transaction latency overhead, and application instance scalability.

Previously, two applications in the Azul compute pool communicate through exchanges between proxy ports. Application network sockets use an IP port on the corresponding proxy server to mediate all communication received from and sent to other applications. Applications are oblivious to the VM Engine; the VM Proxy mechanism facilitating this “communication by proxy.”

DirectPath makes one key modification: it forms a direct network path between the corresponding appliances and opens a connection on that path. While TCP/IP control (setup, state transition, and so on) is maintained on the proxy, all data exchanges occur on the appliance path. When both instances reside on the same compute appliance, the exchange loops back internally without ever reaching the physical layer. DirectPath is supported in both stream socket and datagram socket communications, and requires minimum AVX and CPM software release levels installed on the compute appliance.

Refer to the AVM release notes for the latest compatibility requirements. Refer to the *Azul DirectPath* application note for an overview of the technology, set up and management aspects, as well as related applicability, RAS, and security issues.

Cooperative Memory Management and Memory Resiliency

The Azul VM provides resiliency to out-of-memory conditions by providing access into shared memory pools.

Grant Pool

The *grant pool* allows the total memory for a process to grow beyond the committed memory level (the memory level guaranteed to the process), by allocating additional memory from a shared appliance-level pool. This growth is limited by how much memory space remains in the grant pool and the MemMax limitation on total application memory. As long as these two conditions are not exceeded, the process survives. The grant pool is only available with Azul pauseless GC or Generational PGC.

Once using memory from the grant pool, the application is at risk since there are no guarantees set for additional grant memory. AVM tracks application memory usage and returns unused memory to the grant pool as soon as the application decreases its live set requirements.

The grant pool is shared across all VMs within an appliance and can only be effective in preventing processes from terminating with out of memory errors if the increases in memory requested by the various VMs are kept to a minimum. Application that experience memory leaks will eventually exhaust the grant pool, potentially affecting other VMs running in the same appliance and using grant memory.

GC Pause Prevention Pool

The GC *pause prevention pool* allows GC processes to allocate extra, temporary memory to Java objects to prevent GC pauses. This is useful when the memory allocation rate for the application exceeds the rate that the garbage collector can free memory. On GC cycle completion, the extra memory returns to the pool and is made available for other GC processes. The GC pause prevention pool is only available if the Azul pauseless GC option is enabled.

Optimistic Thread Concurrency

Java and J2EE platform-based applications are usually highly threaded, and use locks to manage concurrent access to shared data. On conventional systems, threads often wait on locks held by other threads even though no data conflict would actually occur if the threads ran concurrently. Thread contention is a common cause of poor application scalability.

The Azul VM uses the innovative lock management scheme, optimistic thread concurrency, to improve throughput. Optimistic thread concurrency allows processing of all threads to proceed while holding locks as long as data conflicts do not occur. If there are no conflicts, the writes are committed and the thread proceeds as expected. If a conflict occurs, the memory writes are rolled back and restart after the lock contention is resolved. Because conflicts only occur in a small number of cases, the overall effect is a significant improvement in application throughput. Existing unmodified programs and commonly used container libraries can use this capacity to scale large numbers of processor cores.

Installation

This chapter provides installation requirements and procedures for the Azul Virtual Machine (VM) software. It describes the following topics:

- [Azul VM Software Packages](#)
- [System Requirements](#)
- [Pre-installation Requirements](#)
- [Install the Azul Virtual Machine Software](#)
- [Verify the Azul VM Software Installation](#)

Azul VM Software Packages

The Azul VM software packages for Java 2 Platform, Standard Edition (J2SE), Version 1.4.2, J2SE 5.0, or J2SE 6.0 are distributed on the Azul support portal in either full or compact package, for compressed file extraction, or in RPM format for use with RPM package management utilities.

Each package includes Azul VM software for all host platforms. Full packages include a stock JDK for the Linux and Solaris platforms; on other platforms (such as, AIX or HP/UX), the stock JDK must be provided and installed on the host platform by the customer. Compact packages do not include a stock JDK for any platform, requiring that the stock JDK be pre-installed on the target host for all platforms. [Table 2-1](#) describes available packages.

Table 2-1 *Available Software Packages*

Package	Description
Azul VM 1.4.2 Full package	Full installation package of the Azul VM for Java 2 Standard Edition 1.4.2, on the Linux, Solaris, HP/UX, and AIX platforms. Includes stock JDKs for Linux and Solaris, a self-extracting installation shell script, and release notes.
Azul VM 1.4.2 Compact package	Compact installation package of the Azul VM for Java 2 Standard Edition 1.4.2, on the Linux, Solaris, HP/UX, and AIX platforms. Requires a local JDK on the target host. Includes a self-extracting installation shell script and release notes.
Azul VM 1.4.2 RPM package	RPM-compatible installation package for Azul VM for Java 2 Standard Edition 1.4.2, on the Linux platform. Includes stock JDKs and release notes.
Azul VM 5.0 Full package	Full installation package of the Azul VM for Java 2 Standard Edition 5.0, on the Linux, Solaris, HP/UX, and AIX platforms. Includes stock JDKs for Linux and Solaris, a self-extracting installation shell script, and release notes.
Azul VM 5.0 Compact package	Compact installation package of the Azul VM for Java 2 Standard Edition 5.0, on the Linux, Solaris, HP/UX, and AIX platforms. Requires a local JDK on the target host. Includes a self-extracting installation shell script and release notes.

Table 2-1 Available Software Packages (Continued)

Package	Description
Azul VM 5.0 RPM package	RPM-compatible installation package for Azul VM for Java 2 Standard Edition 5.0 on the Linux platform. Includes stock JDKs and release notes. Available in 32-bit for Intel or AMD and 64-bit for AMD platforms.
Azul VM 6.0 Full package	Full installation package of the Azul VM for Java 2 Standard Edition 6.0, on the Linux, Solaris, HP/UX, and AIX platforms. Includes stock JDKs for Linux and Solaris, a self-extracting installation shell script, and release notes.
Azul VM 5.0 Compact package	Compact installation package of the Azul VM for Java 2 Standard Edition 6.0, on the Linux, Solaris, HP/UX, and AIX platforms. Requires a local JDK on the target host. Includes a self-extracting installation shell script and release notes.

Supported Host Platforms

The Azul VM is available for the combinations of J2SE version and host platform environments shown in [Table 2-2](#).

Table 2-2 Azul VM Host Platform Support Matrix

	J2SE 1.4		J2SE 5.0		J2SE 6.0	
	32-bit	64-bit	32-bit	64-bit	32-bit	64-bit
Solaris (SPARC)	x	x	x	x	x	x
Solaris (x86)	x		x		x	
Linux (x86)	x		x	x	x	x
AIX (PowerPC)	x	x	x	x	x	x
HP-UX (PA-RISC)	x		x		x	

System Requirements

The following are software and system requirements for running the Azul VM software:

- 300 MB of free disk space
- For Linux® operating systems:
 - Intel® IA-32 or compatible processor
 - Red Hat Linux AS or ES, versions 2.1, 3.0, 4.0, or 5.0
 - Novell® SUSE® Linux Professional versions 8 or 9, using kernel 2.4 or 2.6
- For Solaris™ operating systems:
 - Solaris 8, 9, or 10 operating system on a Sun™ SPARC-based system
 - Solaris 10 operating system on x86-based system
- For AIX operating systems:

- ❑ IBM® AIX® 5.2 or 5.3 on a PowerPC-based system
- ❑ A local IBM JDK (not included in the Azul VM distribution)



All application hosts and NFS servers must be configured to use an NTP server. The Azul Virtual Machine software uses advanced file caching technologies that rely on accurate time synchronization between application host servers and NFS servers that the VM-based application may be accessing. VM-based applications running on hosts not synchronized with the NFS server to an NTP server are vulnerable to application crashes and file corruption.

- For HP/UX operating systems:
 - ❑ HP/UX 11i on a PA-RISC-based system
 - ❑ A stock HP JDK (not included in the VM distribution)
- For compact installations:
 - ❑ A stock JDK installation

Pre-installation Requirements

Complete the following tasks before installing the Azul VM software:

- Install all compute appliances on the network
- Establish and configure the CPM domain



For information on installing a compute appliance, go to the *Compute Appliance Installation and Reference Guide*. For information on initializing a new CPM domain, go to the *Compute Pool Manager Installation and Upgrade Guide*.

- gzip is required for all installations. gzip is available at Sunfreeware.com.

javax.crypto

If you are using javax.crypto unlimited strength policy files and have copied these files into your JDK, they must also be copied into the Azul JDK. There are two jar files in the \$JAVA_HOME/jre/lib/security directory: US_export_policy.jar and local_policy.jar. If these files are not copied to the Azul JDK, the following exception displays on install:

```
java.lang.SecurityException: Unsupported keysize or algorithm parameters
at javax.crypto.Cipher.init(DashoA12275)
```

Install the Azul Virtual Machine Software

The Azul VM software is installed from either a compressed ZIP file that includes a self-extracting installation shell script or from an RPM file for use with RPM Package Manager installations in a Linux environment.

Use the instructions in [Common Procedure](#) to complete AVM software installation using the compressed ZIP file method on the following supported platforms:

- | | | |
|-------------------|-----------------|-------------------|
| ■ Linux (x86) | ■ AIX (PowerPC) | ■ HP/UX (PA-RISC) |
| ■ Solaris (SPARC) | ■ Solaris (x86) | |

For RPM Package Manager installations, see [RPM Installations](#) on page 10.

Common Procedure

Use the following procedure common to all supported platforms to install the Azul VM software. Use either the Full or the Compact installation package for this procedure. For Compact package installations, the stock JDK must already be installed for all host platforms. When the Full package is used the stock JDK for Solaris or Linux platforms is part of the installation package, and is automatically accessed. For other platforms (AIX and HP-UX) the stock JDK must already be installed.

Step 1 Copy the .sh file for the appropriate Java 2 Standard Edition version to the desired Azul VM directory.

The file has the following naming format depending on the Java version supported.

- ❑ Full version (including Solaris and Linux stock JDK):

```
azul-jdk1.4.2-<version>-<buildnumber>-full.sh
```

or

```
azul-jdk1.5.0-<version>-<buildnumber>-full.sh
```

- ❑ Compact version (does not contain any stock JDKs):

```
azul-jdk1.4.2-<version>-<buildnumber>-compact.sh
```

or

```
azul-jdk1.5.0-<version>-<buildnumber>-compact.sh
```

Each compressed ZIP file installs an Azul AVM compatible with all supported platforms.

Step 2 Install the Azul VM software.

- a. Run the script file to install the Azul VM software corresponding to the host platform where the applications will be run.
- b. Type the file name preceded by the ./ characters to run the script, as follows:

```
./<filename>
```

For example,

```
./azul-jdk1.4.2-2.3.0.0-124-full.sh
```

Without options, this script automatically detects the current host platform and installs the 32 bit compatible version of the Azul VM software.






NOTE The pax utility is required to run the script. If the pax utility is not available, use the **-tar** option instead.

For example,

```
./azul-jdk1.4.2-2.3.0.0-124-full.sh -tar
```

Options:

- tar** to use the GNU tar executable when the pax utility is not available.
Example:
`./azul-jdk1.4.2-2.3.0.0-124-full.sh -tar`
- target <sys>** to manually specify the host compatible version of Azul VM software to install.
Available options for <sys> are:
- | | |
|-----------|---|
| linux | Linux running on x86 (32-bit) |
| linux64 | Linux running on x86 (64-bit) |
| solaris | Sun Solaris running on SPARC (32-bit or 64-bit) |
| solaris86 | Sun Solaris running on x86 (32-bit) |
| aix | IBM AIX running on PowerPC (32-bit) |
| aix64 | IBM AIX64 running on PowerPC (64-bit) |
| hpux | HP/UX running on PA-RISC (32-bit) |
- Example:
`./azul-jdk1.4.2-2.3.0.0-124-full.sh -target aix64`
-  **NOTE** The **-target** option must be used to install 64-bit compatible versions of the Azul VM software for Linux or AIX.
- was** to indicate that the Azul VM software will be used with IBM WebSphere (version 5.1 or 6.0).
 **NOTE** Refer to [Appendix A, Azul VM Configuration for IBM WebSphere](#) for information on the script run when using the **-was** option.
- resin** to indicate that the Azul VM software will be used with Caucho Technology's Resin application server 3.0.9 or higher
 **NOTE** Refer to [Appendix B, Azul VM Configuration for Resin](#) for information on the script run when using the **-resin** option.
- cp** to install a full copy of the Solaris or Linux stock JDK along with the Azul VM software when using the Full installation package.

Step 3 Provide the location for the stock JDK.

Installation of the Azul VM software requires access to the stock JDK corresponding to the chosen platform. The stock JDK must match the version of Azul VM software being installed, JDK 1.4 or JDK 5.0 (1.5), and 32- or 64-bit.

- ❑ Compact installation or when installation is for an AIX or HP/UX platform:

Enter the path to the stock JDK when prompted, as follows:

The Azul VM installer needs the location of the current local JDK to complete the installation. Enter the desired path to the JDK (leave empty to use `$JAVA_HOME`).
Current local JDK to use: <path name>

- ❑ Full installation for Solaris or Linux:

The stock JDK is within the package and accessed as required by the script. Entering the stock JDK location is not required.

When the installation completes, the following message displays:

Installation completed successfully.

Step 4 Install JDK customizations.

Make customizations to the Azul JDK to match any done to the stock JDK. For example, if jar files were added to an ext directory, those files must also be added to the Azul JDK.

Step 5 Verify the installation.

Perform the procedures listed in [Verify the Azul VM Software Installation](#) to verify that Java programs can start from an application host server and execute on the Azul compute pool using the Azul VM.

RPM Installations

Three files are available to install the AVM package into an RPM facility on a Linux platform.

- JDK 1.4, full installer (includes native JDK)
 - 32-bit: azul-jdk-1.4.2<version>-<build number>.i386.rpm
- JDK 5.0, full installer (includes native JDK)
 - 32-bit: azul-jdk-1.5<version>-<build number>.i386.rpm
 - 64-bit: azul-jdk-1.5<version>-<build number>.x86_64.rpm

To install the JDK:

Step 1 Download the file to the desired host server.

Step 2 Ensure that the file size is the same as shown on the download page.

Step 3 Enter the root shell:

- a. Type the su command.
- b. Enter the super-user password when prompted.

Step 4 Install the download file:

For example:

```
rpm -ivh ./azul-jdk1.4.2_15-2.4.1.0-682.i386.rpm
```

The contents of the RPM package are installed into the /opt/azul directory. To examine the contents of the RPM package prior to installation, execute the rpm command using the -qpl option. For example:

```
rpm -qpl ./azul-jdk1.4.2-<version>-<build number>.i386.rpm
```

Verify the Azul VM Software Installation

After installing the Azul VM software, verify that Java programs can start from an application host server and can execute on the Azul compute pool using the Azul VM. The Azul VM proxy can use multicast or unicast IP discovery to establish communication with the compute pool. By default, unicast discovery is enabled and multicast discovery disabled on the Azul VM proxy. This section explains how to verify the Azul VM software installation for both discovery mechanisms. Choose and verify the mechanism to use for compute pool configuration. Once the discovery mechanism is verified, installation is complete.

IP Unicast Discovery

IP unicast discovery is a routing technique that allows IP traffic to propagate from a single source to a single destination. This method is required in network configurations not supporting multicast communications from host servers to Azul Compute Appliances.



NOTE

For unicast discovery, the VM proxy must know the virtual IP address (VIP) of the primary CPM policy server.

This discovery method allows the Azul VM proxy to determine the compute pool domain policy server to establish application placement and resource allocation parameters. Specify the following parameter for the policy server virtual host to instruct the Azul VM proxy to use IP unicast discovery:

```
-PX:PolicyVHost=<name of policy server, either IP address or host name>
```

When this parameter is specified in the command line, multicast is not used.

Ensure that the following requirements are met to verify IP unicast discovery:

- The virtual host IP address of the CPM policy server is specified during compute pool initialization.
- The CPM domain name is specified during CPM domain initialization. The default CPM domain name is *cpm01*.

The policy server VIP address can be mapped to a user-created virtual host name on a DNS server. If a DNS mapping is used, the virtual host name can be specified in place of the policy server VIP address in the Java command line.



NOTE

The default DNS virtual host name is “cpm01-ps.” If “cpm01-ps” is the virtual host name, it does not need to be specified in the Java command line.

IP Unicast Verification

Use the following procedure to verify IP unicast discovery for a compute pool domain where multicast is disabled. Unicast discovery can be over TCP or UDP.

Step 1 Log on to a local host used in conjunction with compute appliances in the CPM domain.

Step 2 Type the following Java command line:

```
$JAVA_HOME/bin/java -PX:CPMDomain=<domain name>
-PX:PolicyVhost=<policy server VIP address> -version
```



NOTE

If using the default CPM domain name “cpm01”, the -PX:CPMDomain option is not necessary.

The following response displays:

```
java version <java version>
Java(TM) 2 Runtime Environment, Standard Edition <build version>
Java HotSpot(TM) 64-Bit Server VM (build 1.4.2_<Sun patch version>-AVM-
<Azul distribution version>, mixed mode)
```

For example,

```
java version "1.4.2_11"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2_11-16)
```

```
Java HotSpot(TM) 64-Bit Server VM (build 1.4.2_11-AVM_2.3.0.0-70, mixed mode)
```

Display of this response to **java -version** indicates that the Java SDK is successfully redirected from the host to the Azul compute pool.



NOTE The policy server VIP address must be the same as configured for the target CPM domain. Refer to the *Compute Pool Manager Guide*.

IP Multicast Discovery

IP multicast is a simple mechanism for allowing a VM proxy to communicate with a target compute pool. If network configuration supports IP multicast forwarding, the IP multicast discovery method can be used. This discovery method allows the Azul VM proxy to determine on which CPM domain policy server to establish application placement and resource allocation parameters. For multicasting to work, the Azul VM proxy must include the CPM domain name in its broadcast. The multicast address is 224.0.23.43:9876.

Ensure that the following requirements are met to verify IP multicast discovery.

- Include the `-PX:+MulticastDiscovery` command line option to enable multicast IP discovery.
- A network configuration that supports IP multicast from the application host server to the compute appliances in the CPM domain.
- The CPM domain name is specified during CPM domain initialization. The default CPM domain name is *cpm01*.

IP Multicast Verification

Use the following procedure to verify IP multicast discovery for a compute pool domain.

- Step 1** Log on to a local host used in conjunction with compute appliances in the CPM domain.
- Step 2** Set the `JAVA_HOME` system variable (or equivalent) to point to the parent directory where the Java executable resides in the Azul VM software, as follows:

```
JAVA_HOME=/<Azul VM directory>
```

- Step 3** Do one of the following:

- Type the following if using the default CPM domain name (*cpm01*):

```
$JAVA_HOME/bin/java -version  
-PX:+MulticastDiscovery
```
- Type the following if using a user-defined CPM domain name:

```
$JAVA_HOME/bin/java -PX:CPMDomain=<domain name>  
-PX:+MulticastDiscovery -version
```

For example, type the following if the domain name is *cpm99*:

```
$JAVA_HOME/bin/java -PX:CPMDomain=cpm99  
-PX:+MulticastDiscovery -version
```

The following response displays:

```
java version <java version>
```

```
Java(TM) 2 Runtime Environment, Standard Edition <build version>
Java HotSpot(TM) 64-Bit Server VM (build 1.4.2_<Sun patch version>-AVM-
<Azul distribution version>, mixed mode)
```

For example:

```
java version "1.4.2_10"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2_11-16)
Java HotSpot(TM) 64-Bit Server VM (build 1.4.2_11-AVM_2.3.0.0-70,
mixed mode)
```

This response to **java -version** indicates that the Java SDK was successfully redirected from the host to the Azul compute pool.

DNS Entry Notes for the Policy Server Virtual IP Address

- If the policy server VIP and CPM domain name are not specified, the VM proxy assumes that the VIP is mapped to the default DNS entry (cpm01-ps).
- If only the CPM domain name is specified, the VM proxy attempts DNS lookup using <CPM domain name>-ps.
- When the policy server virtual host (-PX:PolicyVhost) is not explicitly specified, the default DNS name is used.
- The policy server virtual host (-PX:PolicyVhost) can be specified as an IP address or a text string. If using a text string to define the policy server virtual host, DNS lookup is performed using that string.

Direct Selection of Discovery Method

Table 2-3 describes the command line options used for IP discovery.

Table 2-3 *Java Command Line Options for Discovery Method*

Java Command Option	Description
-PX:+MulticastDiscovery	Enables multicast IP discovery of CPM.
-PX:+UnicastUDPDDiscovery	Enables unicast UDP discovery of CPM.
-PX:+UnicastTCPDiscovery	Enables unicast IP TCP discovery of CPM.

Use the -PX:+MulticastDiscovery command line option to enable the multicast IP discovery option. Multicast requirements include having a multicast group defined in the network, and specifying the domain with **-PX:CPMDomain** or using the default domain.

If a DNS entry for the policy virtual host is defined and matches the default or the value in **-PX:PolicyVhost** (or its IP address is specified in this option), then unicast discovery functions properly. It is not necessary to create both definitions.

Notes

Command Line Options

This chapter discusses the use of Azul Virtual Machine command line options that control virtual machine operation and the interface to CPM. The following topics are discussed:

- [Java Command Line Options for the Azul VM](#)
- [Java Command Line Modifications for Azul Compute Pools](#)
- [Launch Parameter Control](#)
- [Library Mapping](#)

Java Command Line Options for the Azul VM

This section describes the Java command line options for the Azul VM.

Compatibility with HotSpot VM Command Line Options

The Azul VM supports all of the native command line options for the HotSpot VM that is a core component of the Java SE platform. Some garbage collection options are automatically mapped to the Azul VM equivalent (see [Garbage Collection Options](#)).

In the case that the Azul VM detects a HotSpot VM native command line option that it does not support, the Azul VM response can be set with the following option:

`-Xnativevmflags:[ignore|error|warn]`

The default option is **warn**.. If an supported option falling into this category is detected, the Azul VM generates the following type of message:

```
Java HotSpot™ 64-Bit Server VM warning: -XX:+UseParNewGC is being
treated as -XX:+UseParallelGC.
```

If **-Xnativevmflags:error** is specified and an option falling into this category is detected, the Azul VM does not launch and generates the following type of message:

```
Java HotSpot™ 64-Bit Server VM error: Unsupported native VM option
'+UseParNewGC'
```

If **-Xnativevmflags:ignore** is specified and an unsupported option falling into this category is detected, no output is displayed.

For command line options that are not native to the HotSpot VM (including those specific to the Azul VM), the Azul VM response is determined by the following option:

`-Xflags:[ignore|error|warn]`

The default setting, error, ensures that unrecognized options are highlighted when the Azul VM is initially used. For example, if the incorrectly formatted option **-XXMaxPermS=128m** is detected, the Azul VM does not launch and generates the message:

```
Java HotSpot™ 64-Bit Server VM error: Unrecognized VM option
'MaxPermS=128m'
```

Several Java options that normally configure VM behavior are not relevant in the Azul VM (for example, **-XX:+UseParNewGC** and **-XX:+UseConcMarkSweepGC**).

VM Proxy Options

Java command line options that control the interaction between the VM proxy and the CPM or AVX system software begin with the prefix **-PX:**. These options are summarized in [Table 3-1](#).

Table 3-1 *Azul Java Command Line Options*

Command Line Option, Note 1	Description	Default
-PX:+Help	Displays the Azul Java command line option help pages.	N/A
-PX:AppLabel=<application label string>	Specifies the application label used by CPM.	unspecified
-PX:GroupLabel=<application group label string>	Specifies the application group label used by CPM.	unspecified
-PX:CPMDomain=<CPM domain name string>	Specifies the CPM domain name where the application will run (requires entry in DNS).	cpm01
-PX:PolicyVhost=<policy server virtual host name or IP address>	Specifies the virtual host name or IP address of the CPM policy server.	cpm01-ps
-PX:DiscoveryTimeout=<time>	Specifies the amount of time to retry CPM discovery. Units can be specified in seconds or minutes. Default unit is seconds. For example: <ul style="list-style-type: none"> -PX:DiscoveryTimeout=60 -PX:DiscoveryTimeout=60s -PX:DiscoveryTimeout=1m 	30 seconds
-PX:MemCommit=<size>	Specifies the committed memory for the VM process. Units can be specified in kilobytes (K or k), megabytes (M or m), or gigabytes (G or g). For example: <ul style="list-style-type: none"> -PX:MemCommit=2g -PX:MemCommit=128M -PX:MemCommit=128m 	unspecified See Committed Memory on page 21 for information on the calculation of the default size of committed memory for the VM process.
-PX:MemMax=<size> Note 2	Specifies the maximum amount of memory that can be allocated to the VM process. MemMax can be greater than MemCommit, but not smaller. Units can be specified in kilobytes (K or k), megabytes (M or m), or gigabytes (G or g). For example: <ul style="list-style-type: none"> -PX:MemMax=2g -PX:MemMax=128M -PX:MemMax=128m 	No maximum

Table 3-1 *Azul Java Command Line Options (Continued)*

Command Line Option, Note 1	Description	Default
-PX:FileDataCacheSize =<nnn>	Specifies the memory size used to cache file data. Units can be specified in kilobytes (K or k), megabytes (M or m), or gigabytes (G or g). For example: <ul style="list-style-type: none"> -PX:FileDataCacheSize=2g -PX:FileDataCacheSize=128M -PX:FileDataCacheSize=128m To change the default value of -PX:FileDataCacheSize, -PX:MemCommit must also be increased accordingly. Failure to do so may result in application out-of-memory termination.	32M
-PX:IOEncryptionMode	Sets the encryption mode used, as follows: <ul style="list-style-type: none"> 0: no encryption 1: encrypts all control messages and file accesses through read/write between VM proxy and VM engine 2: encrypts all control messages, file accesses through read/write and socket data between VM proxy and VM engine 	-PX:IOEncryptionMode=0
-PX:+MulticastDiscovery	Enables multicast IP discovery of CPM.	Not enabled
-PX:-MulticastDiscovery	Disables multicast IP discovery of CPM.	Enabled
-PX:+UnicastUDPDiscovery	Enables unicast UDP discovery of CPM.	Enabled
-PX:-UnicastUDPDiscovery	Disables unicast UDP discovery of CPM.	Not enabled
-PX:+UnicastTCPDiscovery	Enables unicast IP TCP discovery of CPM.	Enabled
-PX:-UnicastTCPDiscovery	Disables unicast IP TCP discovery of CPM.	Not enabled
-PX:BackendCoredumpLimit=<size>	Sets the allowable core dump file size. The value can be specified as bytes (no suffix), kilobytes (K or k), megabytes (M or m), gigabytes (G or g), or terabytes (T or t). The value can be 'unlimited,' which puts no predetermined limits on the core dump file size.	If unspecified, the core dump file size is the file size of the command shell that launched Java (which can also have an 'unlimited' value).
-PX:+BackEndLeakDetection	Enables the diagnosis of any memory leaks in the internal VM memory (also known as the C heap). The output can be captured in a text file or from RTPM.	off
-PX:NetworkTimeOut=<setting>	Sets the time duration of allowed network outage before the VM is terminated. Valid <settings> are: <ul style="list-style-type: none"> short = 10–15 seconds intermediate = 30–60 seconds long = 60–120 seconds Exact timing of the VM termination is variable, depending on the increasing retry intervals.	short

Table 3-1 *Azul Java Command Line Options (Continued)*

Command Line Option, Note 1	Description	Default
-PX:ProxySourcePortRange=<start-end>	Specifies a constrained range of ports on the host server to be used for communications between the VM proxy on the host and the assigned compute appliance. Enables compliance with firewall source port restrictions. Example: -PX:ProxySourcePortRange=5000-6000	All ports allowed
Notes: <ol style="list-style-type: none"> 1. Azul VM Java command line options are case sensitive. 2. Contact Azul customer support for more information about this option 		

NOTE



See [Chapter 4, REALTime Performance Monitor \(RTPM\)](#) for RTPM options.

Set Application Co-location Group

An application co-location group (ColoGroup) allows applications to be grouped so that they can run on the same appliance (see *Managing Resource Policies and Rules* in the *Compute Pool Manager Guide*). Add the **-PX:ColoGroup** option to authorize applications to either join an existing ColoGroup or create the group on an appliance if the group does not already exist in the domain. This ensures that these applications run on the same appliance. If ColoGroup is specified and the matching rule is not a ‘Shared: 1 per ColoGroup’ rule, the launch is rejected with the message ‘Matching rule does not support placement by ColoGroup.’

DirectPath Options

The DirectPath command line options can be enabled/disabled per Azul VM instance. When enabled, the Azul VM instance attempts to use DirectPath for every network communication it creates with another Azul VM instance. The matching CPM policy rule must also permit DirectPath to be used. If the Azul VM instance requests DirectPath but CPM denies it, the Azul VM instance will launch with DirectPath disabled. See [DirectPath on page 2](#).

[Table 3-2](#) lists the DirectPath VM Proxy commands.

Table 3-2 *DirectPath VM Proxy Java Command Line Options*

Command Line Option	Description	Default
-PX:+UseDirectPath	Enable DirectPath. Insert in the launch command for all AVM instances to reduce proxy CPU load.	enabled
-PX:-UseDirectPath	Disable DirectPath communication.	not enabled

System Call Tracing Options

Use the commands listed in [Table 3-3](#) to enable system call tracing and trace filtering. The system call trace is displayed in the RealTime Performance Monitor (under **I/O Tab > Trace Window**).

Table 3-3 Java Command Line Options for System Call Tracing

Command Line Option	Description	Default
-PX:Trace=<file-path> off stdout stderr <memory buffer size in MB>	Logs every system call made to the proxy. <ul style="list-style-type: none"> • off: deactivates the trace feature • stdout: activates trace and sends to standard output • stderr: activates trace and sends to standard error • <file-path>: activates trace and sends it to the specified path and file on the proxy file system • <memory buffer size in MB>: activates trace and captures it in memory (default: 1MB) 	off
-PX:TraceFilter=<filter-text-list>	Specifies the filter to use on trace data. <ul style="list-style-type: none"> • <filter-text-list>: creates a comma separated list of specific functions or event class names (use the ! suffix on file names to remove them from the set of event classes) 	empty (no filter)

Garbage Collection Options

Garbage collectors are described in [Table 3-4](#). The default garbage collector is

-XX:+UseGenPauselessGC.

Table 3-4 Garbage Collection (GC) Java Command Line Options

Command Line Option	Description
-XX:+UseParallelGC	Parallel throughput collector.
-XX:+UsePauselessGC	Pauseless garbage collector optimized for Azul Compute Appliances. This is not supported for J2SE 6.0. If specified for J2SE 6.0, the -XX:+UseGenPauselessGC will be used instead.
-XX:+UseGenPauselessGC	Generational Pauseless GC (GPGC) optimized for Azul Compute Appliances.
-XX:PrintGCHistory= <i>N</i>	Set the number of historical PGC cycle reports to save for display in the REALTime Performance Monitor (default =50).
-XX:GCWarningHistory= <i>N</i>	Set the number of historical PGC warnings to save for display in the REALTime Performance Monitor (default =50).

[Table 3-5](#) lists unsupported garbage collection options that are mapped to an Azul VM equivalent.

Table 3-5 Azul VM Garbage Collection Option Substitutions

Typical Java GC Option	Azul VM GC Option
-XX:+UseParNewGC	-XX:+UseParallelGC
-XX:+UseConcMarkSweepGC	-XX:+UsePauselessGC
-Xcongcg	-XX:+UsePauselessGC

Property Control Options

[Table 3-6](#) lists the property control command line options for the Azul VM.

Table 3-6 Property Control Java Command Line Options

Command Line Option	Description
-Dazul.ignore.nio.channels.spi.SelectorProvider	Disables the java.nio.channels.spi.SelectorProvider property control. The specified selector provider can be disabled when it has native code that does not perform optimally.

Other Azul VM Options

Table 3-7 lists other command line options for the Azul VM used to enable profile tracking.

Table 3-7 *Property Control Java Command Line Options*

Command Line Option	Description	Default
-XX:+ProfileLiveObjects	Enables live object allocation profile tracking.	enabled
-XX:+ProfileAllocatedObjects	Enables per-thread object allocation profile tracking.	enabled

Java Command Line Modifications for Azul Compute Pools

Do the following to enable the Azul VM software to transparently redirect the Java workload processing from a conventional server to an Azul compute pool:

- Step 1** Set the `JAVA_HOME` variable (or equivalent) so that the Java call points to the parent directory of the Azul VM software.
- Step 2** Make the necessary modifications to the Java command line. The required Java command line are `-PX:CPMDomain` and `-PX:PolicyVHost`.



NOTE Several other Java options can be specified for parameters such as committed memory and CPM policy rule matching. Those are not required, but can be specified when needed.

Additional modifications for Azul compute pools include:

- IP Discovery
- Committed Memory

IP Discovery

Use the following procedure to modify Java command lines when IP unicast discovery is used with the VIP address.

- Step 1** Add the `-PX:CPMDomain` option to specify the name of the target CPM domain.
- The VM proxy uses this CPM domain name to establish communication with the primary CPM policy server. This option is not required when using the default CPM domain name, `cpm01`.
- Step 2** Add the `-PX:PolicyVhost` option to specify the IP address of the CPM policy server virtual host.
- The following is an example script file where `-PX:CPMDomain` is `cpmWest` and the CPM policy server VIP address is `10.10.212.127`:

```
JAVA_HOME=/azul-j2sdk1.4.2-2.5.0.0-57-Solaris
JAVA=$JAVA_HOME/bin/java
$JAVA -classpath <path> -Xmx512m -PX:CPMDomain=cpmWest
-PX:PolicyVhost=10.10.212.127 <javaClass>
```

Use the following procedure to modify Java command lines when unicast IP discovery is used with the VIP DNS host name.

- Step 1** Configure the DNS server used by the host with a host name that corresponds to the VIP address for the CPM policy server.
- Step 2** Set the `JAVA_HOME` variable (or equivalent) so that the Java call points to the parent directory of the Azul VM software.

Step 3 Add the **-PX:CPMDomain** option and specify the name of the target CPM domain.

The VM proxy uses this CPM domain name to establish communication with the target CPM policy server. This option is not required when using the default CPM domain name, `cpm01`.

Step 4 Add the **-PX:PolicyVhost** option and specify the IP address or DNS host name assigned to the CPM policy server VIP.

The following is an example script file where **-PX:CPMDomain** is `cpmWest` and the CPM policy server DNS host name is `cpmpolicy`:

```
JAVA_HOME=/azul-j2sdk1.4.2-2.5.0.0-57-Solaris
JAVA=$JAVA_HOME/bin/java
$JAVA -classpath <path> -Xmx512m -PX:CPMDomain=cpmWest
-PX:PolicyVhost=cpmpolicy <javaClass>
```

As an option, IP multicast discovery can be used. Add the **-PX:CPMDomain** option to specify the name of the target CPM domain when IP multicast discovery is used. The VM proxy uses this CPM domain name to establish communication with the target CPM policy server. This option is not required when using the default CPM domain name, `cpm01`.

The following is an example script file where **-PX:CPMDomain** is `cpmWest`:

```
JAVA_HOME=/azul-j2sdk1.4.2-2.5.0.0-57-Solaris
JAVA=$JAVA_HOME/bin/java
$JAVA -classpath <path> -Xmx512m -PX:CPMDomain=cpmWest <javaClass>
```

Committed Memory

When a Java application starts on a compute appliance within a compute pool, memory is allocated for application heap, thread stacks, GC working space, and so on. This total amount of memory allocated is called the committed memory for the application. Committed memory for the application can be specified in two ways:

- The VM proxy by default calculates the size of committed memory based on the application heap size specified by the **-Xmx** command line option (see [Committed Memory Calculation](#)).
- The **-PX:MemCommit** command line option can be specified to override the use of the default formula to calculate the size of the committed memory for the VM process when exact memory requirements for the application are known. In this case, the **-Xmx** command line option can also be specified. If the **-Xmx** command line option is not specified, the Java heap size is set to a default value of 212 MB for version 1.4, and 1024 MB for J2SE 5.0 and J2SE 6.0.



TIP For initial evaluation of a Java application using the Azul VM, it is recommended that the committed memory be calculated using the default method. Run the application several times, viewing its memory use in CPM. Modify the command line to reflect this usage using the **-PX:MemCommit** option.



NOTE Do not use the values used on the native server environment for the **-Xmx** and **-MaxPermSize** command line options. The **-Xmx** command line option must be larger for the Azul VM because it is a 64-bit process and to enable pauseless garbage collection.

Committed Memory Calculation

For Azul pauseless garbage collection, the maximum Java heap size is determined by adding the value of the **-XX:MaxPermSize** command line option to the value of the **-Xmx** command line option:

Maximum Java Heap Size = `-Xmx` + `-XX:MaxPermSize`

The parameters in this formula are defined as follows:

- **-Xmx** specifies the maximum heap size. The default value is 212 MB in J2SE version 1.4, and 1024 MB for J2SE 5.0 and J2SE 6.0.
- **-XX:MaxPermSize** specifies the maximum permanent generation size. The default value is 128 MB.

The following formula for calculation of committed memory is used when the **-XX:MemCommit** option is not present in the Java command line:

$$\text{MemCommit} = 215 \text{ MB} + (1.25 \times \text{Maximum Java Heap Size}) + \text{FileDataCacheSize}$$

The parameters in this formula are defined as follows:

- 215 MB is a fixed constant, called base memory.
- Maximum Java Heap Size is calculated as previously described. The default value is 128 MB.
- FileDataCacheSize is the size of the file data cache. The default value is 32 MB.

The following example uses the version 1.4 default value for **-Xmx** (1024 MB in J2SE 5.0 and J2SE 6.0) and **-XX:MaxPermSize** (128 MB):

$$\text{MemCommit} = 215 \text{ MB} + (1.25 \times (212 \text{ MB} + 128 \text{ MB})) + 32 \text{ MB} = 672 \text{ MB}$$

The following example uses an **-Xmx** value of 512 MB and **-XX:MaxPermSize** value of 200 MB:

$$\text{MemCommit} = 215 \text{ MB} + (1.25 \times (512 \text{ MB} + 200 \text{ MB})) + 32 \text{ MB} = 1,137 \text{ MB}$$

Launch Parameter Control

The Azul VM proxy can read launch parameter sets from text files and use them as new system-wide defaults that change the default VM settings. The Azul VM proxy looks for two text files: azul.java.vminitargs.pre and azul.java.vminitargs.post.

The contents of azul.java.vminitargs.pre are inserted before any specified command line parameters. The contents of azul.java.vminitargs.post are appended to the specified command line. The proxy concatenates azul.java.vminitargs.pre, the specified command line parameters, and azul.java.vminitargs.post, and then processes the entire resultant parameter set.

The “pre” parameters are useful for changing parameters globally, but are overridden by any parameters specified in the command line or in the .post file. For example, include the -PX:CPMDomain=<domain name> command (or the proper IP discovery command) in the pre file so that it does not have to be specified numerous times. If most applications use an Xmx value of 1 GB, include -Xmx=1g in the pre file, and then it only needs to be specified when the Xmx value must be different.

The “post” parameters override any values set in the command line or in the pre file. Change the post file to eliminate having to change parameters in the many launch statements in the server scripts. It is best altered for tuning sessions or when running only one application.

As demonstrated in the previous example, when the discovery parameters are in the pre file, one application can use a different domain (even temporarily for testing) through a command line override. If these parameters are in the post file, command line overrides are not allowed, which is better for single domain, or similar, situations.

Launch Parameter File Processing

The VM proxy looks for the launch parameter files in two locations and in the following order, from the first file found:

- The current working directory in which the Java process was launched.
- The VM proxy home directory (the directory that `JAVA_HOME` is normally set to when Java is under `JAVA_HOME/bin`).

The uses previously described for the files assume they are in the VM proxy home location. Ensure that changes in application start script launch directories are taken into consideration. For example, use the current launch directory when different applications will run on the same physical server, using the same VM, but have different launch requirements. This is especially relevant where the same application may be deployed and run from many different machines. Bundle the launch parameter file with the application to ensure that the correct parameters are used for the specific application.

Specify Default Launch Parameters for all Applications

Use the following procedure to accelerate moving all applications from a native VM to the Azul VM.

- Step 1** Set the `JAVA_HOME` path variable.
- Step 2** Create an `azul.java.vminitargs.pre` file in the `JAVA_HOME` path.
- Step 3** Set the Azul domain discovery parameters in the `azul.java.vminitargs.pre` file (or use `azul.java.vminitargs.post` as long as no application must connect to a different domain).

After performing these steps, all Java launches run the Azul VM proxy, can locate Compute Pool Manager, and can be redirected to run on a compute appliance.

Set Launch Parameters for a Single Application Environment

To set parameters for all launches to a new value, add the new value in the `azul.java.vminitargs.post` file in the `JAVA_HOME` path. This is useful for tuning and setting memory options for a single application when all other applications are terminated, and eliminates script editing.

Rules for Editing the Launch Parameter Files

Use the `#` character at the beginning of lines to comment in the pre and post files. Non-comment lines specify launch parameters. Some options are not supported and are controlled using the `-Xflags` option (see [Compatibility with HotSpot VM Command Line Options](#)). Use the post file to override any option with the setting preferred for running in the Azul environment.



NOTE Ensure proper use of overrides. Some applications require the use of the `-client` or the `-d64` options when 64-bit support for shared objects is available (for example, in the Solaris SPARC environment).

Example 3-1 Recommended postFileOverrideSettings

```
# This overrides -client.  
-server  
# This overrides -d64.  
-d32  
#This overrides -ea or -enableassertions.  
-da  
#This overrides -esa or -enablesystemassertions.  
-dsa
```

Run Java -version without Additional Command Line Arguments

Use the following procedure to run Java `-version` in the Azul environment without additional command line arguments.

Step 1 Set the current directory to `JAVA_HOME`.

Step 2 Add the following lines to the `.post` file:

```
-PX:PolicyVhost=<ip-address>  
-PX:CPMDomain=<domain-name>
```

Step 3 Run ``${JAVA_HOME}/bin/java -version``.

Library Mapping

Some applications use shared objects (third-party native libraries) with known performance impact on the Azul Compute Appliance. Azul has identified a method to run these applications without the native library code, as long as the applications provide a fallback option for using Java libraries (if they exist).

When shared objects are loaded, the Azul VM proxy determines if the object is in the list of known adverse objects (native libraries that perform operations that conflict with the Azul VM operational model). Objects on the list are not loaded and the application receives file not found or unsatisfied link exceptions. The file names here are modified in the same way that Java modifies when it performs a `loadLibrary` action. For example, on a linux system, `wlfileio2` would cause the library `libwlfileio2.so` to not be loaded. Many Java applications, particularly application server environments, can fallback to use a pure Java class. If a fallback occurs, the performance issue is resolved. If the application does not recover, it is terminated and the following message displays:

```
"Exception in thread "main" java.lang.UnsatisfiedLinkError: no <library  
name> in java.library.path".
```

Trace Messages

The VM proxy displays the following message about loading objects:

```
Application is requesting to load shared object xxxxxxxxxxxxxxxx. This  
object is on the shared object remapping list. This load request is denied.
```

Output control is enabled through the following proxy-level flag:

```
-PX:+TraceLibraryMapping
```

This option enables output tracing when a listed native library is prevented from loading. The default option is `-PX:TraceLibraryMapping`, which displays no output.

Specify Native Libraries

The library mapping file is a list of excluded native libraries and is maintained as a text file in a known location in the Azul JAVA_HOME directory. The content of the library mapping file contains known adverse native libraries. Customers can update this list in the field as new adverse libraries are found. The library mapping file location is \$JAVA_HOME/etc/library.map. This location is convenient if the list is common to all applications using the same JDK location. Use the following VM proxy flag to override this location and allow applications to exclude different libraries:

```
-PX:LibraryMappingFile=<path to custom library mapping file>
```

The flag overrides the default location of the library mapping file.

Library Mapping File Formatting Rules

The library mapping file contains one library name per line. Lines starting with the “#” character are comments and empty lines and trailing whitespace are also ignored. The following lists the contents of the default library mapping file:

```
# library.map -- shared library loading remapping list
#
# Libraries listed here are excluded from the shared library search
process.
# They will not be loaded by the JVM.
wlfileio2
nbio
NBIO
```

Notes

REALTime Performance Monitor (RTPM)

This chapter describes configuring and using the REALTime Performance Monitor (RTPM) profiling and diagnostics tool for the Azul VM. The following topics are discussed:

- [About the REALTime Performance Monitor \(RTPM\)](#)
- [Enable RTPM](#)
- [User Authentication and Authorization](#)
- [Log In to RTPM](#)
- [Configuration Tab](#)
- [Threads Tab](#)
- [I/O Tab](#)
- [Ticks Tab](#)
- [Monitors Tab](#)
- [Memory Tab](#)
- [Settings Tab](#)

About the REALTime Performance Monitor (RTPM)

RTPM is a profiling and diagnostics tool embedded in the Azul VM. RTPM provides programmers and deployment engineers with information about both the running application and the underlying VM (proxy and engine). It also helps users tune application launch parameters and code to optimize performance using the following capabilities:

- complete internal and environment settings inventory
- system call tracing
- statistical, flat profiling of processor core usage from the thread level to run-time tasks down to the method level
- VM proxy interaction analysis

General Guidelines for Capturing Data

Take two data samples within a period of time (for example, in a 20 sec. interval) to calculate performance counter increments.

Azul Support

If providing data for problem resolution for a case with Azul customer advocacy, capture the requested data in its entirety and send it by e-mail or FTP upload for analysis., including the timestamp (if available),

System Call Tracing

A system call trace is a record of every system call made that goes through the proxy. The log file output includes the calling thread ID, date and time, call arguments, and return value. The arguments help to identify the target file or directory for file I/O calls, and lists the peer host and port for socket I/O calls.

Statistical Tick Profiler

The tick profiler displays how processor core time is spent between various software execution units (for example, threads, methods, and run-time tasks). The Azul tick profiler differs from other profilers as follows:

- It is a statistical, flat (not tree-based) profiler. It does not capture every processor core cycle. Instead, for every *n* processor core cycles, it records where in the VM the processor core was running (this is the program counter). It then uses this data to build a flat, non-nested profile.
- The captured data represents a specific time period.
- It profiles both within and outside the Java code. Most Java profilers use byte code instrumentation for Java code.
- It does not profile when threads are blocked on locks. To retrieve information on I/O and monitors, go to the System Call Profile and the Monitors window.



NOTE Because it is a statistical, flat profiler, the Azul tick profiler introduces an extremely small performance impact when running with the VM.

Access the statistical tick profiler through the Ticks window.

Table Sorting

Variable size tables can be sorted by column. Click the column head to sort in ascending order. Click the column head again to resort in descending order. Fixed tables (for example, the [GC Summary Window](#)) cannot be sorted.

Web Browser Support

RTPM is accessible from supported Web browsers with access to the host running the VM proxy. The following browsers are supported:

- Firefox™ 1.0.3 or Mozilla™ 1.5 on all supported operating systems
- Microsoft® Internet Explorer 6 (or later) on Windows 2000 or Windows XP
- Safari™ 2.0 for Mac OS® X

Enable RTPM

Configure the Java command line option `-PX:RTPMPort=<port>` to enable RTPM on a host machine, as shown in [Table 4-1](#).

Table 4-1 Required Java Command Line Options for RTPM

Java Command Option	Description
<code>-PX:RTPMPort=<port></code>	<p>Enables the REALTime Performance Monitor and allows the embedded HTTP server to listen on the defined port. Valid options for port are:</p> <ul style="list-style-type: none"> • <code>off</code> (default): disables the REALTime Performance Monitor. • <code><port></code>: port number for the REALTime Performance Monitor to use on the host. • <code>any</code>: the REALTime Performance Monitor uses any free port. This setting is useful when starting multiple instances of the REALTime Performance Monitor for multiple VMs. <p>Once enabled, you can view the selected port number in the CPM Console. Refer to the section, “Viewing the REAL Time Performance Monitor Port Number,” in Chapter 5, Applications, in the <i>Compute Pool Manager Guide</i> for more information.</p>

Supply a port number or set this option to `any` to enable RTPM.

User Authentication and Authorization

The default user authentication and authorization settings grant any user the highest access level. The command line options listed in [Table 4-2](#) configure authentication and authorization and describe the heap memory profiler option. Authentication option settings cannot be changed while the VM is running.

Table 4-2 Optional Java Command Line Options for RTPM

Java Command Option	Description
<code>-PX:RTPMAuthentication=<authentication method></code>	<p>Controls user authentication. Multiple authentication methods can be configured. Valid options are:</p> <ul style="list-style-type: none"> • (default) <code>off</code>: allows HTTP access without a log in (no authentication). • <code>os</code>: requires HTTPS access using POSIX login/password authentication. • <code>vm</code>: requires HTTPS access using the JDK password database file at <code>\$JAVA_HOME/etc/rtpm/passwd</code>. • <code><path></code>: requires HTTPS access using an a password database file stored in an alternate location. Use the full path, which can include logical names. <p>Specifying this option has the same requirements as using the “<code>vm</code>” option, except that the file location must be specified with the <code>-f</code> flag when using the password database utility to add users and groups.</p>
<code>-PX:RTPMAuthorization=<authorization rule></code>	<p>Controls user authorization. Valid options are:</p> <ul style="list-style-type: none"> • <code><1–9></code>: defines the level of functionality available to any authenticated user, or if no authentication is used, to anyone. Levels 6 and above allow access to application data. “9” is default. • <code>vm</code>: directs the proxy to use the JDK security configuration file for access rules at <code>\$JAVA_HOME/etc/rtpm/security.conf</code>. This file applies the same security rules to all VMs started from that <code>JAVA_HOME</code>. • <code><path></code>: specifies an alternate access rules file. Use the full path, which is parsed using the local proxy environment and can include logical names.

Table 4-2 *Optional Java Command Line Options for RTPM (Continued)*

Java Command Option	Description
-PX:RTPMInterface=<ip>	Instructs the VM proxy on which network interface to accept HTTP connections. The default value is <0.0.0.0> (the wildcard interface). This setting means that connections on any interface are accepted by RTPM. When the proxy host has more than one interface (for example, in out-of-band configurations where the proxy is connected to a data network and a management network), by default the access to RTPM is opened on both networks. To ensure that RTPM connections are made only from workstations connected to the management network, specify the IP address of that interface and RTPM will only bind to it. To only accept connections from the local host, specify the loopback interface, <127.0.0.1>.
-XX:+ProfileLiveObjects	Enables the heap memory profiler, which displays class allocation information in the Memory tab (default = enabled).
-XX:PrintGCHistory= <i>N</i>	Set the number of historical PGC cycle reports to save for display in the REALTime Performance Monitor (default = 50).
-XX:GCWarningHisor= <i>N</i>	Set the number of historical PGC warnings to save for display in the REALTime Performance Monitor (default = 50).

When authentication is enabled, the REALTime Performance Monitor uses HTTPS for all communications and requires a user name and password to establish a session. When authentication is disabled, the REALTime Performance Monitor uses HTTP, does not require a user name or password, and does not track user sessions.

User Databases

The following two user, group, and password databases are supported for REALTime Performance Monitor authentication:

- The standard POSIX database located at /etc/passwd allows integration with Lightweight Directory Access Protocol (LDAP).
- A non-OS authentication password database:
 - A default JDK-level database stored at \$JAVA_HOME/etc/rtpm/passwd in the Azul JDK.
 - A launch-specific file option. Use the full path, which can include logical names.

By default, authentication is disabled. User authentication can be configured to use the REALTime Performance Monitor password database, the POSIX database, or an alternate password database file.

Password Database Utility

The rtpm-passwd command line utility allows adding and deleting users in the REALTime Performance Monitor password database and adding users to groups. This password database is used for authentication if “vm” is configured as the authentication method in the security.conf file. The utility is located at \$JAVA_HOME/bin/rtpm-passwd. Use the commands listed in [Password Database Command Line Options](#) [Table 4-3](#) to control the password database utility.

Table 4-3 *Password Database Command Line Options*

Command Line	Description
\$JAVA_HOME/bin/rtpm-passwd <username>	Add users to the REALTime Performance Monitor password database.
\$JAVA_HOME/bin/rtpm-passwd -d <username>	Delete users from the REALTime Performance Monitor password database.

Table 4-3 Password Database Command Line Options (Continued)

Command Line	Description
<code>\$JAVA_HOME/bin/rtpm-passwd -g <groupname> <username></code>	Add a user to a group. Each group requires at least one user.
<code>\$JAVA_HOME/bin/rtpm-passwd -gd <groupname> <username></code>	Delete a user from a group. If the user deleted is the only user in the group, the group is also deleted.
<code>\$JAVA_HOME/bin/rtpm-passwd -g <groupname></code>	Delete a group.

Session Time Out

By default, the REALTime Performance Monitor uses an 1,800 second (30 minute) inactivity time out. After this period, the login screen displays, prompting the user to log in. The session time-out interval can be changed in the security.conf file. Refer to [Authorization](#) for more information.

Authorization

The `$JAVA_HOME/etc/rtpm/security.conf` file controls authorization. Change the `-PX:RTPMAuthorization` option to use a different authorization control file, as shown in [Table 4-1](#).

The default security.conf file assigns an authorization level to the VM owner, and denies authorization to all other users. The file can be modified to define authorization levels for additional users and groups. The security.conf file also defines the session time-out interval.

Authorization Levels and Users

The authorization level is a value from 1–9 that specifies the features in the REALTime Performance Monitor that a user can access. Authorization levels 6 and above allow the user or group to access application data (such as live objects within the VM); levels 5 and below do not.

The following user types can be specified:

- **Owner:** the user who launched the VM. By default, the owner is granted authorization level 9 (full authorization).
- **User:** the user name and password are defined in either the REALTime Performance Monitor database or the standard POSIX database, depending on the authorization scheme configured in `-PX:RTPMAuthorization`.
- **Group:** authorization level assigned to all members in a group. If using a POSIX database, this refers to the POSIX groups. If using the REALTime Performance Monitor database, groups are defined and members added to groups using the [Password Database Utility](#).

Syntax of security.conf

Use the entries “enable authentication” or “disable authentication” in the security.conf file to control authentication. By default, authentication is disabled. If multiple entries exist in the file, the first encountered takes precedence. Authentication cannot be enabled or disabled while the VM is running. The password and security.conf files are re-read each time a user attempts to login.

When authentication is enabled, a simple rule change grants or denies access to users and groups. The first rule that matches a user trying to login is final; no further rules are evaluated. The syntax of grant rules is `grant <user|owner|group> level <1-9>`. The syntax to deny rules is `deny <all|user|group>`. There is an implied “deny all” at the end of the rules chain if it is not specified explicitly. Precede comments with the '#' character.

In [Example 4-1](#), the default `security.conf` file grants full authorization (level 9) to the VM owner and denies access to all other users.

Example 4-1 Default `security.conf` File

```
grant owner level 9
deny all
```

In [Example 4-2](#), a `security.conf` file is modified to grant full authorization (level 9) to the VM owner, grant level 5 to user john (john can view all non-application data), grant level 6 to group administrators (admins can view application data), and deny access to all other users. Additionally, the session time-out interval is 3600 seconds (1 hour).

Example 4-2 Modified `security.conf` File

```
set session timeout 3600
grant owner level 9
grant user john level 5
grant group admins level 6
deny all
```

Log In to RTPM

Access the REALTime Performance Monitor using the following addresses in a Web browser window:

- If authentication is enabled, access the REALTime Performance Monitor at the following URL:

`https://<proxy address of host>:<port number>`

For example: `https://host123:1600`

- If authentication is disabled, access the REALTime Performance Monitor at the following URL:

`http://<proxy address of host>:<port number>`

For example: `http://host123:1600`

Certificate Security Warnings

The following security warnings display in the Web browser when RTPM starts ([Figure 4-1](#)):

- The REALTime Performance Monitor uses a self-signed certificate that cannot be validated.
- The common name of the application, “RTPM,” does not match the host name of the machine.



Figure 4-1 Security Certificate Warnings

Click Yes to accept the included certificate. See the following section for information on using a user-supplied certificate.

Provide a Secure User Certificate

To provide increased security and avoid the display of security warnings in the Web browser, install signed certificates from a customer certificate authority or from a third-party certificate provider such as VeriSign. This requires providing one each of the following files per proxy host installation:

- security.cert (a signed certificate)
- security.key (a private key)

These files must be stored in the \$JAVA_HOME/etc/rtpm directory.

Login Window

On successful connection to the host, the Login screen displays (Figure 4-2).

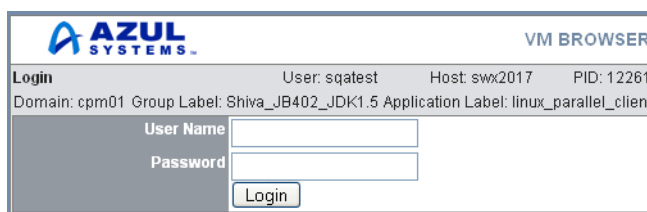


Figure 4-2 Example Login Dialog

Table 4-4 describes the Login window elements.

Table 4-4 Process Window Elements

Element	Description
User	The user name or a numeric ID for the user.
Host	The host of the VM proxy.
PID	A numeric ID for the host proxy process.

Perform the following steps to log in to RTPM.

Step 1 Enter a user name and password.

Step 2 Click Login.

Graphical User Interface

The REALTime Performance monitor has the following main tabs.

- [Configuration Tab](#)
- [Threads Tab](#)
- [I/O Tab](#)
- [Ticks Tab](#)
- [Monitors Tab](#)
- [Memory Tab](#)
- [Compiler Tab](#)
- [Settings Tab](#)

Each main tab has sub tabs relating to the task.

Configuration Tab

The Configuration tab of the REALTime Performance Monitor contains the Process, Environmental variables, and HotSpot flags tabs.

Process Window

The Process window (Figure 4-3) provides a detailed view of all the process settings for the VM. This tab displays after log in.

Configuration	Azul Support	Threads	I/O	Ticks	Monitors	Memory	Compilers	Settings
Process	Environment variables	HotSpot flags						
Configuration - Process								
Name	Value							
Version	Java HotSpot(TM) 64-Bit Tiered VM (1.6.0_07-AVM_2.5.0.0-57-product) for aztek-vega2, built on May 15 2009 15:23:03 by buildmaster with gcc-3.2.1							
Command line	<pre>JNI_CreateJavaVM() -Djava.class.path=. -PX:PolicyVhost=192.168.102.200 -PX:CPMDomain=podv2demo - PX:AppLabel=geneticfinan ce -PX:RTPMPort=any -PX:RTPMAuthorization=9@ -XX:+DisableExplicitGC - Djava.util.logging.config.file=testLogging.properties -Djava.class.path=./patch:wrapper.jar:genfin.jar:commons-logging-1.1.jar:junit-4.1.jar:mysqljdbc.jar:ws-commons- util- 1.0.2.jar:xmlrpc-common-3.1.jar:xmlrpc-server-3.1.jar:xmlrpc-client-3.1.jar:java_memcached-release_2.0.1- GF.jar:memcache Lib-1-0-GF.jar:./patch -Xmx12g -Dsun.java.command=com.gf.pool.StatefulEvolution_Test 1000 1000 state3.dat distributed gfsl.geneticfinance.com 8991 0 false 50 -Dsun.java.launcher=SUN_STANDARD -Dsun.java.launcher.pid=2209</pre>							
Prepended flags / filename								
Appended flags / filename								

Figure 4-3 Process Window

Table 4-5 describes the Process window elements.

Table 4-5 Process Window Elements

Element	Description
Version	The Azul VM version.
Prepended flags / filename	The list of option flags from the identified file that were inserted into the front of the java command line.
Appended flags / filename	The list of option flags from the identified file that were appended to the end of the java command line.
Current working directory	The current working directory.
Java Home	The Java Home path.
Class path	The path to the class.
Boot class path	The path to the boot class.
Extensions directories	The directories for extensions.
Endorsed directories	The directories for classes implementing newer versions of endorsed standards.
Library path	The native library path.
Boot library path	The boot library path.
Open descriptor count limit	The limit in bytes for open file descriptors for the VM proxy.
Core file size limit	The maximum size in bytes for a core dump file.

Table 4-5 *Process Window Elements (Continued)*

Element	Description
DirectPath state	Displays the actual state of DirectPath for the application instance (see DirectPath on page 2), as follows: <ul style="list-style-type: none"> • Enabled • Disabled. The flag -PX:-UseDirectPath was specified. • Disabled. Disallowed by CPM policy. • Disabled. Authentication failed with DirectPath subsystem (lsockdb). • Disabled. Unable to connect to DirectPath subsystem (lsockdb). Incompatible AVX version?
Argument	The command line arguments.
Command	The Java class and its parameters.

Environmental Variables Window

The Environmental variables window ([Figure 4-4](#)) provides a detailed view of all environmental variables set by the shell that launched the VM proxy.

Configuration	Azul Support	Threads	I/O	Ticks	Monitors	Memory	Compilers	Settings
Process	Environment variables	HotSpot flags						
Environment variables								
Name	Value							
_	/podtools/tools/avm/x86/azul-jdk1.6.0_07-2.5.0.0-57/bin/java							
ENA_CODESET_TYPE	WESTERN							
ENA_INSTALL_TYPE	Interstage							
FAS_ROOT_DIR	/opt/FJSSVena/Enabler/server							
FONTCONFIG_PATH	/podtools/tools/avm/x86/azul-jdk1.6.0_07-2.5.0.0-57/jre/lib/X11/etc/fonts							
G_BROKEN_FILENAMES	1							
HISTCONTROL	ignoredups							
HISTSIZE	1000							
HOME	/home/babak							
HOSTNAME	podlx01.pod.azurelinux.com							
IBPATH	/usr/bin							
INPUTRC	/etc/inputrc							
JVM_HPROF_TXT_PATH	/podtools/tools/avm/x86/azul-jdk1.6.0_07-2.5.0.0-57/jre/lib/jvm.hprof.txt							
JVM_JCOV_TXT_PATH	/podtools/tools/avm/x86/azul-jdk1.6.0_07-2.5.0.0-57/jre/lib/jvm.jcov.txt							

Figure 4-4 *Environmental Variables Window*

HotSpot Flags Window

The HotSpot flags window (Figure 4-5) displays all -XX flags the VM uses, their description, and current values. This screen is for advanced diagnostics and is not intended for beginning users. Azul Customer Advocacy may instruct users to look up flag values in this window.

Configuration | [Azul Support](#) | [Threads](#) | [IO](#) | [Ticks](#) | [Monitors](#) | [Memory](#) | [Compilers](#) | [Settings](#)

Process | [Environment variables](#) | [HotSpot flags](#)

Configuration - HotSpot flags

Type	Name	Value	Description
Boolean	AbortOnOOM	false	Cause core dump to occur when internal VM memory is exhausted
Boolean	AbortOnWeblogicMuxerError	false	Abort when a Weblogic muxer warning or error occurs
Unsigned integer	AdaptivePermSizeWeight	20	Weight for perm gen exponential resizing, between 0 and 100
Unsigned integer	AdaptiveSizeDecrementScaleFactor	4	Adaptive size scale down factor for shrinking
Unsigned integer	AdaptiveSizeMajorGCDecayTimeScale	10	Time scale over which major costs decay
Unsigned integer	AdaptiveSizePausePolicy	0	Policy for changing generation size for pause goals
Unsigned integer	AdaptiveSizePolicyCollectionCostMargin	50	If collection costs are within margin, reduce both by full delta
Unsigned integer	AdaptiveSizePolicyInitializingSteps	20	Number of steps where heuristics is used before data is used
Unsigned integer	AdaptiveSizePolicyOutputInterval	0	Collector interval for printing information, zero => never
Unsigned integer	AdaptiveSizePolicyWeight	10	Weight given to exponential resizing, between 0 and 100
Unsigned integer	AdaptiveSizeThroughPutPolicy	0	Policy for changing generation size for throughput goals
Unsigned integer	AdaptiveTimeWeight	25	Weight given to time in adaptive policy, between 0 and 100
Boolean	AdjustConcurrency	false	call thr_setconcurrency at thread create time to avoid LWP starvation on MP

Figure 4-5 HotSpot Flags Window

Table 4-6 describes the HotSpot flags window elements.

Table 4-6 HotSpot Flags Window Elements

Element	Description
Type	The type of flag. Possible values are: <ul style="list-style-type: none"> Boolean: the flag requires a boolean value of true (+) or false (-). Integer or Unsigned Integer: the flag requires an integer value. String: the flag requires a string.
Name	The name of the flag.
Value	The value configured for the flag.
Description	The description of the flag.

Threads Tab

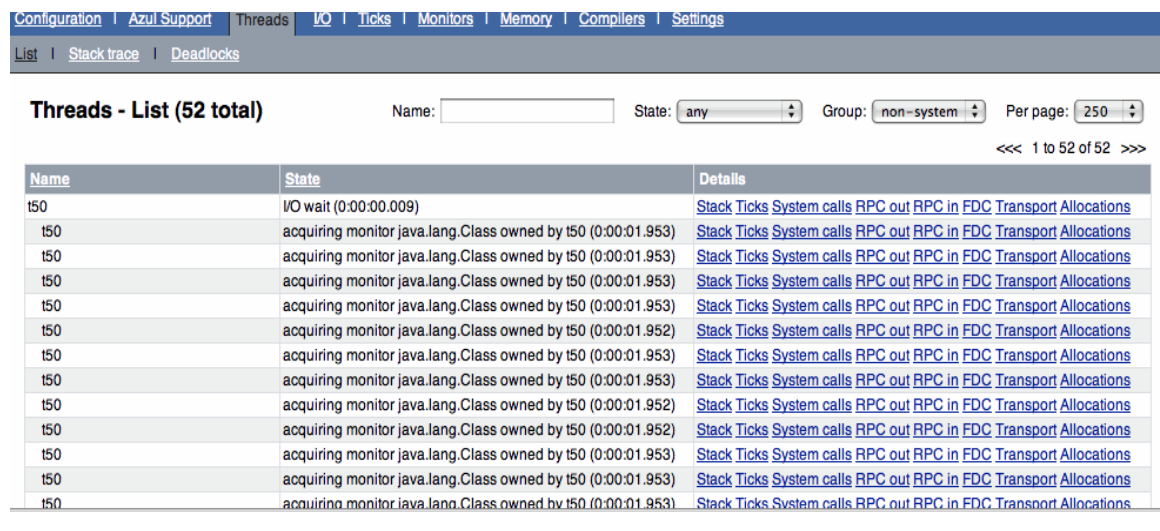
The Threads tab (Figure 4-6) contains the following sub-tabs:

- Threads List window shows all threads, status of each thread, and links to detailed information about a thread
- Stack trace window provides information on where threads are blocked
- Deadlocks window displays threads that are deadlocked

Threads List Window

Examine this window to check for thread states and for links to thread-level profiling. For example, the thread state for locked threads is listed as “acquiring monitor;” J2EE Worker threads waiting to serve requests from the web tier are listed as “waiting on monitor;” JDBC connection pool threads waiting for database query responses are listed as “I/O wait.”

Refresh this window to view thread progress. Click the Name and State column titles to sort the table.



Threads - List (52 total) Name: State: Group: Per page:

<<< 1 to 52 of 52 >>>

Name	State	Details
t50	I/O wait (0:00:00.009)	Stack Ticks System calls RPC out RPC in FDC Transport Allocations
t50	acquiring monitor java.lang.Class owned by t50 (0:00:01.953)	Stack Ticks System calls RPC out RPC in FDC Transport Allocations
t50	acquiring monitor java.lang.Class owned by t50 (0:00:01.953)	Stack Ticks System calls RPC out RPC in FDC Transport Allocations
t50	acquiring monitor java.lang.Class owned by t50 (0:00:01.953)	Stack Ticks System calls RPC out RPC in FDC Transport Allocations
t50	acquiring monitor java.lang.Class owned by t50 (0:00:01.953)	Stack Ticks System calls RPC out RPC in FDC Transport Allocations
t50	acquiring monitor java.lang.Class owned by t50 (0:00:01.952)	Stack Ticks System calls RPC out RPC in FDC Transport Allocations
t50	acquiring monitor java.lang.Class owned by t50 (0:00:01.953)	Stack Ticks System calls RPC out RPC in FDC Transport Allocations
t50	acquiring monitor java.lang.Class owned by t50 (0:00:01.953)	Stack Ticks System calls RPC out RPC in FDC Transport Allocations
t50	acquiring monitor java.lang.Class owned by t50 (0:00:01.952)	Stack Ticks System calls RPC out RPC in FDC Transport Allocations
t50	acquiring monitor java.lang.Class owned by t50 (0:00:01.952)	Stack Ticks System calls RPC out RPC in FDC Transport Allocations
t50	acquiring monitor java.lang.Class owned by t50 (0:00:01.953)	Stack Ticks System calls RPC out RPC in FDC Transport Allocations
t50	acquiring monitor java.lang.Class owned by t50 (0:00:01.953)	Stack Ticks System calls RPC out RPC in FDC Transport Allocations
t50	acquiring monitor java.lang.Class owned by t50 (0:00:01.953)	Stack Ticks System calls RPC out RPC in FDC Transport Allocations
t50	acquiring monitor java.lang.Class owned by t50 (0:00:01.953)	Stack Ticks System calls RPC out RPC in FDC Transport Allocations

Figure 4-6 Threads List Window

Click the **Stack** link inline with the desired thread to determine the status of a thread and where it is blocked. Refer to [Stack Trace Window](#).

Table 4-7 describes the Threads List window elements.

Table 4-7 Threads List Window Elements

Element	Description
Name	Lists the name of the thread as assigned by the application.

Table 4-7 *Threads List Window Elements (Continued)*

Element	Description
State	<p>The most common states for threads are:</p> <ul style="list-style-type: none"> • running: the thread is executing on a processor core. • I/O wait: the thread is waiting for socket or file I/O. • waiting on monitor: the thread is blocked in <code>java.lang.Object.wait()</code> awaiting notification. • low memory detection. • acquiring monitor: the thread is waiting to acquire a synchronization monitor. • acquiring and releasing monitor: the thread requires and releases a synchronization monitor. This typically happens when the critical section of the monitor is extremely small. • sleeping: the thread executed <code>java.lang.Thread.sleep()</code>. <p>Other states, such as idle, semaphore wait, waiting on VM monitor 'Threads_lock', or Waiting on Monitor 'SLT' lock indicate threads that are internal to the Azul VM. These states are not normally useful for analysis of application performance. Also, during start up, compiler threads may be running. The time that the thread has spent since entering the state is provided in parenthesis after the waiting state indicators.</p>
Details	<p>Provides links to a stack trace, tick profile, system call profile, and object profile for the selected thread. These functions are the same as the regular tick and profile features, except that only information on a particular thread displays. The following links appear inline with threads listed in this window:</p> <ul style="list-style-type: none"> • Stack: displays a thread execution details window. • Ticks: displays a per-thread profile. • System Calls: displays a thread-specific profile window. • RPC In: displays a thread-specific RPC incoming profile window. • RPC Out: displays a thread-specific RPC outgoing tick profile window. • FDC: displays a thread-specific FDC profile window. • Transport: displays a thread-specific transport profile window. • Allocations: displays a thread-level Object profile. This window displays real-time counters for allocated bytes, allocated count, and average object size per object type allocated by the thread since the most recent GC cycle start.

Stack Trace Window

The **Stack trace** window lists all threads and displays if a thread is executing or is blocked (Figure 4-7). This can help to determine locks to change in the application code. To perform a search by thread name, type a string in the **Name** field to display all thread names that include that text string (case sensitive).

The screenshot shows the 'Thread Stack trace' window. At the top, there are tabs for Configuration, Azul Support, Threads, IO, Ticks, Monitors, Memory, Compilers, and Settings. Below these are sub-tabs for List, Stack trace, and Deadlocks. The main title is 'Threads - Stack trace (7 total)'. There are filters for Name, State (set to 'any'), Group (set to 'non-system'), and Per page (set to '100'). A detail level selector shows 'low', 'medium' (selected), and 'high'. A pagination control shows '<<< 1 to 7 of 7 >>>'. The selected thread is 'Thread "150": I/O wait (0:00:00.026)'. Below this, the address is '0x0000080015cfa00, Priority: 5, Object blocked: 37,268 ms, Object wait: 0 ms, CPU wait: 2 ms, I/O wait: 78 ms, CPU: 266 ms'. A list of stack frames follows, including methods like 'java.net.SocketInputStream.socketRead0', 'com.mysql.jdbc.util.ReadAheadInputStream.fill', and 'com.mysql.jdbc.ConnectionImpl.createNewIO'.

Figure 4-7 Thread Stack trace Window

Click the thread links in the Stack trace window to display a thread execution details window. Use the **State** drop-down menu to only display threads in a certain state (for example, running, I/O wait, acquiring monitor, and so on). Use the Per page drop-down menu to determine the number of items to display on this page.

Refresh this screen to view the progress of the thread. The most relevant information displays at the top of the screen.

Deadlocks Window

The Deadlocks window (Figure 4-8) lists detected deadlock cycles. Threads involved for each cycle are listed with the monitor type the thread is trying to acquire and the thread that is the current owner.

The screenshot shows the 'Deadlocks' window. It has the same top navigation as Figure 4-7. The sub-tab 'Deadlocks' is selected. The title is 'Threads - Deadlocks (1 total)'. A list shows a single deadlock cycle: '1. Thread-C is waiting to acquire a java.lang.Object which is owned by Thread-A; Thread-A is waiting to acquire a java.lang.Object which is owned by Thread-B; Thread-B is waiting to acquire a java.lang.Object which is owned by Thread-C'. The footer shows the date 'Tue, 06 Feb 2007 22:39:50 GMT' and the copyright 'Copyright (c) 2004-2006 Azul Systems Inc. All rights reserved.'

Figure 4-8 Deadlocks Window

I/O Tab

The I/O tab provides data for understanding the number, type, and frequency of calls made to the proxy host. Call profiles in this tab collect and summarize all calls made from the start of the program or from the last reset. Some metrics are only calculated for the most-recent time period (moving window). The I/O window contains the following sub tabs:

- System call profile
- Outgoing RPC profile
- Incoming RPC profile
- FDC profile
- FDC stats
- Transport profile
- Open sockets
- Open files
- Buffers
- Trace
- Trace settings

System Call Profile Window

This window (Figure 4-9) profiles the system calls made from the Java domain.

Configuration Azul Support Threads I/O Ticks Monitors Memory Compilers Settings										
System call profile Outgoing RPC profile Incoming RPC profile FDC profile FDC stats Transport profile Open sockets Open files Buffers Trace Trace settings										
System call profile - Summary							Download as csv		Reset System Call Profile	
Name	Count	Duration (ms)					Rate moving avg (calls/s)		Duration moving avg (ms)	
		Mean	Stddev	Min	Max	Sum	1 min	5 min	1 min	5 min
accept	8	2.72	0.34	2.34	3.56	21.72	0.02	0.01	2.62	2.69
available	336,672	0.00	0.02	0.00	11.14	38.76	8.18	6.52	0.00	0.00
close	3,195	10.00	47.04	0.37	728.59	31,953.27	0.23	0.12	23.48	17.47
closedir	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
connect	3,041	78.46	76.96	25.37	3,028.51	238,587.82	0.25	0.11	43.20	65.43

Figure 4-9 System Call Profile Window

Table 4-8 describes the System call profile window elements. Click column titles to sort by relevance.

Table 4-8 System Call Profile Window Elements

Element	Description
Name	The name of the system call.
Count	The number of system calls.
Duration (ms): Mean	The mean of the collected system call durations.

Table 4-8 *System Call Profile Window Elements (Continued)*

Element	Description
Duration (ms): Std Dev	The standard deviation of the collected system call durations.
Rate moving averages (calls/sec)	An average of calls per second over a 1 minute and 5 minute time period.
Duration moving average (ms)	Provides the average time duration, in ms, over a 1 minute and 5 minute time period.

Examine System Call Profile Data

Click **Reset System Call Profile** to clear the system call profile data. This allows new calculations of the count, mean, and standard deviation. The moving averages figures are not meaningful until sufficient samples are collected. For example, the one-minute moving average only gives meaningful data several minutes after the reset button is clicked.

- The *mean* parameter shows how long it takes for traffic to travel round-trip from the application to the proxy host.
- The *count* parameter shows the number of operations performed.

Other important statistics to examine are the number of sends, receives, and accepts. If there are more than 200 accepts per second, there may be a bottleneck if a single thread is accepting connections from clients. 500–1000 accepts per second indicates a saturated condition. Check the thread list to determine if a single thread is performing the accepts. The Thread List can be filtered by thread state, facilitating determination of threads in a certain state.

Outgoing and Incoming RPC Profile Windows

These windows profile the calls made from the AVM engine to the proxy (outgoing) and from the proxy to the engine (incoming). These calls are not directly made from the Java domain, but are often the result of an original Java call. RPC calls include JNI invocations and callbacks.

Figure 4-10 shows the Outgoing RPC profile window. All outgoing remote procedure calls are listed in this window.

Configuration Azul Support Threads I/O Ticks Monitors Memory Compilers Settings										
System call profile Outgoing RPC profile Incoming RPC profile FDC profile FDC stats Transport profile Open sockets Open files Buffers Trace Trace settings										
Outgoing RPC profile - Summary										Download as .csv Reset Outgoing RPC Profile
Name	Count	Duration (ms)					Rate moving avg (calls/s)		Duration moving avg (ms)	
		Mean	Stddev	Min	Max	Sum	1 min	5 min	1 min	5 min
proxy_canonicalize	21	0.50	0.07	0.37	0.56	10.49	0.00	0.00	0.00	0.00
proxy_close	145	176.66	85.50	0.37	465.69	25,615.66	0.00	0.00	174.04	174.20
proxy_echo	596,889	0.42	0.63	0.19	111.60	252,946.98	9.96	9.96	0.43	0.41
proxy_tstat	20,173	0.74	15.21	0.28	2,102.74	15,012.57	0.29	0.33	0.54	0.58
proxy_getcwd	3	0.50	0.12	0.37	0.66	1.50	0.00	0.00	0.40	0.39

Figure 4-10 *Outgoing RPC Profile Window*

Figure 4-11 shows the Incoming RPC profile window. All incoming remote procedure calls are listed in this window. The durations shown for incoming RPC calls include only the processing duration of Azul, and exclude the network round-trip time for the call.

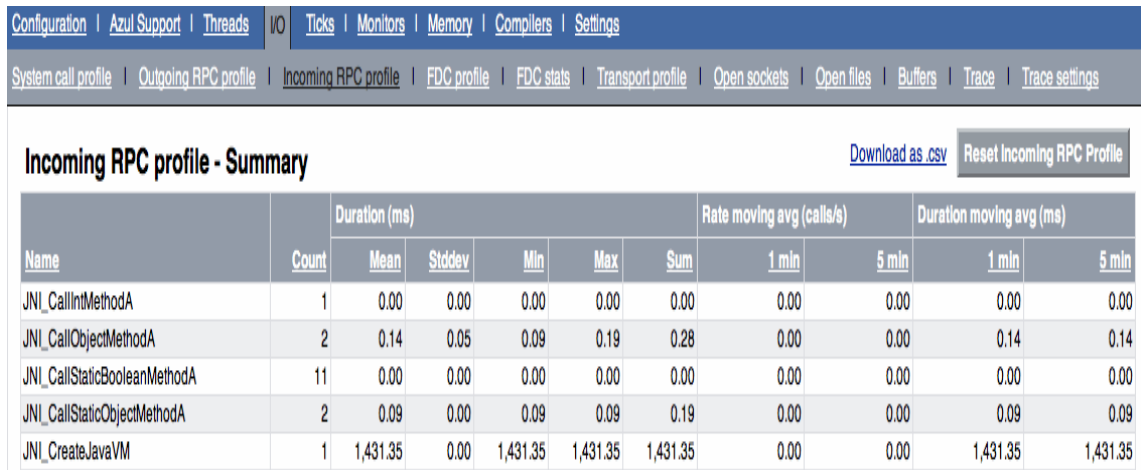


Figure 4-11 Incoming RPC Profile Window

Table 4-9 describes the window elements for both the Outgoing RPC profile and Incoming RPC profile windows. Click column titles to sort by relevance.

Table 4-9 RPC Profile Window Elements

Element	Description
Name	The name of the proxy call.
Count	The number of proxy calls.
Duration (ms)	<ul style="list-style-type: none"> Mean: the mean of the collected proxy call durations. Stddev: the standard deviation of the collected proxy call durations. Min: the minimum of the collected proxy call durations. Max: the maximum of the collected proxy call durations. Sum: the sum of the collection proxy call durations.
Rate moving avg (calls/s)	An average of calls per second over a 1 minute and 5 minute time period.
Duration moving avg (ms)	Provides the average time duration, in ms, over a 1 minute and 5 minute time period.

FDC Profile Window

Figure 4-12 shows the File Data Cache profile window.

Configuration Azul Support Threads IO Ticks Monitors Memory Compilers Settings										
System call profile Outgoing RPC profile Incoming RPC profile FDC profile FDC stats Transport profile Open sockets Open files Buffers Trace Trace settings										
File Data Cache profile - Summary										Download as .csv Reset File Data Cache Profile
Name	Count	Duration (ms)					Rate moving avg (calls/s)		Duration moving avg (ms)	
		Mean	Stddev	Min	Max	Sum	1 min	5 min	1 min	5 min
vf_close	146	204.40	95.18	0.47	728.59	29,842.67	0.00	0.00	200.45	201.63
vf_fstat	23	0.17	0.38	0.00	1.40	3.84	0.00	0.00	0.79	0.84
vf_open	172	10.35	22.01	0.47	273.57	1,780.48	0.00	0.00	8.71	9.16
vf_read	876,999	0.00	0.05	0.00	33.71	579.35	0.01	0.01	1.50	1.39
vf_stat	2,837	0.03	0.20	0.00	5.80	91.75	0.00	0.13	0.00	0.00
vf_write	86,070,715	0.00	0.07	0.00	333.59	11,229.63	865.57	1,743.25	0.00	0.00
Total	86,950,892	0.00	0.30	0.00	728.59	43,527.73	865.58	1,743.40	0.00	0.00

Figure 4-12 File Data Cache Window

Table 4-10 describes the File data cache window elements.

Table 4-10 File Data Cache Window Elements

Element	Description
Name	Name of the event.
Count	The number of events recorded.
Duration (ms)	An average of calls per second over a 1 minute and 5 minute time period.
Rate moving avg (calls/s)	
Duration moving avg (ms)	

FDC Stats Window

The FDC stats window provides an overview of the performance of the cache. It allows the user to understand application behavior related to file operations. Increase the FDC cache size if this data shows a high ratio of misses to hits.

[Configuration](#)
[Threads](#)
[IO](#)
[Ticks](#)
[Monitors](#)
[Memory](#)
[Compilers](#)
[Settings](#)

[System call profile](#)
[Outgoing RPC profile](#)
[Incoming RPC profile](#)
[FDC profile](#)
[FDC stats](#)
[Transport profile](#)
[Open sockets](#)
[Open files](#)
[Buffers](#)
[Trace](#)
[Trace settings](#)

File Data Cache stats

File name cache

Event	Count	Description
Good hits	212,784,061	The file name was found in the cache.
Negative hits	0	The name was searched for in the cache and the file was not found.
Bad hits	0	A cache entry was found but it was not usable.
False hits	0	A cache entry was found but it was not usable.
Misses	11,019,177	The cache was searched for a file but it was not found.
Stale Hits	668	A hit was found in the cache but the cache information is too old.

File page cache

Event	Count	Description
Read hits	126,264	An information block is already in the cache.
Read misses	9,605	Data in a file is not local file content; the data is on the proxy machine.
Write hits	14,346	The file area you are writing into is local to the Azul VM engine.
Overwrites	0	The page was found in the cache but the user request overwrote it.
Write grows	1,640	Contents are not in the cache because data was appended to a file.
Write misses	7	Contents are not in a cache because data was written to the middle of a file.
Page waits	0	The thread doing file IO had to wait for a free page.

File node queues

Active		Inactive	
Number	Length	Number	Length
0	7	0	100
1	8		
2	8		
3	6		
4	7		
5	6		
6	7		
7	9		

Figure 4-13 FDC Stats Window

Table 4-11, Table 4-12, and Table 4-13 describe the elements that appear in the file name cache, file page cache, and file node queues tables in the FDC Stats window.

Table 4-11 FDC Stats Window Elements – File Name Cache Table Elements

Element	Description
Event	Possible events are: <ul style="list-style-type: none"> • Good hits: the page being read is already in the cache. • Negative hits: the page being read was not in the cache and had to be read remotely from the proxy machine. • Bad hits: a cache entry was found, but was not usable. • False hits: a cache entry was found, but was not usable. • Misses: the cache was searched for a file, but it was not found. • Stale hits: a hit was found in the cache, but the cache information is too old.
Count	The number of events recorded.
Description	A description of the event.

Table 4-12 FDC Stats Window Elements – File Page Cache Table Elements

Element	Description
Event	Possible events are: <ul style="list-style-type: none"> • Read hits: the page being read is already in the cache. • Read misses: the page being read was not in the cache and had to be read remotely from the proxy machine. • Write hits: the page being written is in the cache. • Overwrites: the page was found in the cache, but a user request overwrote it. • Write grows: contents are not in the cache because data was appended to a file. • Write misses: contents are not in a cache because data was written to the middle of a file.
Count	The number of events recorded.
Description	A description of the event.

Table 4-13 FDS Stats Window Elements – File Node Queues Table Elements

Element	Description
Active	Number and length of active file node queues.
Inactive	Number and length of inactive file node queues.

Transport Profile Window

Provides a list of low-level I/O transport operations that correspond to network I/O. [Figure 4-14](#) shows the Transport profile window.

Configuration Azul Support Threads IO Ticks Monitors Memory Compilers Settings										
System call profile Outgoing RPC profile Incoming RPC profile FDC profile FDC stats Transport profile Open sockets Open files Buffers Trace Trace settings										
Transport profile - Summary										Download as .csv Reset Transport Profile
Name	Count	Duration (ms)					Rate moving avg (calls/s)		Duration moving avg (ms)	
		Mean	Stddev	Min	Max	Sum	1 min	5 min	1 min	5 min
atcpn_direct_channel_drain	3,103	0.00	0.00	0.00	0.09	0.09	0.06	0.06	0.00	0.00
atcpn_direct_channel_preclose	3,103	0.00	0.03	0.00	1.69	10.39	0.06	0.06	0.00	0.00
atcpn_direct_channel_read	1,854,029	61.31	36,391.08	0.00	49,550,334.89	113,662,067.90	32.53	30.97	33.73	38.98
atcpn_direct_channel_timeout	342,292	0.00	0.00	0.00	1.50	10.49	8.25	6.20	0.00	0.00
atcpn_direct_channel_write	29,538	0.10	2.08	0.00	66.66	2,844.44	0.74	0.59	0.00	0.02
atcpn_direct_channel_writeall	29,153	0.10	2.21	0.00	76.12	2,850.62	0.69	0.53	0.00	0.03
atcpn_direct_channel_writewait	29,153	0.10	2.21	0.00	76.12	2,850.62	0.69	0.53	0.00	0.03

Figure 4-14 Transport Profile Window

[Table 4-14](#) describes the Transport profile window elements. Click column titles to sort by relevance.

Table 4-14 Transport Profile Window Elements

Element	Description
Name	The name of the proxy call.
Count	The number of proxy calls.
Duration (ms)	<ul style="list-style-type: none"> • Mean: the mean of the collected proxy call durations. • Stddev: the standard deviation of the collected proxy call durations. • Min: the minimum of the collected proxy call durations. • Max: the maximum of the collected proxy call durations. • Sum: the sum of the collection proxy call durations.
Rate moving avg (calls/s)	An average of calls per second over a 1 minute and 5 minute time period.

Table 4-14 *Transport Profile Window Elements*

Element	Description
Duration moving avg (ms)	Provides the average time duration, in ms, over a 1 minute and 5 minute time period.

Open Sockets Window

Shows information and statistics about the open network sockets. Figure 4-15 shows the Open Sockets window.

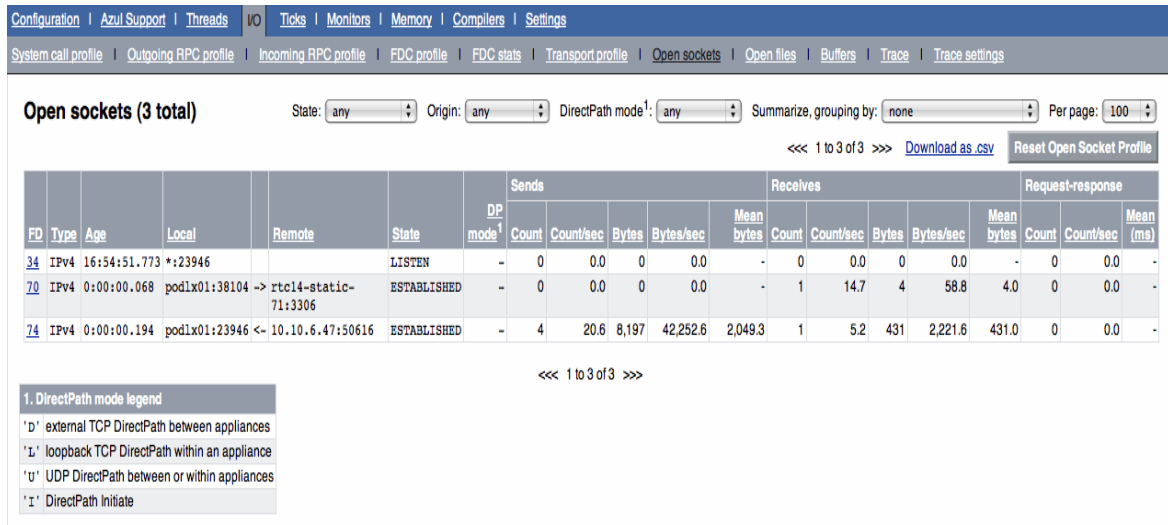
Figure 4-15 *Open Sockets Window*

Table 4-15 describes the Open Sockets window elements. In this table, the phrase, “the application,” refers to the VM instance being monitored by RTPM.

Table 4-15 *Open Sockets Window Elements*

Element	Description
State: <input type="text" value="any"/> <div> <input type="button" value="any"/> <input type="button" value="established"/> <input type="button" value="listen"/> <input type="button" value="closed"/> </div>	Filter the list by the following constraints: <ul style="list-style-type: none"> any = no filter established listen unconnected
Origin: <input type="text" value="any"/> <div> <input type="button" value="any"/> <input type="button" value="<- accept"/> <input type="button" value="-> connect"/> </div>	Filter the list by the following constraints: <ul style="list-style-type: none"> any = no filter <- accept -> connect
DirectPath mode ¹ : <input type="text" value="any"/> <div> <input type="button" value="any"/> <input type="button" value="none"/> <input type="button" value="D, L, or U"/> <input type="button" value="D"/> <input type="button" value="L"/> <input type="button" value="U"/> </div>	Filter the list by the following DirectPath mode constraints (see DP Mode below): <ul style="list-style-type: none"> any = no filter none = no DirectPath mode D, L, or U = filter by all DirectPath modes. D = filter by external DirectPath TCP sockets. L = filter by loopback DirectPath TCP sockets. U = filter by UDP DirectPath sockets.

Table 4-15 *Open Sockets Window Elements (Continued)*

Element	Description
<p>Summarize, grouping by:</p> <div> <div>none</div> <div>none</div> <div>client IP & server IP:port</div> <div>client IP & server IP</div> <div>client IP & server port</div> <div>client IP</div> <div>server IP:port</div> <div>server IP</div> <div>server port</div> <div>local IP & remote IP</div> <div>local IP</div> <div>local port</div> </div>	<p>Summarizes the list using the metrics of multiple entries and grouping them by the following:</p> <ul style="list-style-type: none"> • client IP & server IP:port: lists one entry per all connections from the same client network interface to each unique server port (network interface and port number combination). This option profiles the communication traffic handled between: <ul style="list-style-type: none"> • each server port of this application and each client system separately. • each other server program and client connection from this application. • client IP & server IP: lists one entry per all connections from a client network interface to a server network interface. This option reveals the total network traffic between the local network interface and network interfaces of other systems, where: <ul style="list-style-type: none"> • the application acts as a client and the other system is a server. • the application acts as a server and the other system is a client. • client IP & server port: lists one entry per all connections from the same client network interface to each server port number, regardless of server network interfaces. This option has two use cases: <ul style="list-style-type: none"> • when multiple remote server instances using the same port number exist, they are regarded as one combined server and the entry shows the network traffic it has with the application acting as a client. • profiling the network traffic the application has over connections it accepted on each port number (across any local interfaces) with each client separately. • client IP: lists one entry per all connections from each client network interface to servers. This option summarizes the total traffic stemming from: <ul style="list-style-type: none"> • connections initiated from the application to any server on the network. • connection initiated from each other system to this application. • server IP:port: lists one entry per all client connections to each unique server port: <ul style="list-style-type: none"> • for the application acting as a server, this option profiles the total client communication traffic encountered. • for remote servers, this option profiles the total client communication traffic between the application and each remote server (for example, the total connection pool traffic the application has against a database). • server IP: lists one entry per all client connections to each server network interface. This option summarizes the total traffic stemming from: <ul style="list-style-type: none"> • connections initiated from the application against each remote system (interface). • any client connections accepted by the application. • server port: lists one entry per all client connections to each server port number on every server with this port open. This option has two case uses: <ul style="list-style-type: none"> • when there are multiple remote server instances using the same port number, they are regarded as one combined server and the entry show the total client communication traffic. • profiling the network traffic the application has over connections it accepted on each port number (across any local interfaces) with all client systems. • local IP & remote IP: lists one entry per all connections between the application's local network interface and each remote network interface (regardless of client or server). This option profiles the total traffic the application has with all peer systems (separated per network interface if there are several). • local IP: lists one entry per all connections to and from the application's local client network interface. This option shows a single rolled up entry per local network interface. • local port: lists one entry per all connections to or from each port that the application has open (across multiple existing local interfaces).

Table 4-15 *Open Sockets Window Elements (Continued)*

Element	Description
Per page	Controls the number of list items per page. Options are: 10, 25, 50, 100, 250, 500, or 1000.
<<< next-page and prev-page>>>	Move to the next or previous pages of the list.
Download as csv	Downloads the entire list (all pages) to a comma separated (csv) file.
FD	The file description number.
Type	The type of socket: IPv4 or IPv6.
Age	The time since the FD was last opened.
Local	The local IP port.
Remote	The remote IP port. No value appears for a UDP socket or when no connection has been established.
State	<ul style="list-style-type: none"> • LISTEN • ESTABLISHED • UNCONNECTED (that is, UDP)
DP (DirectPath) Mode	<p>This column is empty for listens or proxy-routed connections.</p> <p>The following codes display for DirectPath connections:</p> <ul style="list-style-type: none"> • D: connection uses DirectPath TCP between appliances. • L: connection uses DirectPath TCP looped back locally on the appliance. • U: connection uses UDP DirectPath between appliances.
Sends	<ul style="list-style-type: none"> • Count: the number of sends since socket was opened. • Count/sec: the mean sends per second since socket was opened. • Bytes: the total number of bytes sent since socket was opened. • Bytes/sec.: the mean bytes sent per second since socket was opened. • Mean bytes: the mean bytes per send since socket was opened.
Receives	<ul style="list-style-type: none"> • Count: the number of receives since socket was opened. • Count/sec: the mean receives per second since socket was opened. • Bytes: the total number of bytes received since socket was opened. • Bytes/sec: the mean bytes received per second since socket was opened. • Mean bytes: the mean bytes per receive since socket was opened.
Request response	<ul style="list-style-type: none"> • Count: the number of requests-response exchanges since socket was opened. • Count/sec: the mean number of requests-response exchanges per second since the socket was opened. • Mean: the mean time between request and response, measured since the socket was opened.

Open Files Window

Displays a table of all open process files of any type. [Figure 4-16](#) shows the Open Files window.

Configuration | Threads | IO | Ticks | Monitors | Memory | Compilers | Settings

System call profile | Outgoing RPC profile | Incoming RPC profile | FDC profile | FDC stats | Transport profile | Open sockets | Open files | Buffers | Trace | Trace settings

Open regular files (56 total)

Per page: 100

<<< 1 to 56 of 56 >>> [Download as .csv](#)

FD	Type	Age	Mode	Name
51	REG	1.18 day	r	/home/jpetz/jboss/jboss-5.1.0.GA/lib/endorsed/jboss-libs-native-jaxws.jar
52	REG	1.18 day	r	/home/jpetz/jboss/jboss-5.1.0.GA/lib/endorsed/jboss-libs-native-saaj.jar
53	REG	1.18 day	r	/home/jpetz/jboss/jboss-5.1.0.GA/lib/endorsed/xalan.jar
54	REG	1.18 day	r	/home/jpetz/jboss/jboss-5.1.0.GA/lib/endorsed/xercesImpl.jar
55	REG	1.18 day	r	/home/jpetz/jboss/jboss-5.1.0.GA/lib/endorsed/stax-api.jar

Figure 4-16 Open Files Window

Table 4-16 describes the Open Sockets window elements

Table 4-16 Open Files Window Elements

Element	Description
Per page	Controls the number of list items per page. Options are: 10, 25, 50, 100, 250, 500, or 1000.
FD	The file description number.
File type	The type of file.
Age	The time since FD was opened.
Open mode	The file mode, where: <ul style="list-style-type: none"> • r = the file is readable • w = the file is writeable • s = the file is synced after writes
Name	The full file name and path.

Buffers Window

The Buffers window () allows users to view the contents of the buffers.

Configuration	Threads	IO	Ticks	Monitors	Memory	Compilers	Settings			
System call profile	Outgoing RPC profile	Incoming RPC profile	FDC profile	FDC stats	Transport profile	Open sockets	Open files	Buffers	Trace	Trace settings

IO - Buffers

- [stdin](#)
- [stdout](#)
- [stderr](#)

Table 4-17 describes the Buffers Window settings window elements.

Table 4-17 Buffers Window Elements

Element	Description
stdin	Display the contents of the stdin buffer.
stdout	Display the contents of the stdin buffer.
stderr	Display the contents of the stdin buffer.

Trace Window

The Trace window lists valid trace profiles. Use the Trace Settings window (Figure 4-17) to enable tracing.

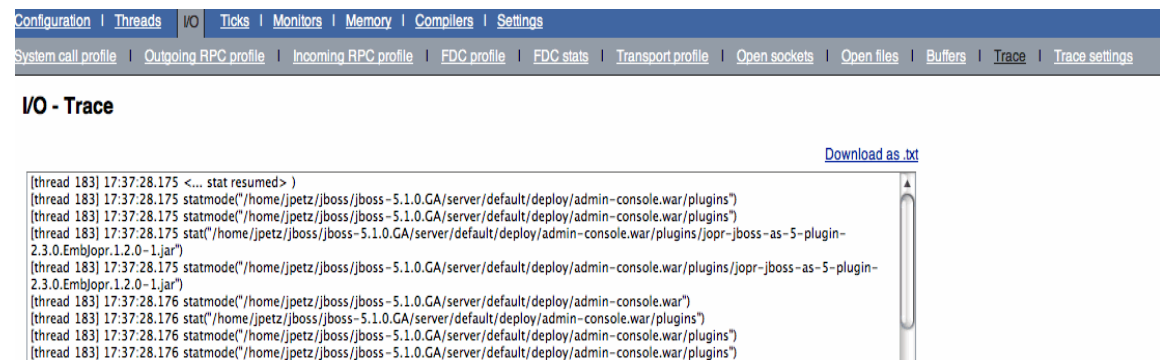


Figure 4-17 Trace Window

Trace Settings Window

Use the Trace settings window to enable tracing and set tracing parameters. Enable trace filtering to focus on a subset of calls. Set the parameters, and then click Configure to save the setting. A text filter can also be applied. The filter is a comma-separated list of the specific function or category names. For example, use “read, write” to capture “read” and “write” calls; use “!accept, !connect” to capture everything except accept and connect calls. Only calls matching the filter are output to the log file.

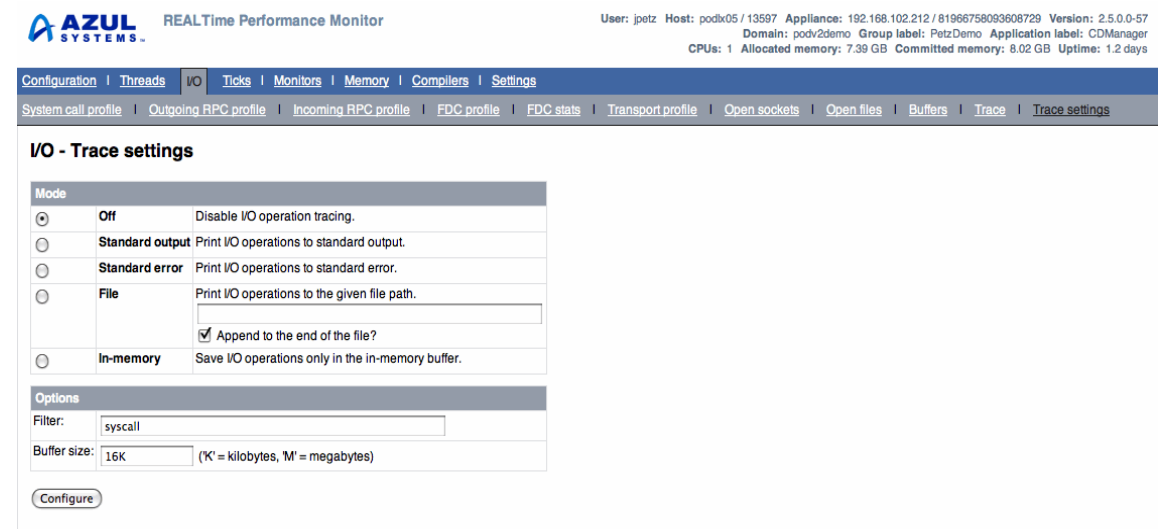


Figure 4-18 Trace Settings Window

Table 4-18 describes the Trace settings window elements.

Table 4-18 *Trace Settings Window Elements*

Element	Description
Mode settings	<ul style="list-style-type: none"> Off Standard output: print output to a standard output. Standard error: print output to standard error Files: print output to specified file. In-memory: print output to an internal buffer. Set the size of memory to occupy (default: 1M).
Options	<ul style="list-style-type: none"> Filter: use the specified filter file to apply filters to traced data. Enter specific I/O call names—as they appear in the I/O profile screens—in the text field. Separate multiple call names with spaces. Use the following labels to include groups of I/O calls: syscall: all system calls (those appearing in the System Call Profile window) RPC: all RPC calls (those appearing in the Incoming and Outgoing RPC Profile windows) Incoming RPC: all incoming RPC calls (those appearing in the Incoming RPC Profile window) Outgoing RPC: all outgoing RPC calls (those appearing in the Outgoing RPC Profile window) FDC: all file data cache calls (those appearing in the FDC Profile window) Transport: all transport calls (those appearing in the Transport Profile window) Thread ID with each line of output add the ID number of the thread making the I/O call to each trace line. Timestamps in GMT with each line of output add the Greenwich Mean Time when the I/O call was made to each trace line.

Ticks Tab

The Ticks tab (Figure 4-19) contains the Profile window that provides information to allow calculation of how processor cores are used for an interval of recorded time. For more description on the tick profiler, refer to [Statistical Tick Profiler](#).

The recorded time interval depends on the number of threads and how busy the threads are. The busier threads are, the greater the number of ticks generated.



NOTE

Method names in the tick profiler are compiled methods and are no longer being interpreted. All time spent by methods being interpreted is summed as one entry named “Interpreter.”

Configuration Threads IO Ticks Monitors Memory Compilers Settings			
Profile			
Tick profile			Reset Tick Profile
Percent	Ticks	Source	
38.1%	99,968	ProxyIO (vm code)	
10.6%	27,895	java.io.UnixFileSystem.getBooleanAttributes0 (native method)	
10.2%	26,824	java.io.UnixFileSystem.getLastModifiedTime0 (native method)	
8.6%	22,656	VM_GCTask (vm code)	
3.0%	7,835	java.io.UnixFileSystem.checkAccess0 (native method)	
2.0%	5,152	object_allocation (vm stub code)	

Figure 4-19 *Tick Profile Window*

Table 4-19 describes the Tick profile window elements.

Table 4-19 Tick Profile Window Elements

Element	Description
Percent	The percentage of CPU resources (% of the total number of ticks collected during the collection period) the function used during the time interval.
Ticks	The number of program counters (“ticks”) collected in the time interval.
Source	<p>A description of the function being counted. The following are definitions for specific functions:</p> <ul style="list-style-type: none"> • VM_GC Task: the garbage collector for the Azul VM. • VM_REALTime_Performance_Monitor: the function that collects and displays REALTime Performance Monitor data. • VM_C2 Compiler: a JIT compiler running that indicates an application is still warming up (compiling methods). • generate_monitorenter: the amount of time spent busy-waiting to acquire monitors.

Data Collection after Application Launch

Check the profile for functions such as VM_C2 Compiler or Interpreter. If these functions are high on the list, then the application is still starting up (a warning message may also appear in the Compiler tab). To determine if the application is warmed up, wait a few seconds, click **Reset Tick Profile**, and then refresh the Web browser until these two threads are not high in the list.

Monitors Tab

The Monitors tab (Figure 4-20) contains the Contention window that provides statistical information for each monitor and sorts monitors by total acquire time. The total acquire time parameter is the time, in ms, for acquires to occur for a given internal or Java-level monitor. Examine the monitors on the list with the highest total acquire times.

Configuration Threads IO Ticks Monitors Memory Compilers Settings						
Contention						
Monitors - Contention						
Name	Acquire time (ms)		Blocking acquires		Waits	
	Total ¹	Max ¹	Count	Count	Max (ms)	Total (ms)
OsrList3_lock	38,123	27	852,682	0	0	0
OsrList1_lock	23,583	28	595,779	0	0	0
PGCTaskManager start monitor	12,052	31	17,850	20,425	3,001,319	1,513,056,014
java.lang.Object	8,780	22	139,800	1,018,183	120,003	1,118,803,193

Figure 4-20 Contention Window

Table 4-20 describes the Contention window elements. Click column titles to sort by relevance.

Table 4-20 Contention Window Elements

Element	Description
Name	The name of the monitor.
Acquire Time (ms)	<ul style="list-style-type: none"> • Total: the time, in ms, for acquires to occur for a given internal or Java-level monitor (not tracked for lightly contended “thin” monitors or those using Optimistic Thread Concurrency). • Max: the maximum time, in ms, to perform an acquire (not recorded when Optimistic Thread Concurrency is enabled).

Table 4-20 *Contention Window Elements (Continued)*

Element	Description
Blocking acquires	<ul style="list-style-type: none"> Count: the number of blocking acquires. Cumulative blockers: the sum of the number of blockers over all blocking attempts. Average blockers: the number of blocking acquires multiplied by the number of threads blocking.
Waits	The number of wait cycles on the thread.
Max wait (ms)	The maximum number, in ms, of allowable wait cycles.
Total wait (ms)	Cumulative wait cycle count, in ms.

Types of Monitors

The following two types of monitors appear in the REALTime Performance Monitor window:

- Java-level monitors: identified by their class name.
- Internal VM monitors: identified with the extension “_lock.” If there are a high number of internal VM monitors, examine the verbose garbage collection (GC) logs. Observe if there are a large number of GC pauses. Contact Azul customer advocacy if this becomes a problem.

Identifying Applications with Locking Problems

Interpret the information in this window in the context of what is expected based on the number of threads in the [Threads Tab](#). This window shows how many threads are contending for locks.

Refresh this window once the application is running, and then refresh it several seconds later. Look for large increases in total acquire times. An application with locking problems shows these increases, which are also visible in the threads list. If there are locking problems in the application, a large number of threads (shown in the [Threads Tab](#)) will be in the state “Acquiring Monitor.”

Calculate Percentage of Time a Typical Thread is Blocked on a Monitor

An important performance measurement is to calculate the percentage of time that a typical thread is blocked on a monitor. Perform the following procedure to calculate thread blocking.

Step 1 Click the **Threads** tab.

Step 2 Count and record the number of threads in the Thread List.

The Thread List can be filtered by thread state, facilitating determination of threads in a certain state.

Step 3 Click the **Monitors** tab.

Step 4 Refresh the screen and measure the time between refreshes by using the “Uptime” indication at the top of the screen by using the timestamp at the bottom of the page.

Step 5 Observe significant increases in the total acquire time for a monitor.

A significant increase is defined as an increase of greater than 1000ms per second. If there were significant increases in the total acquire time, perform the following two calculations:

- ❑ Measure the total acquire times per second:

Increase in Total acquire times in seconds (1000ms per second)/elapsed time in seconds
between measurements = Total acquire times per second.

For example:

4000 (400000 ms) / 10 seconds = 40 seconds of total acquire times per second

- ❑ Measure the percentage of time that a typical thread blocked on this monitor:

Total acquire times per second/number of threads in the thread list = percentage of time that a typical thread blocked on this monitor.

For example:

40 seconds of total acquire times per second / 50 threads = 80%

In this example, every thread was waiting an average of 80% of its time on a monitor, which indicates a locking problem. A percentage close to zero or a single-digit measurement indicates a healthy condition.

Memory Tab

The Memory tab contains the Memory Summary, GC summary, GC history, Allocated Objects and Live Objects profile tabs.

Summary Window

The Memory Summary window (Figure 4-21) provides a summary of total memory usage and grant memory usage.

Configuration

Threads

I/O

Ticks

Monitors

Memory

Compilers

Settings

Summary

GC summary

GC history

Allocated objects

Live objects

Memory - Summary

Java heap usage (as reported to applications)

Metric	Value	Description
Used	16.75 GB	The amount of memory used for live objects and uncollected garbage
Capacity	16.75 GB	The amount of memory (in page increments) in use by the Java heap
Requested capacity	20.13 GB	The amount of committed memory requested at the VM launch (-Xmx)
GC cycle count	86	Number of garbage collection cycles performed since program start

Memory accounts

Name	Allocated	Balance	Committed	Grant
VM internal	1.31 GB	3.93 GB	5.24 GB	0
Java heap	19.40 GB	742.00 MB	20.13 GB	0
Java pause prevention	0	n/a	n/a	n/a

Figure 4-21 Memory Summary Window

Table 4-21 and Table 4-22 describe the Memory Summary window elements.

Table 4-21 Memory Summary Window Elements – Java Heap Usage

Element	Description
Used	The amount of memory currently used by the VM.
Capacity	The amount of memory allocated for use by the VM.
Requested capacity	The maximum amount of memory requested for use by the VM (-Xmx).
GC cycle count	The number of garbage collection cycles.

Table 4-22 Memory Summary Window Elements – Memory Accounts

Element	Description
VM internal	Shows allocated, balance, committed, and grant memory usage for the VM.
Java heap	Shows allocated, balance, committed, and grant memory usage for the Java heap.

Table 4-22 *Memory Summary Window Elements – Memory Accounts*

Element	Description
Java pause prevention	Shows allocated, balance, committed, and grant memory usage for the Java pause prevention pool.

GC Summary Window

Provides a cumulative summary of the recent GC cycle history.

Configuration | Threads | IO | Ticks | Monitors | Memory | Compilers | Settings

Summary | GC summary | GC history | Allocated objects | Live objects

Memory - GC summary (last 50 of 200 cycles over 12.4 hours with 15/9 GC threads)

New generation cycles (25)

Old generation cycles (25)

Category	Statistic	Mean	Stddev	Min	Max	Category	Statistic	Mean	Stddev	Min	Max
Cycle	Interval (sec)	1,728.00	1,199.59	598.70	3,001.26	Cycle	Interval (sec)	1,728.05	1,197.48	599.82	3,002.39
	Pause ratio	0.00%	0.00%	0.00%	0.01%		Pause ratio	0.00%	0.00%	0.00%	0.00%
Committed	Peak used (MB)	3,969.04	2,334.52	1,657	6,400	Committed	Peak used (MB)	3,969.04	2,334.52	1,657	6,400
Pause	Peak used (MB)	3.80	6.99	0	26	Pause	Peak used (MB)	3.80	6.99	0	26
	Unreturned (MB)	0.00	0.00	0	0		Unreturned (MB)	0.00	0.00	0	0
Generations	Total fragmentation (MB)	68.04	6.04	55	80	Generations	Total fragmentation (MB)	55.12	5.83	42	65
	New used (MB)	18.48	8.71	5	44		New used (MB)	60.68	17.75	23	108
	Old used (MB)	243.20	3.30	236	252		Old used (MB)	229.28	2.99	221	237
	Perm used (MB)	102.40	3.81	95	108		Perm used (MB)	101.88	3.59	94	107
	Live (MB)	7.96	7.68	0	30		Live (MB)	281.80	2.26	275	284
	Fragmentation (MB)	9.32	3.70	2	15		Fragmentation (MB)	45.48	3.88	36	52
Garbage	Found (MB)	3,438.92	2,264.89	1,192	5,810	Garbage	Found (MB)	6.12	3.18	0	11
	Collected (MB)	3,438.92	2,264.89	1,192	5,810		Collected (MB)	6.12	3.18	0	11
	Sideband limited (MB)	0.00	0.00	0	0		Sideband limited (MB)	0.00	0.00	0	0
Pages	Promoted to old	13.72	2.20	7	15	Pages	Promoted to old	0.00	0.00	0	0
	Relocated	10.20	3.67	3	15		Relocated	11.32	3.25	5	16
	No relocate	7.48	7.44	0	29		No relocate	13.80	2.26	7	16
	Relocation spike	0.00	0.00	0	0		Relocation spike	3.48	3.06	0	10
	Small	18.48	8.71	5	44		Small	331.16	4.21	321	338
	Large	0.00	0.00	0	0		Large	0.00	0.00	0	0
Pauses	Pause 1 duration (ms)	2.57	0.51	1.82	3.43	Pauses	Pause 1 duration (ms)	2.57	0.51	1.82	3.43
	Pause 2 duration (ms)	10.80	10.92	1.70	55.16		Pause 2 duration (ms)	2.12	1.18	0.85	5.75
	Pause 3 duration (ms)	0.53	0.09	0.39	0.68		Pause 3 duration (ms)	11.10	0.77	9.84	13.32
	Pause 3 count	NaN	NaN	NaN	NaN		Pause 3 count	NaN	NaN	NaN	NaN
	Pause 3 max (ms)	NaN	NaN	NaN	NaN		Pause 3 max (ms)	NaN	NaN	NaN	NaN
Intercycle	Duration (sec)	1,753.46	1,168.72	598.46	3,000.95	Intercycle	Duration (sec)	1,740.85	1,167.91	587.50	2,991.13
	Allocation rate (MB/s)	2.08	0.05	2.01	2.17		Allocation rate (MB/s)	0.00	0.00	0.00	0.00
	Perm allocation rate (MB/s)	0.00	0.00	0.00	0.00		Perm allocation rate (MB/s)	0.00	0.00	0.00	0.00
Intracycle	Duration (sec)	0.29	0.05	0.21	0.41	Intracycle	Duration (sec)	12.61	2.94	5.02	18.86
	Allocation rate (MB/s)	27.60	29.47	0.00	112.31		Allocation rate (MB/s)	1.26	0.51	0.42	3.19
	Perm allocation rate (MB/s)	3.38	0.59	2.26	4.48		Perm allocation rate (MB/s)	0.06	0.07	0.00	0.28
App threads	Total threads	94.16	0.37	94	95	App threads	Total threads	94.12	0.32	94	95
	Threads delayed	0.00	0.00	0	0		Threads delayed	0.00	0.00	0	0
	Average thread delay (sec)	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		Max thread delay (sec)	0.00	0.00	0.00	0.00

Figure 4-22 *GC Summary Window*

Figure 4-23 shows the GC Summary window when Generational Pauseless GC is active.

Configuration Threads I/O Ticks Monitors Memory Settings					
Summary GC summary GC history Object profile					
Memory - GC summary (first 24 cycles over 8.8 minutes with 15/4 GC threads)					
New generation cycles (12)			Old generation cycles (12)		
Category	Statistic	Mean	Stddev	Min	Max
Cycle	Interval (sec)	41.21	23.59	3.13	60.20
	Pause ratio	0.22%	0.38%	0.06%	1.47%
Committed	Peak used (MB)	272.58	123.94	153	511
Pause	Peak used (MB)	0.00	0.00	0	0
	Unreturned (MB)	0.00	0.00	0	0
Generations	Total fragmentation (MB)	4.75	1.01	3	7
	New used (MB)	31.25	31.78	1	80
	Old used (MB)	99.25	42.26	0	157
	Perm used (MB)	31.67	9.61	11	40
	Live (MB)	34.75	37.74	0	91
	Fragmentation (MB)	0.50	0.96	0	3
Garbage	Found (MB)	97.67	101.55	10	287
	Collected (MB)	97.67	101.55	10	287
	Sideband limited (MB)	0.00	0.00	0	0
Pages	Promoted to old	2.75	7.41	0.00	27.00
	Relocated	0.00	0.00	0.00	0.00
	No relocate	8.08	10.36	0.00	27.00
	Relocation spike	2.25	5.13	0.00	16.00
	Small	28.33	28.27	1.00	80.00
Pauses	Large	2.92	7.45	0.00	27.00
	Pause 1 duration (ms)	26.34	20.93	10.29	90.76
	Pause 2 duration (ms)	16.89	5.15	11.75	28.76
	Pause 3 duration (ms)	3.32	0.98	1.85	5.01
	Pause 3 count	1.00	0.00	1.00	1.00
Intercycle	Pause 3 max (ms)	3.32	0.98	1.85	5.01
	Duration (sec)	42.18	21.77	2.85	59.86
	Allocation rate (MB/s)	4.65	4.87	0.15	14.06
Intracycle	Perm allocation rate (MB/s)	0.14	0.20	0.00	0.70
	Duration (sec)	0.26	0.12	0.14	0.50
	Allocation rate (MB/s)	24.12	41.32	-52.97	105.09
App threads	Perm allocation rate (MB/s)	3.60	2.60	0.00	7.08
	Total threads	64.75	6.06	51	69
	Threads delayed	0.00	0.00	0	0
	Average thread delay (sec)	0.00	0.00	0.00	0.00
	Max thread delay (sec)	0.00	0.00	0.00	0.00
Category	Statistic	Mean	Stddev	Min	Max
Cycle	Interval (sec)	41.30	23.55	3.50	60.67
	Pause ratio	0.33%	0.53%	0.14%	2.09%
Committed	Peak used (MB)	272.58	123.94	153	511
Pause	Peak used (MB)	0.00	0.00	0	0
	Unreturned (MB)	0.00	0.00	0	0
Generations	Total fragmentation (MB)	4.83	1.34	3	8
	New used (MB)	33.00	32.92	1	87
	Old used (MB)	90.17	37.42	0	150
	Perm used (MB)	31.17	8.94	12	39
	Live (MB)	107.67	46.22	9	177
	Fragmentation (MB)	4.17	1.28	1	6
Garbage	Found (MB)	11.75	13.54	0	49
	Collected (MB)	11.75	13.54	0	49
	Sideband limited (MB)	0.00	0.00	0	0
Pages	Promoted to old	0.00	0.00	0.00	0.00
	Relocated	0.00	0.00	0.00	0.00
	No relocate	15.42	22.91	0.00	76.00
	Relocation spike	29.00	22.58	10.00	78.00
	Small	92.25	38.43	13.00	156.00
Pauses	Large	29.42	9.07	0.00	33.00
	Pause 1 duration (ms)	26.34	20.93	10.29	90.77
	Pause 2 duration (ms)	55.53	17.97	17.68	78.67
	Pause 3 duration (ms)	6.39	3.12	2.17	14.00
	Pause 3 count	1.83	0.37	1.00	2.00
Intercycle	Pause 3 max (ms)	3.88	1.95	2.17	9.42
	Duration (sec)	41.39	21.28	2.43	58.93
	Allocation rate (MB/s)	0.00	0.00	0.00	0.00
Intracycle	Perm allocation rate (MB/s)	0.13	0.19	0.00	0.70
	Duration (sec)	1.14	0.50	0.32	1.90
	Allocation rate (MB/s)	23.91	33.26	-3.17	97.51
App threads	Perm allocation rate (MB/s)	1.12	1.80	0.00	6.36
	Total threads	64.75	6.06	51	69
	Threads delayed	0.00	0.00	0	0
	Average thread delay (sec)	0.00	0.00	0.00	0.00
	Max thread delay (sec)	0.00	0.00	0.00	0.00

Figure 4-23 GC Summary Window for Generational Pauseless GC

Table 4-23 lists the GC Summary window elements. Differing GPGC elements are noted.

Table 4-23 GC Summary Window Elements

Element	Description
Cycle	Interval (sec): elapsed time between successive GC cycle starts. Pause ratio: ratio of time application spent in safepoint pauses during the cycle to cycle interval time
Type (GPGC only)	Indicates whether it is a new or old GC cycle, as follows: <ul style="list-style-type: none"> • Old: old generation cycle • NTO: new generation cycle • Sys: PGC cycle
Mode	<ul style="list-style-type: none"> • p: indicates if allocation from pause pool is enabled. • g: indicates if allocation from grant pool is enabled.

Table 4-23 GC Summary Window Elements (Continued)

Element	Description
Heap	<ul style="list-style-type: none"> • Live (MB): the amount of Java memory used by all application objects currently not known to be dead. • Peak (MB): the peak size of the Java heap recorded during the GC cycle. • Peak (pgs): the peak size of the Java heap recorded during the GC cycle (in MB by page increments). • Max (MB): maximum amount of heap memory guaranteed to the application. This is generally set by <code>-Xmx</code> option.
Pause	These values are used for internal Azul diagnostic purposes only.
Generations (GPGC only)	<ul style="list-style-type: none"> • Total fragmentation (MB): the total fragmentation in the heap. • New used (MB): the amount of memory used by all young objects not known to be dead during the GC cycle. • Old used (MB): the amount of memory used by all old objects not known to be dead during the GC cycle. • Perm used (MB): the amount of memory used by all permanent objects not known to be dead during the GC cycle. • Live (MB): the live memory in the generation(s) being collected. • Fragmentation (MB): the fragmentation in the generation(s) being collected.
Garbage ¹	<ul style="list-style-type: none"> • Found (MB): the garbage found in the generation(s) being collected. • Freed (MB): the garbage freed in the generation(s) being collected. • Sideband limited (MB): used for internal Azul diagnostics purposes only.
Pages (GPGC only)	These values are used for internal Azul diagnostic purposes only.
Pauses ²	<ul style="list-style-type: none"> • Pause 1 start (sec): the start time of the pause 1 safepoint. • Pause 1 duration (ms): the duration time of the pause 1 safepoint. • Pause 2 start (sec): the start time of the pause 2 safepoint. • Pause 2 duration (ms): the duration of the pause 2 safepoint • Pause 3 duration (ms): the total duration of all pause 3 safepoints for this GC cycle.
Pauses (GPGC only)	<ul style="list-style-type: none"> • Pause 3 start (sec): the start time of the pause 3 safepoint. • Pause 3 count: the number of pause 3 safepoints. • Pause 3 max: the maximum duration of any one pause 3 safepoint.
Intercycle	<ul style="list-style-type: none"> • Duration (sec.): the elapsed time from the end of the last GC cycle to the start of this GC cycle. • Allocation rate (MB/s): the object allocation rate in the generation being collected in the period between the last GC cycle to the start of this GC cycle.
Intercycle (GPGC only)	<ul style="list-style-type: none"> • Perm allocation rate (MB/s): the object allocation rate in the permanent generation during this GC cycle.
Intracycle	<ul style="list-style-type: none"> • Duration (sec.): the GC cycle elapsed time; the duration of time from the beginning to the end of this GC cycle. • Allocation rate (MB/s): the object allocation rate in the generation being collected during this GC cycle.
Intracycle (GPGC only)	<ul style="list-style-type: none"> • Perm allocation rate (MB/s): the object allocation rate in the permanent generation during this GC cycle.

Table 4-23 GC Summary Window Elements (Continued)

Element	Description
App threads	<ul style="list-style-type: none"> Total threads: the total count of Java threads in the VM. Threads delayed: the number of threads blocked trying to allocate memory during the GC cycle. When the number of delayed threads is greater than 0, application performance is affected due to lack of available memory. Average thread delay (sec): the average delay for threads blocked trying to allocate during the GC cycle. Max thread delay (sec): the maximum delay for threads blocked trying to allocate.

Note 1. Depending on application behavior and memory usage, the garbage collector may collect fewer objects from memory than total dead objects found. This optimizes the GC process and lowers the impact on the application (if any).

Note 2. The GC cycle has several phases for garbage collection. When shifting between phases, all threads reach a safepoint, incurring a short pause.

GC History Window

Provides a detailed summary of the recent GC cycle history with a row per cycle. Click column titles to sort by relevance.

Configuration

Threads

IO

Tools

Monitors

Memory

Compilers

Settings

Summary

GC summary

GC history

Allocated objects

Live objects

Memory - GC history (last 50 of 204 cycles over 12.0 hours with 15/9 GC threads)

Download as txt

Cycle		Memory (MB)										Pauses										Intercycle										App threads										
		Heap		Committed		Pause		Generations		Garbage		Pages		1		2		3		4		Durs		Max		Durs		Max		Durs		Max		Durs		Max						
Type	#	Modes	Max	Peak	Used	Peak	Un ret	All frag	New	Old	Perm	Live	Frag	Found	Free	Side-band limit	Move to old	Reloc	No reloc	Reloc spike	Sm	Lg	Start (sec)	Durs (ms)	Start (sec)	Durs (ms)	Start (sec)	Durs (ms)	Count	Start (sec)	Durs (ms)	Max (ms)	Durs (sec)	Alloc (MB/s)	Perm alloc (MB/s)	Durs (sec)	Alloc (MB/s)	Perm alloc (MB/s)	#	Avg (sec)	Max (sec)	
NTO	155	-p000	6,400	1,680	389	0	0	58	26	240	97	14	7	1,221	1,221	0	8	8	14	0	26	0	133,280.01	2.37	133,280.15	11.99	133,280.22	0.51	1	133,280.27	6.66	6.66	611.03	2.11	0.00	0.28	64.54	3.35	94	0	0.00	0.00
Old	155	-p000	6,400	1,680	419	0	0	44	72	224	97	276	36	4	4	0	0	11	8	2	321	0	133,280.01	2.37	133,281.53	3.74	133,281.58	11.77	2	133,298.77	2.16	1.18	597.94	0.00	0.00	18.86	0.42	0.05	94	0	0.00	0.00
NTO	157	-p000	6,400	6,396	381	0	0	64	18	239	98	4	14	5,804	5,804	0	15	3	0	18	0	0	136,262.82	3.13	136,262.94	2.03	136,262.99	0.42	1	136,263.05	5.23	5.23	2,982.53	2.02	0.00	0.27	11.12	3.46	94	0	0.00	0.00
Old	158	-p000	6,400	6,396	406	0	0	56	52	230	98	283	42	3	3	0	0	13	15	9	328	0	136,262.82	3.13	136,264.28	0.85	136,264.34	11.08	2	136,274.15	2.43	1.40	2,963.94	0.00	0.00	11.43	1.31	0.08	94	0	0.00	0.00
NTO	159	-p000	6,400	1,720	386	0	0	61	20	241	99	11	7	1,253	1,253	0	11	8	11	0	20	0	136,880.02	2.38	136,880.22	12.14	136,880.28	0.42	1	136,880.33	6.51	6.51	616.93	2.17	0.00	0.34	35.04	2.77	94	0	0.00	0.00
Old	159	-p000	6,400	1,720	411	0	0	48	59	227	98	279	40	11	11	0	0	10	11	0	325	0	136,880.02	2.38	136,881.56	11.57	2	136,893.45	2.28	1.28	605.77	0.00	0.00	13.53	0.96	0.00	94	0	0.00	0.00		
NTO	161	-p010	6,400	6,400	411	26	0	68	44	242	99	30	14	5,801	5,801	0	15	29	0	44	0	0	139,862.17	1.84	139,862.26	1.06	139,862.32	0.42	1	139,862.39	2.35	2.35	2,981.81	2.02	0.00	0.26	112.31	3.61	94	0	0.00	0.00
Old	161	-p010	6,400	6,400	418	26	0	59	61	231	100	293	45	8	8	0	0	14	15	6	331	0	139,862.17	1.84	139,863.54	1.93	139,863.59	10.29	2	139,866.95	2.18	1.11	2,968.61	0.00	0.00	6.87	2.04	0.28	94	0	0.00	0.00
NTO	163	-p000	6,400	1,700	383	0	0	68	12	244	101	1	10	1,245	1,245	0	13	10	1	0	12	0	140,480.07	2.69	140,480.19	8.54	140,480.25	0.44	1	140,480.30	6.56	6.56	617.65	2.10	0.00	0.25	3.98	73.8	94	0	0.00	0.00
Old	163	-p000	6,400	1,700	405	0	0	52	52	229	98	281	42	2	2	0	0	7	13	1	327	0	140,480.07	2.69	140,481.50	2.81	140,481.55	11.30	2	140,496.26	2.54	1.52	611.03	0.00	0.00	16.33	0.98	0.00	94	0	0.00	0.00

Memory - GC history (last 50 of 686 cycles over 24.4 minutes with 15/4 GC threads)

			Memory (MB)										Pauses																														
Cycle			Heap	Committed	Pause	Generations					Garbage					Pages					1					2					3					4					Intercycle		Intra
Type	#	Modes	Max	Peak	Used	Peak	Un ret	All frag	New	Old	Perm	Live	Frag	Found	Freed	Side-band limit	Mov to old	Reloc	No Reloc	Reloc spike	Sm	Lg	Start (sec)	Dur (ms)	Start (sec)	Dur (ms)	Start (sec)	Dur (ms)	Start (sec)	Dur (ms)	Start (sec)	Dur (ms)	Alloc (MB/s)	Perm alloc (MB/s)	Dur (sec)								
NTD	637	-p000	1,152	172	162	0	0	2	1	104	36	0	0	10	10	0	0	1	0	0	1	0	18,930.66	19.24	18,930.73	16.56	18,930.80	1.55	-	-	59.85	0.17	0.00	0.1									
Old	638	-p000	1,152	172	163	0	0	2	2	104	36	129	2	3	3	0	0	16	0	16	107	33	18,930.66	19.24	18,931.59	60.67	18,931.75	1.54	18,931.81	1.55	58.78	0.00	0.00	1.2									
NTD	639	-p000	1,152	174	162	0	0	2	1	104	36	0	0	12	12	0	0	1	0	0	1	0	18,990.66	19.22	18,990.74	13.99	18,990.80	1.43	-	-	59.85	0.20	0.00	0.1									
Old	640	-p000	1,152	174	165	0	0	2	4	104	36	129	2	3	3	0	0	16	0	16	107	33	18,990.66	19.23	18,991.59	59.17	18,991.76	1.51	18,991.81	1.08	58.79	0.00	0.00	1.2									
NTD	641	-p000	1,152	171	162	0	0	2	1	104	36	0	0	9	9	0	0	1	0	0	1	0	19,050.67	19.01	19,050.73	14.30	19,050.80	1.59	-	-	59.85	0.15	0.00	0.1									
Old	642	-p000	1,152	171	163	0	0	2	2	104	36	129	2	3	3	0	0	16	0	16	107	33	19,050.67	19.01	19,051.60	59.24	19,051.77	1.80	19,051.82	1.29	58.79	0.00	0.00	1.2									
NTD	643	-p000	1,152	174	162	0	0	2	1	104	36	0	0	12	12	0	0	1	0	0	1	0	19,110.67	19.10	19,110.74	14.28	19,110.80	1.57	-	-	59.85	0.20	0.00	0.1									
Old	644	-p000	1,152	174	163	0	0	2	2	104	36	129	2	3	3	0	0	16	0	16	107	33	19,110.67	19.10	19,111.61	61.05	19,111.78	1.79	19,111.83	1.31	58.78	0.00	0.00	1.2									
NTD	645	-p000	1,152	173	162	0	0	2	1	104	36	0	0	11	11	0	0	1	0	0	1	0	19,170.67	19.58	19,170.74	14.22	19,170.80	1.64	-	-	59.85	0.18	0.00	0.1									
Old	646	-p000	1,152	173	163	0	0	2	2	104	36	129	2	3	3	0	0	16	0	16	107	33	19,170.67	19.58	19,171.60	60.82	19,171.77	1.51	19,171.82	1.30	58.77	0.00	0.00	1.2									

Figure 4-25 GC History Window for Generational Pauseless GC

Allocated Objects Window

The Allocated Objects window (Figure 4-26) provides information on instances of allocated objects and how much memory they occupy. An object can be selected in the Allocated Objects window to display the objects which own them to track memory possible memory leaks. This information is useful when analyzing garbage collection problems related to a large number of allocated objects, or to identify applications with memory leaks.

The Allocated Objects window is enabled by default. When using PGC (Figure 4-26), the Allocated Objects window presents statistics for objects allocated by threads in the most recent GC cycle interval (that is, between the two most recent successive GCcycle start points).

Configuration Threads IO Ticks Monitors Memory Compilers Settings									
Summary GC summary GC history Allocated objects Live objects									
Memory - Allocated objects									
									Per page: 10
<<< 1 to 10 of 240 >>> Download as .csv									
Class name	Size (B)	Count	Avg (B)						
Total	1,209,228,704	18,612,177	65.0						
char[]	610,355,984	6,089,569	100.2						
int[]	127,861,416	2,311,841	55.3						
byte[]	103,478,936	833,400	124.2						
java.util.regex.Matcher	88,387,360	1,104,842	80.0						
java.lang.String	86,103,744	3,587,656	24.0						
java.nio.HeapByteBuffer	41,281,104	860,023	48.0						
java.nio.HeapCharBuffer	41,281,104	860,023	48.0						
java.lang.Object[]	30,923,800	320,822	96.4						
java.lang.StringBuilder	23,315,064	971,461	24.0						
org.jboss.virtual.VirtualFile	12,080,544	377,517	32.0						
<<< 1 to 10 of 240 >>>									

Figure 4-26 Allocated Objects Window

When GPGC is active, the Allocated Objects window presents statistics for objects allocated by threads since the beginning of the last GC cycle. Information updates once per GC cycle..

Table 4-24 describes the Allocated Objects window elements. Click column titles to sort by relevance.

Table 4-24 Allocated Objects Window Elements

Element	Description
Size (B)	The total memory size, in bytes, for objects allocated by the thread since the beginning of the last new generation GC cycle.
Count	The number of allocated objects of a particular class.
Avg (B)	The average size, in bytes, for each allocated object in the class. For regular classes, the average size is the same as the actual size of each object. For arrays, the average is a real average of different sized objects.
Class name	The name of the class. Regular classes do not have brackets ([]) in the name. Arrays have brackets following the name. For example, java.lang.Object[] is an array of java.lang.Object.

Live Objects Window

The Live Objects window (Figure 4-27) provides information on instances of classes and how much memory they occupy. An object can be selected in the Allocated Objects window to display the objects which own them to track memory possible memory leaks. This information is useful when analyzing garbage collection problems related to a large number of live objects, or to identify applications with memory leaks.

The Allocated Objects window is enabled by default. When using PGC, the Live Objects window presents statistics for live objects found in the last GC cycle. Information updates once per GC cycle.

Class name	Size (B)	Count	Avg (B)
Total	283,073,216	4,277,900	66.2
char[]	46,800,848	544,623	85.9
Built-in VM methodKlass	17,017,672	111,893	152.1
java.util.TreeMap\$Entry	16,889,152	301,592	56.0
Built-in VM constMethodKlass	15,598,976	111,893	139.4
java.util.HashMap\$Entry[]	14,274,920	84,947	168.0
java.lang.String	13,025,760	542,740	24.0
Built-in VM constantPoolKlass	12,780,240	11,463	1,114.9
Built-in VM instanceKlass	10,324,032	11,463	900.6
java.util.HashMap\$Entry	10,250,720	256,268	40.0
java.lang.reflect.Method	7,798,464	54,156	144.0

Figure 4-27 Live Objects Window

When GPGC is active, the Live Objects window presents statistics for objects allocated by threads since the beginning of the last GC cycle. Information updates once per GC cycle..

Table 4-25 describes the Object profile window elements. Click column titles to sort by relevance.

Table 4-25 Live Objects Window Elements

Element	Description
Size (B)	The total memory size, in bytes, for objects allocated by the thread since the beginning of the last new generation GC cycle.
Count	The number of allocated objects of a particular class.
Avg (B)	The average size, in bytes, for each allocated object in the class. For regular classes, the average size is the same as the actual size of each object. For arrays, the average is a real average of different sized objects.
Class name	The name of the class. Regular classes do not have brackets ([]) in the name. Arrays have brackets following the name. For example, java.lang.Object[] is an array of java.lang.Object.

Compiler Tab

The Compiler tab of the REALTime Performance Monitor ([Figure 4-28](#)) displays the status of Server and Client compiler tasks.

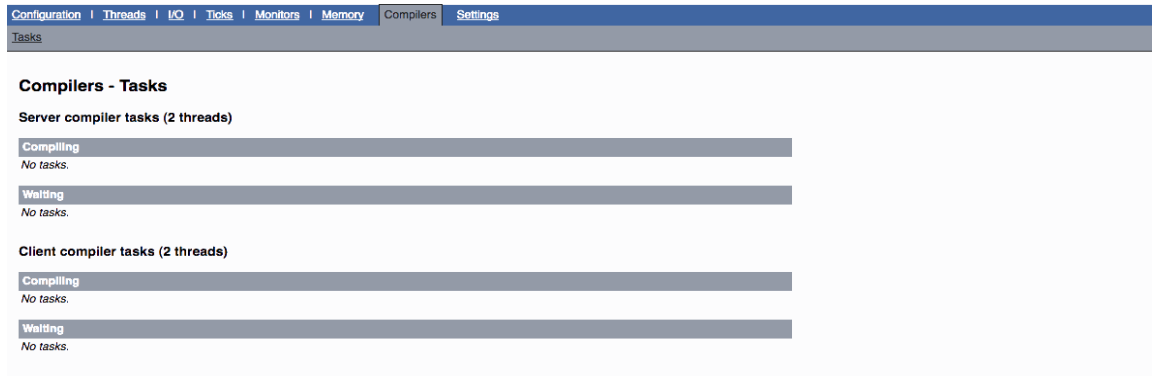


Figure 4-28 *Compiler Tab*

[Table 4-26](#) describes the Compiler Tab window elements. .

Table 4-26 *Compiler Tab Window Elements*

Element	Description
Compiling	Status of threads being compiled.
Waiting	Status of threads waiting to be compiled

Settings Tab

The Settings tab ([Figure 4-29](#)) provides controls for enabling GC logging, choosing the level of detail logged, and whether the output is sent to standard out or saved to a new or existing file.

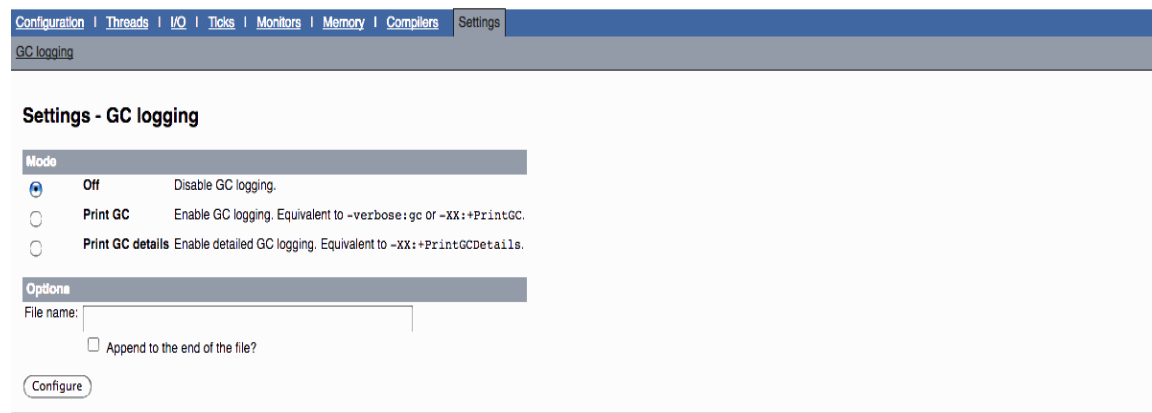


Figure 4-29 *Settings Window*

Table 4-27 describes the Settings window elements.

Table 4-27 *Settings Window Elements*

Element	Description
Off	Disables GC logging for this VM instance. This is the default setting.
Print GC	Enables GC logging with summarized output. Output is equivalent to using the command line option -XX:+PrintGC.
Print GC details	Enables GC logging with detailed output. Output is equivalent to using the command line option -XX:+PrintGCDetails.
Options: File name	Type a file name to send GC log output instead of sending it to standard out. Select the check box and type an existing file name to append the output to the end of an existing file.

Azul VM Configuration for IBM WebSphere

This appendix describes how to configure the Azul VM software for use with IBM WebSphere (version 5.1, 6.0, and 6.1) on Solaris, Linux, and AIX. Configuration does not require modification to the JVM provided with IBM WebSphere, but creates a WebSphere-compatible JDK from an existing Azul JDK. The script used can be started manually or automatically by using the **-was** option during Azul VM installation.

Create a WebSphere-compatible JDK from an Existing Azul JDK

To ensure that the Azul VM software operates correctly in the IBM WebSphere environment, a script is provided to create symbolic links from the VM to a number of files in the IBM JDK.



NOTE Note that when using the IBM JDK version 1.4.1, WebSphere 5.1 application server security does not always work with the Azul VM. It is recommended that the IBM JDK version 1.4.2 be used (WAS 5.1.1 service pack 1).

Use the following procedure to create an IBM WebSphere-compatible JDK from an existing Azul JDK. This procedure applies to versions 5.1, 6.0, and 6.1 of IBM WebSphere.

Step 1 Ensure that the following items are available:

- ❑ The WebSphere JDK from a Solaris WebSphere installation. This is a Sun HotSpot JDK with a few files modified by IBM for their CORBA and J2SE implementations. These JDK files must have open read permissions.



NOTE The IBM JDK for Solaris is included with the Solaris distribution of WebSphere. The Azul VM software does not include the IBM JDK for Solaris.

- ❑ The stock Azul JDK. Write permissions must be open on the `jre/lib`, `jre/lib/endorsed`, `jre/lib/ext`, and `jre/lib/security` directories of the Azul JDK.
- ❑ For installations on the IBM AIX platform, a local AIX JDK is required.
- ❑ The `create_was_jdk.sh` script, and a bash interpreter. The standard Bourne shell will not work. The script file is stored in the home directory of the Azul JDK.
- ❑ When running the WebSphere application server with the Azul JDK, read permissions must be open on the files in the WebSphere JDK directory, as well as the files in the Azul JDK directory.

Step 2 Make a copy of the Azul JDK and perform all modifications to the copy.

Retain the original Azul JDK for future use.

Step 3 Run the WebSphere compatibility script in interactive mode.

The script prompts for necessary information including the WebSphere version, location of the WebSphere JDK from where the files are symbolically linked, and the location of the Azul JDK to modify. Default option values display in parenthesis. The script automatically uses the default value for an option if no value is entered. For example,

```
$ ./create_was_jdk.sh
WebSphere version [5|6] (6):
Location of IBM Solaris JDK (/opt/WebSphere/AppServer/java):
Location of Azul JDK (): /azul-j2sdk1.4.2-2.3.0.0-124
```

NOTE



In the Azul JDK path, the phrase “2.3.0.0-124” indicates the Azul software release version.

WebSphere 5.1 installations

Step 4 Navigate to the Azul JDK directory and verify that the following files are symbolically linked using the `ls -l` command.

- `ibm_bin/`
- `ibm_lib/`
- `jre/lib/endorsed/xml.jar`
- `jre/lib/ext/activation.jar`
- `jre/lib/ext/dnsns.jar`
- `jre/lib/ext/gskikm.jar`
- `jre/lib/ext/ibmcertpathfw.jar`
- `jre/lib/ext/ibmcertpathprovider.jar`
- `jre/lib/ext/ibmext.jar`
- `jre/lib/ext/ibmjcefijs.jar`
- `jre/lib/ext/ibmjceprovider.jar`
- `jre/lib/ext/ibmjgssprovider.jar`
- `jre/lib/ext/ibmjssefijs.jar`
- `jre/lib/ext/ibmjsse.jar`
- `jre/lib/ext/ibmorib.jar`
- `jre/lib/ext/ibmpkcs11.jar`
- `jre/lib/ext/ibmpkcs.jar`
- `jre/lib/ext/ibmspnego.jar`
- `jre/lib/ext/ibmtools.jar`
- `jre/lib/ext/iwsorbutil.jar`
- `jre/lib/ext/log.jar`
- `jre/lib/ext/mail.jar`
- `jre/lib/ext/oldcertpath.jar`
- `jre/lib/ext/PD.jar`
- `jre/lib/orb.properties`

Step 5 Verify that a backup copy of the `jre/lib/security/java.security` file exists in the Azul JDK installation as `jre/lib/security/java.security.orig`.

WebSphere 6.0 and 6.1 installations

Navigate to the Azul JDK directory and verify that the following files are symbolically linked using the `ls -l` command.

- `ibm_bin`
- `ibm_lib`
- `jre/lib/ext/dnsns.jar`
- `jre/lib/ext/gskikm.jar`
- `jre/lib/ext/ibmjcefijs.jar`
- `jre/lib/ext/ibmjceprovider.jar`
- `jre/lib/ext/ibmjssefijs.jar`
- `jre/lib/ext/ibmjsseprovider2.jar`
- `jre/lib/ext/ibmpkcs11impl.jar`
- `jre/lib/ext/ibmpkcs11.jar`
- `jre/lib/ext/ibmspnego.jar`
- `jre/lib/ext/ibmtools.jar`
- `jre/lib/ext/iwsorbutil.jar`
- `jre/lib/ext/oldcertpath.jar`
- `jre/lib/endorsed/ibmcertpathfw.jar`
- `jre/lib/endorsed/ibmcertpathprovider.jar`
- `jre/lib/endorsed/ibmext.jar`
- `jre/lib/endorsed/ibmjgssfw.jar`
- `jre/lib/endorsed/ibmjgssprovider.jar`
- `jre/lib/endorsed/ibmoribapi.jar`
- `jre/lib/endorsed/ibmorib.jar`
- `jre/lib/endorsed/ibmpkcs.jar`
- `jre/lib/endorsed/xml.jar`
- `jre/lib/orb.properties`
- `jre/lib/security/java.security`

Verify that the `jre/lib/endorsed/azul.jar` file exists.

This jar file contains the `sun/misc/Version.class` class file.

Step 6 Ensure that the following files are backed up (the script backs up these files with a “.orig” extension) and are symbolically linked:

- jre/lib/jsse.jar
- jre/lib/security/cacerts
- jre/lib/security/java.security
- jre/lib/security/local_policy.jar
- jre/lib/security/US_export_policy.jar

Step 7 Ensure that the following file is backed up:

- jre/lib/security/java.policy

This file is modified by the script.

Step 8 Change directories to <WasHome>/profiles/<AppServer Profile>/configuration.

Step 9 Edit the config.ini file by commenting out the following line:

```
osgi.framework.extensions=com.ibm.cds
```

Step 10 Change directories to <WasHome>/plugins.

Step 11 Rename the com.ibm.cds_1.0.0.jar file; for example, com.ibm.cds_1.0.0.jar.tmp.

Ensure that .jar is no longer the extension of the file.

Step 12 Start the WAS Server instance:

- a. Change directories to <WasHome>/profiles/<AppServer Profile>/bin
- b. Execute the following script command

```
./tartServer.sh <server-name> -script
```

This generates the start_<server-name>.sh script, where, <server_name> is the name of the server. For example, if the server name is server1, then use start_server1.sh.

Step 13 Edit the last line so that it runs using Azul VM and uses Azul options.

Notes

Azul VM Configuration for Resin

This appendix explains how to configure the Azul VM for use with Resin. The overlay script used can be started manually or automatically by using the **-resin** option during Azul VM installation.

Overview

Installations of the Resin application server 3.0.9 or higher from Caucho Technology may already be configured to use an optional native code library to improve performance for socket and file I/O in host-based environments. When using this optional native code library, also use a special Resin Professional overlay script to ensure optimal performance with the Azul VM.

A script is provided with the Azul VM software to replace performance-sensitive parts of the Resin native code library with Java implementations optimized for use with Azul Compute Appliances. If the optional native code library was not created, it is created during installation.

The script file is located at: `$JAVA_HOME/bin/overlay`. The script performs the following actions:

- Builds the Resin Professional JNI libraries, `$RESIN_HOME/libexec/libresin.so` (if they do not exist).
- Creates backup copies of the following files:
 - `$RESIN_HOME/lib/resin.jar`
 - `$RESIN_HOME/lib/pro.jar` (version 3.0.13 and higher)
 - `$RESIN_HOME/lib/resin-util.jar` (version 3.1.0 and higher)
- Replaces selected file contents with Java methods and related utility classes.

Script File Syntax

Use the following script to interactively install or uninstall modifications to a number of application installations. This script allows applications to optimally perform with the Azul VM.

```
overlay [-v|-h] install|uninstall resin
```

Backups of all modified files are created.

Table B-1 describes the script file syntax.

Table B-1 *Script File Syntax*

Modes of operation	
install	Perform each modification and create backups.
uninstall	Undo each modification from existing backups.
Supported application	
Keyword	resin
Application	Resin Professional
Versions	3.0.x and 3.1.x
Options	
-v	Print the arguments and output of each command executed.
-h	Print this message and exit.

Running the Script

Use the following procedure to run the Resin Professional overlay script.

Step 1 Enter the following command:

```
overlay [-v|-h] install|uninstall resin
```

For example:

```
user@host: /azul-j2sdk1.4.2-2.2.2.0-1-linux/bin/overlay
install resin
Beginning Resin Professional overlay
Please enter your JAVA_HOME: /usr/local/j2sdk1.4.2_06
Please enter the Resin Professional 3.0.x installation directory:
/.../resin/resin-pro-3.0.13
Warning: /.../resin/resin-pro-3.0.13/libexec/libresin.so
doesn't exist which implies that the Resin Professional JNI libraries
have not been built. You must build them to gain the performance
benefits of their Azul replacements.
Would you like for this script to build them now (y/n)? [y] :
> cd /.../resin/resin-pro-3.0.13
> JAVA_HOME=/usr/local/j2sdk1.4.2_06 ./configure
checking build system type... i686-pc-linux-gnu
checking host system type... i686-pc-linux-gnu
checking target system type... i686-pc-linux-gnu
> make
(cd modules/c/src; make)
```

```
make[1]: Entering directory `/home/mcneil/tmp/resin/resin-pro-3.0.13/
modules/c/src'

for dir in common resin resinssl; do (cd $dir; make); done

> make install
(cd modules/c/src; make install)

make[1]: Entering directory `../../resin/resin-pro-3.0.13/modules/c/src'
for dir in common resin resinssl; do (cd $dir; make install); done

Backing up resin.jar to resin.jar.bak
Backing up pro.jar to pro.jar.bak
Extracting shared classes from overlay.jar
Creating classes with Java code replacing native code
Updating resin.jar
Updating pro.jar

Done overlaying Resin Professional

Cleaning up
```

Step 2 Enter the JAVA_HOME directory at the prompt.

For example:

```
/usr/local/j2sdk1.4.2_06
```

Step 3 Enter the Resin Professional 3.0.x installation directory at the prompt.

For example:

```
../../resin/resin-pro-3.0.13
```

Step 4 Type Y when prompted to build the Resin Professional JNI libraries.

At this point the script makes the necessary modifications and displays status messages.

Notes

Index

A

- AIX 6
- application hosts
 - NTP server configuration and 7
- application servers 1
 - Resin 69
- application throughput 4
- Azul JDK 65
- Azul VM proxy 12
- azul.java.vminitargs.post 22
- azul.java.vminitargs.pre 22

B

- base memory 22
- browser support 28

C

- command line options
 - Azul VM GC substitutions 19, 70
 - for Azul compute pools 20
 - PX:+Help 16
 - PX:+MulticastDiscovery 12, 13, 17
 - PX:+TraceLibraryMapping 24
 - PX:+UnicastTCPDiscovery 13, 17
 - PX:+UnicastUDPDiscovery 13, 17
 - PX:+UseDirectPath 18
 - PX:AppLabel 16
 - PX:BackendCoredumpLimit 17
 - PX:CPMDomain 11, 16, 21
 - PX:DiscoveryTimeout 16
 - PX:FileDataCacheSize 17
 - PX:GroupLabel 16
 - PX:IOEncryptionMode 17
 - PX:MemCommit 16
 - PX:MemMax 16
 - PX:-MulticastDiscovery 17
 - PX:NetworkTimeOut 17
 - PX:PolicyVhost 13, 16, 20, 21
 - PX:ProxySourcePortRange 18

- PX:RTPMAuthentication 29
- PX:RTPMAuthorization 29
- PX:RTPMInterface 30
- PX:RTPMPort 29
- PX:-Trace 19
- PX:-TraceFilter 19
- PX:-UnicastTCPDiscovery 17
- PX:-UnicastUDPDiscovery 17
- PX:-UseDirectPath 18
- VIP DNS hostnames and 20
- VM proxy 16
- Xcongc 19
- Xflags 15
- Xnativevmflags 15
- XX:+ProfileAllocatedObjects 20
- XX:+ProfileLiveObjects 20, 30
- XX:+UseConcMarkSweepGC 19
- XX:+UseParallelGC 19
- XX:+UseParNewGC 19
- XX:+UsePauselessGC 19
- XX:+UseSerialGC 19
- XX:GCWarningHistor 19, 30
- XX:PrintGC 63
- XX:PrintGCDetails 63
- XX:PrintGCHistory 19, 30
- committed memory 21
- compute appliance 2
- compute pools 1
 - command line modifications for 20
- container libraries 4
- CPM domain name 12
- customer support ix

D

- data path encryption 2
- DirectPath 2
 - command line options 18
- connections 49
- DP mode 49

- mode constraints 47
 - state 36
- disk space required 6
- DNS server 11
 - notes on policy server VIP 13
- documentation, related viii
- E**
- encryption 2, 17
- F**
- file cache 44
- G**
- garbage collection 2, 19, 56, 59
- GC. *See garbage collection*
- Generational Pauseless Garbage Collection 2
- Generational Pauseless GC 19, 56, 59
- gzip 7
- H**
- heap 2
- heap size
 - specifying maximum 22
 - Xmx option 22
- host servers 1
- HP-UX 7
- I**
- installation 8
 - disk space required 6
 - prerequisites 7
 - RPM 10
 - verification 10
- IP multicast discovery 12
 - command line options for 21
 - verification 12
- IP unicast discovery 11, 20
 - command line options with VIP addresses 20
 - verification 11
- J**
- JAVA_HOME system variable 12
- JDK 65
- L**
- launch parameter control 22
- library mapping 24
 - file formatting rules 25
- Linux 6
- lock contention 4
- lock management scheme 4

M

- MaxPermSize 22
- MemCommit 16, 22
- memory
 - base 22
 - formula for calculating committed 22
 - maximum permanent generation size 22
 - MaxPermSize 22
 - specifying committed 21
- monitors 53

N

- network attached processing 1
- NFS servers 7
- NTP server 7

O

- operating systems
 - AIX 6
 - Linux 6
 - Solaris 6
- optimistic thread concurrency 4
- outgoing RPC profile 42

P

- pauseless garbage collection 2
- platforms supported 2
- policy server 11
- proxy
 - encrypt data path 2

R

- REALTime Performance Monitor 27
 - class allocation 55
 - file cache 44
 - HotSpot flags 36, 37
 - Java command line configuration 29
 - memory 55
 - monitors 53
 - password database utility 30
 - security.conf 31
 - system call profile 41
 - threads list 38
 - tick profiler 52
 - user authentication and authorization 29
 - Web browser support 28
- REALTime Performance Monitor (RTPM) 27
- related documentation viii
- Resin
 - configuring Azul VM for 69
 - JNI libraries 71
- RPC 42

RPM installations 10

S

Solaris 6

stack trace 38

system call profile 41

system call trace 28

system requirements 6

system variables

 JAVA_HOME 12

T

TCP 13,17

thread concurrency 4

thread deadlocks 38

threads 4

threads list 38

tick profiler 28,52

U

UDP 13,17,49

unicast IP discovery

 using TCP 13,17

 using UDP 13,17

V

verifying IP multicast discovery 12

verifying IP unicast discovery 11

virtual host name 11

virtual IP address (VIP) 11

VM proxy 12

 IP unicast discovery and 11

W

Web browser support 28

WebSphere-compatible JDK 65

Notes

Notes

ABOUT AZUL SYSTEMS

Azul Systems has pioneered the industry's first network attached processing solution designed to enable unbound compute™ resources for Java and J2EE based enterprise applications. Azul Compute Appliances eliminate capacity planning at the application level and much of the cost and complexity associated with the conventional delivery of computing resources. More information about Azul Systems can be found at www.azulsystems.com.

Copyright © 2005–2009, Azul Systems, Inc. All rights reserved. Azul Systems, Azul, the Azul arch logo, unbound compute, and Vega are trademarks of Azul Systems Inc. in the United States and other countries. Linux is a registered trademark of Linus Torvalds. RedHat is the property of Red Hat, Inc. Sun, Sun Microsystems, Solaris, J2EE, J2SE, Java and all Java based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. Microsoft and Windows are registered trademarks of Microsoft Corporation in the United States and/or other countries. Other marks are the property of their respective owners and are used here only for identification purposes. Products and specifications discussed in this document may reflect future versions and are subject to change by Azul Systems without notice.

