Red Hat Enterprise MRG 2.0

Tuna User Guide

Using Tuna to perform advanced tuning procedures for the MRG Realtime component of the Red Hat Enterprise MRG distributed computing platform

Lana Brindley

Alison Young

Red Hat Enterprise MRG 2.0 Tuna User Guide
Using Tuna to perform advanced tuning procedures for the MRG Realtime component of the Red Hat Enterprise MRG distributed computing platform
Edition 1

Author	Lana Brindley	*lbrindle@redhat.com*
Author	Alison Young	*alyoung@redhat.com*

1801 Varsity Drive
Raleigh, NC 27606-2072 USA
Phone: +1 919 754 3700
Phone: 888 733 4281
Fax: +1 919 754 3701

This book contains information on using the Tuna program to perform advanced tuning procedures for the MRG Realtime component of the Red Hat Enterprise MRG distributed computing platform. For more information on tuning, see the *MRG Realtime Tuning Guide*.

Preface

Red Hat Enterprise MRG

This book contains basic installation and usage information for Tuna. Tuna was developed for tuning the MRG Realtime component of Red Hat Enterprise MRG, but can also be used to tune standard Red Hat Enterprise Linux systems. Red Hat Enterprise MRG is a high performance distributed computing platform consisting of three components:

1. *M*essaging — Cross platform, high performance, reliable messaging using the Advanced Message Queuing Protocol (AMQP) standard.

2. *R*ealtime — Consistent low-latency and predictable response times for applications that require microsecond latency.

3. *G*rid — Distributed High Throughput (HTC) and High Performance Computing (HPC).

All three components of Red Hat Enterprise MRG are designed to be used as part of the platform, but can also be used separately.

Tuna

Tuna is a tool that can be used to adjust scheduler tunables such as scheduler policy, RT priority and CPU affinity. It also allows the user to see the results of these changes.

Threads and IRQ handlers are able to be tuned. It is also possible to isolate CPU cores and sockets, moving all threads away from them so that a new, more important set of threads can run exclusively.

Tuna provides a graphical user interface (GUI). The GUI displays the CPU topology on one screen, which helps identify problems. It also allows changes to made to running threads, and see the results of those changes immediately.

Most Tuna operations can be performed on either the command line, or in the GUI.

Performing tuning tasks using traditional Linux tools can be daunting and complicated. Tuna reduces that complexity and provides powerful tools for getting the most of the MRG Realtime system.

For more information about MRG Realtime, see the *MRG Realtime Installation Guide*, *MRG Realtime Tuning Guide*, and the *MRG Realtime Reference Guide*.

1. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the *Liberation Fonts*[1] set. The Liberation Fonts set is also used in HTML editions if the set is installed on your system. If not, alternative but equivalent typefaces are displayed. Note: Red Hat Enterprise Linux 5 and later includes the Liberation Fonts set by default.

[1] https://fedorahosted.org/liberation-fonts/

1.1. Typographic Conventions

Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

`Mono-spaced Bold`

Used to highlight system input, including shell commands, file names and paths. Also used to highlight keycaps and key combinations. For example:

> To see the contents of the file **`my_next_bestselling_novel`** in your current working directory, enter the **`cat my_next_bestselling_novel`** command at the shell prompt and press **`Enter`** to execute the command.

The above includes a file name, a shell command and a keycap, all presented in mono-spaced bold and all distinguishable thanks to context.

Key combinations can be distinguished from keycaps by the hyphen connecting each part of a key combination. For example:

> Press **`Enter`** to execute the command.
>
> Press **`Ctrl`**+**`Alt`**+**`F2`** to switch to the first virtual terminal. Press **`Ctrl`**+**`Alt`**+**`F1`** to return to your X-Windows session.

The first paragraph highlights the particular keycap to press. The second highlights two key combinations (each a set of three keycaps with each set pressed simultaneously).

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in **`mono-spaced bold`**. For example:

> File-related classes include **`filesystem`** for file systems, **`file`** for files, and **`dir`** for directories. Each class has its own associated set of permissions.

Proportional Bold

This denotes words or phrases encountered on a system, including application names; dialog box text; labeled buttons; check-box and radio button labels; menu titles and sub-menu titles. For example:

> Choose **System** → **Preferences** → **Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, click the **Left-handed mouse** check box and click **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).
>
> To insert a special character into a **gedit** file, choose **Applications** → **Accessories** → **Character Map** from the main menu bar. Next, choose **Search** → **Find…** from the **Character Map** menu bar, type the name of the character in the **Search** field and click **Next**. The character you sought will be highlighted in the **Character Table**. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to your document and choose **Edit** → **Paste** from the **gedit** menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in proportional bold and all distinguishable by context.

`Mono-spaced Bold Italic` or ***`Proportional Bold Italic`***

Whether mono-spaced bold or proportional bold, the addition of italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

> To connect to a remote machine using ssh, type **ssh *username*@*domain.name*** at a shell prompt. If the remote machine is **example.com** and your username on that machine is john, type **ssh john@example.com**.
>
> The **mount -o remount *file-system*** command remounts the named file system. For example, to remount the **/home** file system, the command is **mount -o remount /home**.
>
> To see the version of a currently installed package, use the **rpm -q *package*** command. It will return a result as follows: ***package-version-release***.

Note the words in bold italics above — username, domain.name, file-system, package, version and release. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

> Publican is a *DocBook* publishing system.

1.2. Pull-quote Conventions

Terminal output and source code listings are set off visually from the surrounding text.

Output sent to a terminal is set in **mono-spaced roman** and presented thus:

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

Source-code listings are also set in **mono-spaced roman** but add syntax highlighting as follows:

```
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;

public class ExClient
{
   public static void main(String args[])
       throws Exception
   {
      InitialContext iniCtx = new InitialContext();
      Object         ref    = iniCtx.lookup("EchoBean");
      EchoHome       home   = (EchoHome) ref;
      Echo           echo   = home.create();

      System.out.println("Created Echo");

      System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
   }
}
```

1.3. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.

Note

Notes are tips, shortcuts or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

Important

Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring a box labeled 'Important' will not cause data loss but may cause irritation and frustration.

Warning

Warnings should not be ignored. Ignoring warnings will most likely cause data loss.

2. Getting Help and Giving Feedback

2.1. Do You Need Help?

If you experience difficulty with a procedure described in this documentation, visit the Red Hat Customer Portal at *http://access.redhat.com*. Through the customer portal, you can:

- search or browse through a knowledgebase of technical support articles about Red Hat products.
- submit a support case to Red Hat Global Support Services (GSS).
- access other product documentation.

Red Hat also hosts a large number of electronic mailing lists for discussion of Red Hat software and technology. You can find a list of publicly available mailing lists at *https://www.redhat.com/mailman/listinfo*. Click on the name of any mailing list to subscribe to that list or to access the list archives.

2.2. We Need Feedback!

If you find a typographical error in this manual, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in Bugzilla: *http://bugzilla.redhat.com/* against the product **Red Hat Enterprise MRG.**

When submitting a bug report, be sure to mention the manual's identifier: *Tuna_User_Guide*

If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.

Installing Tuna

Tuna is currently only available through the MRG Realtime channels on the *Red Hat Network*[1] (RHN).

Procedure 1.1. Download and Install Tuna

1. In order to install Tuna you will need to have registered your system with *Red Hat Network*[2], and subscribe to one of the following channels:
   - MRG Realtime v. 1 (for RHEL 5 Server 64-bit x86_64)
   - MRG Realtime v. 1 (for RHEL 5 Server 32-bit i686)
2. Tuna requires the following packages:
   - **python-linux-procfs**
   - **python-schedutils**
   - **python-ethtool**

   In order to use the graphical user interface, the following packages are also required:
   - **pygtk2**
   - **pygtk2-libglade**
3. Once you have registered your system with Red Hat Network, and subscribed to the appropriate channel, Tuna can be installed using the **yum** command. This will install all the necessary dependencies:

```
# yum install tuna
```

4. Although Tuna can be run as an unprivileged user, not all processes will be available for configuration. For this reason, in most cases you will need to run Tuna as the root user:

```
# tuna
```

   With the appropriate privileges, Tuna could also be run with the **sudo** command:

```
$ sudo tuna
```

Note

If you find that yum is not installing all the dependencies you require, make sure that you have registered your system with *Red Hat Network*[3].

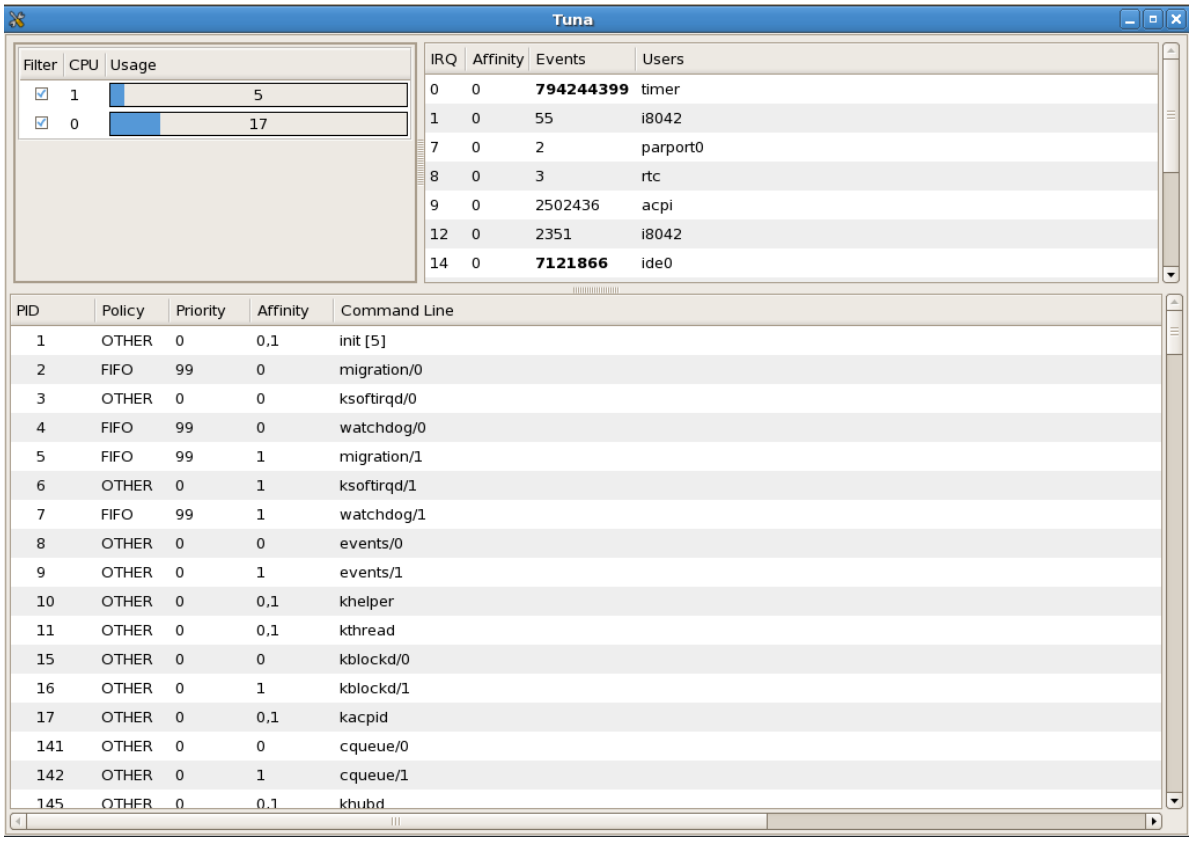
[1] https://rhn.redhat.com/help/about.pxt
[3] https://rhn.redhat.com/help/about.pxt

Using the Graphical User Interface

Tuna can be used either from the command line interface, or the graphical interface. Both provide the same range of functionality. This chapter covers the graphical user interface.

2.1. Reviewing the System

The main Tuna screen shows the current state of the system.



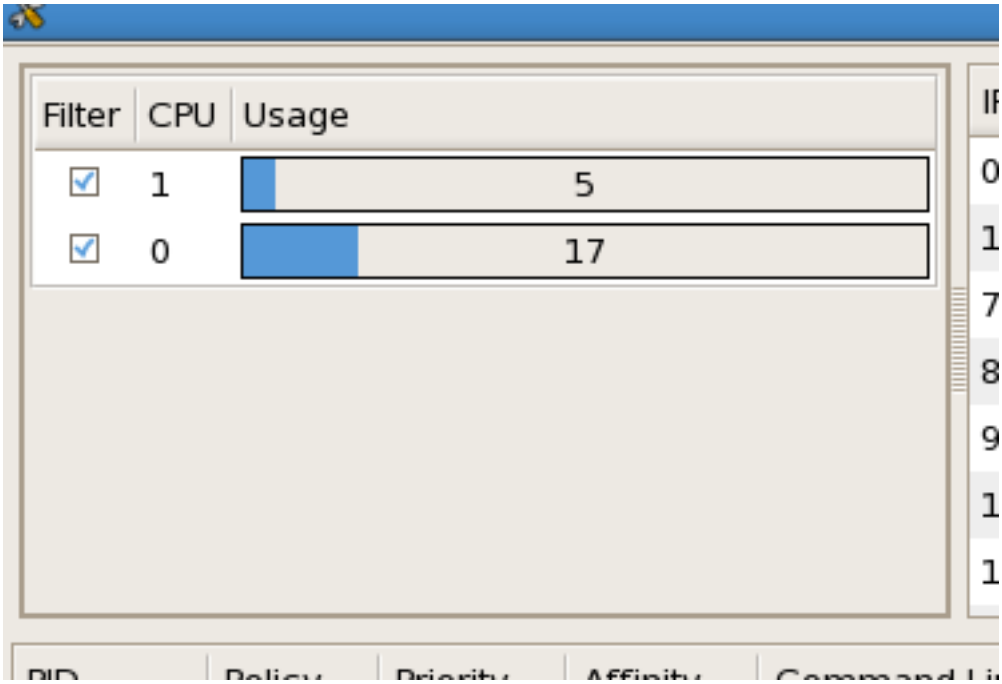
Procedure 2.1. Reviewing the System in the GUI

1. The main Tuna window is divided into three sections, for CPU, IRQ, and process information. The sections are divided by grab bars for adjustment. The window itself can also be resized.

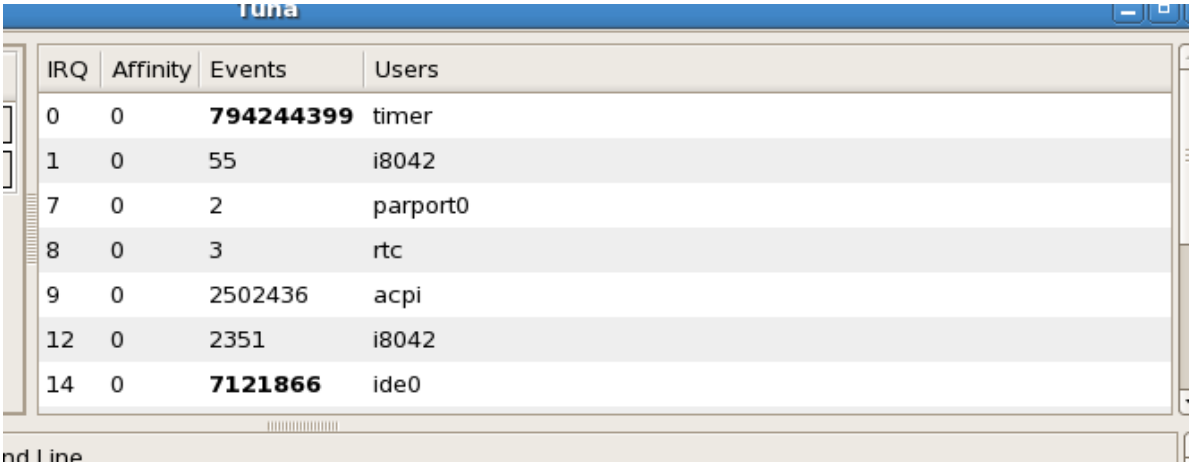
   As values in each of the sections change, the entries are shown in bold.

2. The CPU list shows all online CPUs and their current usage.

   The check-box beside the name of the CPU is used to filter the task list at the bottom of the window. Only tasks and IRQs that belong to checked CPUs will be displayed.



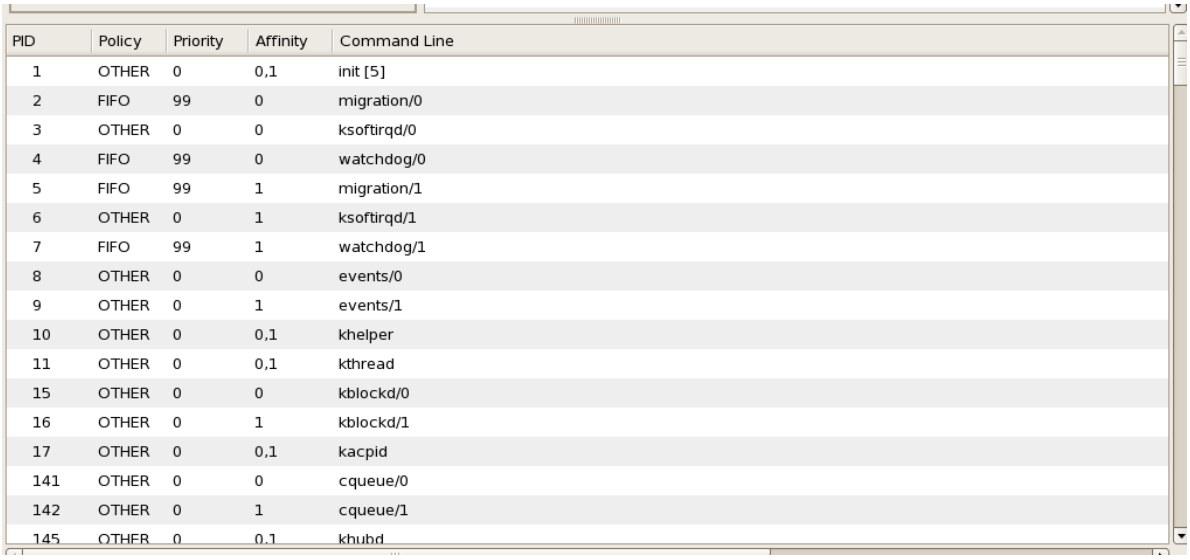
3. The IRQ list shows all active interrupt requests (IRQs), their process ID (PID) and policy and priority information.



4. The task list shows all running tasks.

   When a process is threaded, the task list shows the parent thread with all the children threads collapsed below it. Click on the arrow to the left of the process to expand the thread.

   The task list has a right-click menu. Select **Hide kernel threads** to hide all kernel threads, and see only user threads. Click **Hide kernel threads** again to restore the kernel threads. Similarly, **Hide user threads** will hide all user threads and show only kernel threads. Clicking **Hide user threads** again will restore the user threads.



PID	Policy	Priority	Affinity	Command Line
1	OTHER	0	0,1	init [5]
2	FIFO	99	0	migration/0
3	OTHER	0	0	ksoftirqd/0
4	FIFO	99	0	watchdog/0
5	FIFO	99	1	migration/1
6	OTHER	0	1	ksoftirqd/1
7	FIFO	99	1	watchdog/1
8	OTHER	0	0	events/0
9	OTHER	0	1	events/1
10	OTHER	0	0,1	khelper
11	OTHER	0	0,1	kthread
15	OTHER	0	0	kblockd/0
16	OTHER	0	1	kblockd/1
17	OTHER	0	0,1	kacpid
141	OTHER	0	0	cqueue/0
142	OTHER	0	1	cqueue/1
145	OTHER	0	0,1	khubd

2.2. CPU Tuning

Procedure 2.2. CPU Tuning in the GUI

1. To isolate a CPU, right click on the selected CPU, and select **Isolate CPU** from the menu. This will cause all tasks currently running on that CPU to move to the next available CPU. This is achieved through removing the selected CPU from the current affinity mask of all threads, so that they no longer see that CPU as being available.

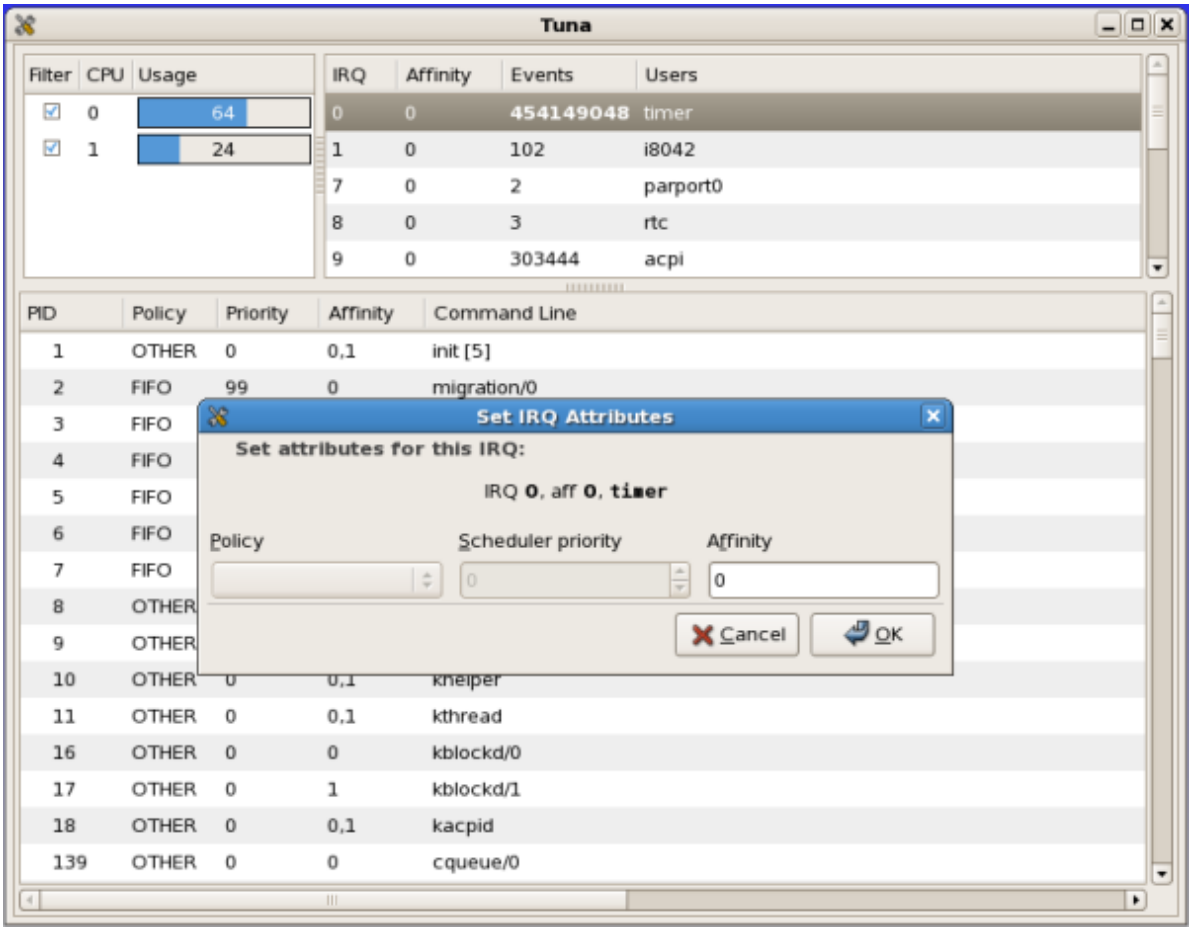
2. To include a CPU, right click on the selected CPU, and select **Include CPU** from the menu. This will allow tasks to run on that CPU.

3. To restore a CPU, right click on the selected CPU, and select **Restore CPU** from the menu. This will restore that CPU to its previous configuration.

2.3. IRQ Tuning

Procedure 2.3. IRQ Tuning in the GUI

1. Right click on an IRQ and select **Set IRQ Attributes** to open the IRQ Attributes dialog box.



2. The IRQ Attributes dialog shows current information about the IRQ. It has three adjustable attributes:

    1. Scheduling Policy

        A drop down list of the available policies.

        SCHED_OTHER is the default policy. **SCHED_FIFO** is a first in/first out realtime policy. A **SCHED_FIFO** policy with a priority of *1* will always run ahead of **SCHED_OTHER**. **SCHED_RR** is a policy where threads of equal priority are treated in a round-robin fashion.

    2. Scheduler Priority

        A drop down list of the available priorities. This attribute will be disabled if the selected IRQ cannot have a set priority.

        Scheduler priorities range from 99 (highest) to 1 (lowest). Priorities can be set for threads that use the **SCHED_FIFO** or **SCHED_RR** policies.

    3. Affinity

        A numeric list of CPUs on which the IRQ can be run. This entry can be in the form of a comma-delimited list of CPU numbers, a range separated by a hyphen, or a combination of both. For example: **0, 2-4, 7, 8**. This would instruct the IRQ to run on CPUs 0, 2, 3, 4, 7 and 8.

This field will also accept hex masks. Hex masks must be preceded by **0x** in order to be recognized and interpreted correctly. Hex masks that do not use that format will be interpreted as a decimal CPU number.

Note

See the *MRG Realtime Reference Guide* for more information on policy, priority, and affinity.

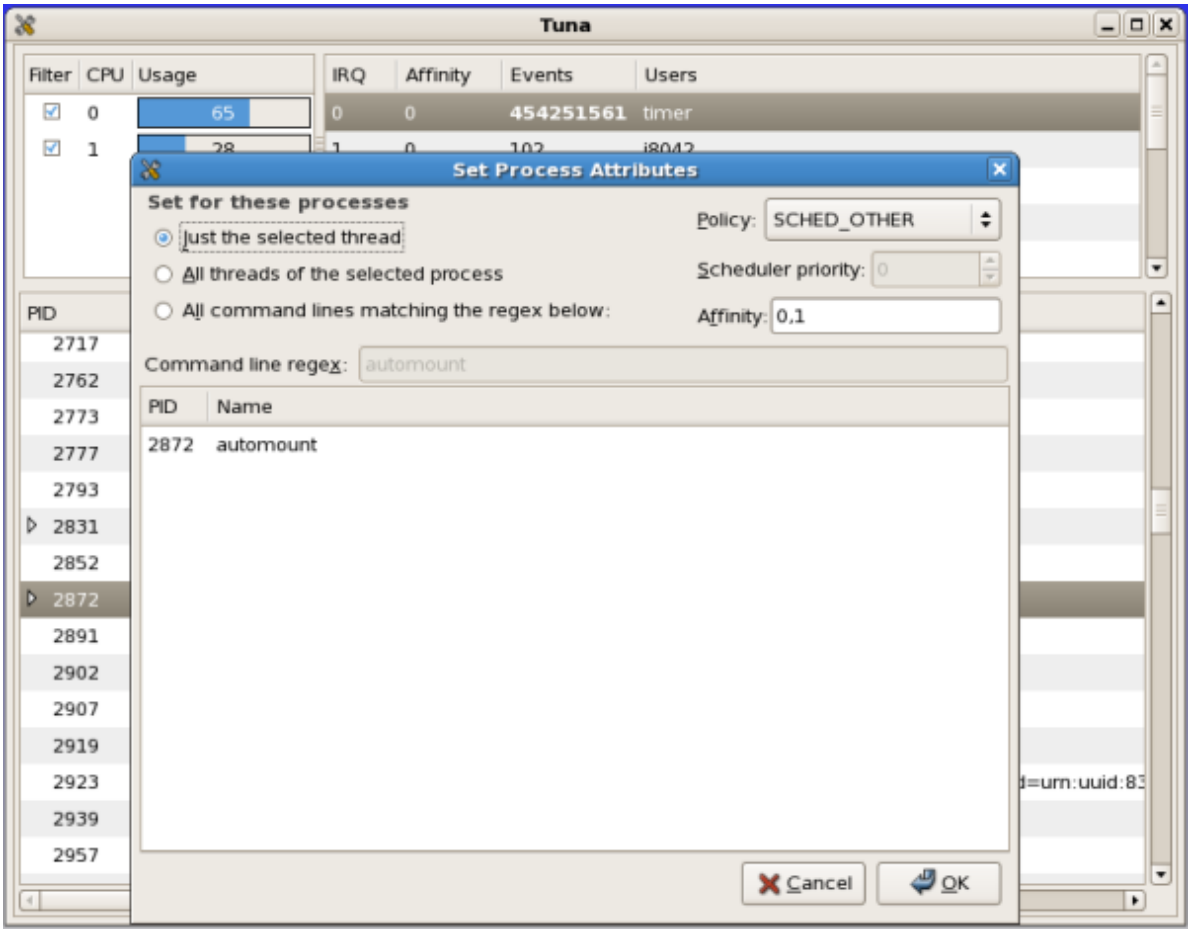
Note

Moving IRQs and threads by specifying the CPUs they are to run on can be time consuming and difficult. Tuna also offers the ability to select threads and IRQs, and drag and drop them over the desired CPUs. This method can make changing the topology much easier.

2.4. Task Tuning

Procedure 2.4. Task Tuning in the GUI

1. Right click on a task and select **Set Process Attributes** to open the Process Attributes dialog box.



2. The Process Attributes dialog shows current information about the task. It allows you to set scheduling policy, scheduler priority, and CPU affinity for a task or set of tasks.

1. Thread Selection

   Just the selected thread is selected by default. If the task has more than one thread, use **All threads of the selected process** to make changes to all of the threads for that task. To use a regular expression (regex) to search for tasks, select **All command lines matching the regex below:**. This will activate the **Command line regex:** field and you can enter the regex. This field supports the ***** and **?** wildcards, and will match the entire command line. The task list will update to show only those tasks that match the regex.

2. Policy, Priority and Affinity

   The **Policy** drop down box contains the available scheduling policy options.

   The **Scheduler Priority** drop down box contains the available priorities. This attribute will be disabled if the selected tasks cannot have a set priority.

   The **Affinity** field contains a numeric list of CPUs on which the selected tasks can be run. This entry can be in the form of a comma-delimited list of CPU numbers, a range using square brackets, or a combination of both.

3. Task List

   This shows a list of the the tasks currently being adjusted based on the thread and regex selections made.

2.5. Examples for Using Tuna with the Graphical User Interface

Example 2.1. Using Tuna with the Graphical User Interface

This example uses a system with four or more processors. Two applications need to be run - **Foo** and **Bar**. The applications need to be run on dedicated processors - processor 0 for **Foo** and processor 1 for **Bar**.

1. Move everything off the chosen processors. Right-click on **CPU 0** in the CPU list and select **Isolate CPU** from the menu. Repeat for **CPU 1**. The task list shows that no tasks are running on those processors.

2. **Foo** is a single task with several threads. The task and all its threads need to run on **CPU 0**. Find **Foo** in the process list, right-click on it and choose **Set process attributes** from the menu. In the **Set Process Attributes** dialog, select the radio button for **All threads of the selected process**. In the **Affinity** text box, change the text to **0**. The scheduling policy and scheduler priority can also be adjusted if required. Click on **OK** to save the changes and close the dialog box.

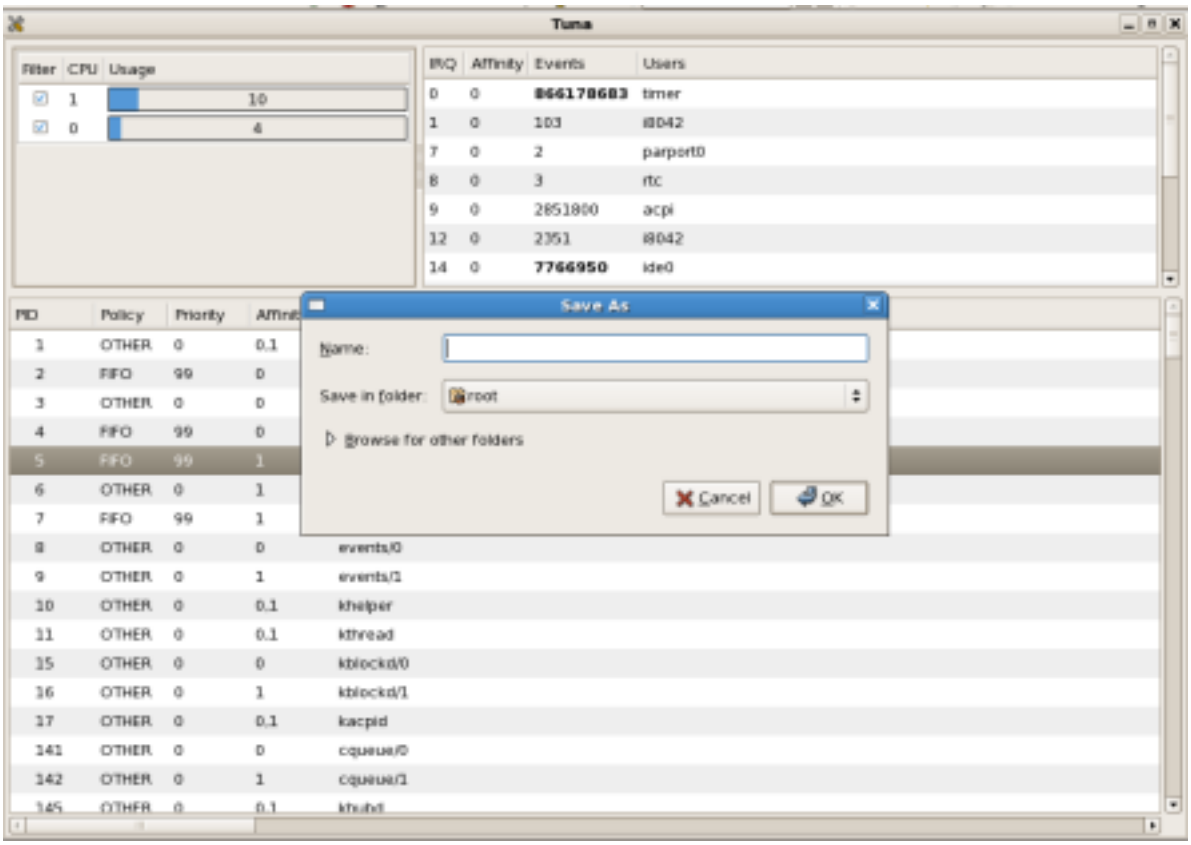
3. **Bar** is an application that has **--none** as its first command line argument. Right-click anywhere in the task list and choose **Set process attributes** from the menu. In the dialog, select the radio-button for **All command lines matching the regex below:**. Type **bar --none *** in the **Command line regex** text box. The task list in the dialog box will update to include the matching processes and any associated threads. Change the **Affinity** to **0**. Make any changes for the scheduler and priority. Click on **OK** to save your changes and close the dialog box.

2.6. Saving Changes

Procedure 2.5. Saving Changes in the GUI

1. Right-click in the Tuna graphical interface, and select the **Save kthreads tunings** menu item.

2. Tuna will prompt for a filename and directory. Enter a filename and select the location to save the file. Select **OK** to continue.



Important

This method will not save every option that is able to be changed with Tuna. This will save the kernel thread changes only. Any processes that are not currently running when they are changed will not be saved.

Using the Command Line Interface

Tuna can be used either from the command line interface, or the graphical interface. Both provide the same range of functionality. This chapter covers the command line interface.

Use the **--help** option to see all the available options:

```
# tuna --help
Usage: tuna [OPTIONS]
        -h, --help                              Give this help list
        -g, --gui                               Start the GUI
        -c, --cpus=CPU-LIST                     CPU-LIST affected by commands
        -C, --affect_children                   Operation will affect children threads
        -f, --filter                            Display filter the selected entities
        -i, --isolate                           Move all threads away from CPU-LIST
        -I, --include                           Allow all threads to run on CPU-LIST
        -K, --no_kthreads                       Operations will not affect kernel threads
        -m, --move                              Move selected entities to CPU-LIST
        -p, --priority=[POLICY]:RTPRIO          Set thread scheduler tunables: POLICY and
 RTPRIO
        -P, --show_threads                      Show thread list
        -Q, --show_irqs                         Show IRQ list
        -q, --irqs=IRQ-LIST                     IRQ-LIST affected by commands
        -s, --save=FILENAME                     Save kthreads sched tunables to FILENAME
        -S, --sockets=CPU-SOCKET-LIST           CPU-SOCKET-LIST affected by commands
        -t, --threads=THREAD-LIST               THREAD-LIST affected by commands
        -U, --no_uthreads                       Operations will not affect user threads
        -v, --version                           Show version
        -W, --what_is                           Provides help about selected entities
        -x, --spread                            Spread selected entities over CPU-LIST
```

When passing commands to Tuna using the command line, it is possible to pass multiple commands in one line and Tuna will process the commands sequentially:

```
tuna --socket 0 --isolate \
   --thread my_real_time_app --move \
   --irq serial --socket 1 --move \
   --irq eth* --socket 2 --spread \
   --show_threads --show_irqs
```

The above command will distribute load across a four socket system. Commands such as this can be added to the initialization scripts of applications to serve as a configuration command.

Table 3.1. Tuna Options

Tuna Options	
--help	Display the help list
--gui	Start the graphical user interface
--cpus=*CPU-LIST*	The CPUs to be controlled by Tuna. The list will remain in effect until a new list is specified
--affect_children	Operation will affect children threads as well as the parent threads
--filter	Filter the display to only show the affected entities
--isolate *CPU-LIST*	Move all threads away from the specified CPUs

Tuna Options	
--include *CPU-LIST*	Allow all threads to run on the specified CPUs
--no_kthreads	Operation will not effect kernel threads
--move	Move selected entities to the specified CPUs
--priority=*[POLICY]:RTPRIO*	Set the thread to have the specified scheduler policy and priority
--show_threads	Show the thread list
--show_irqs	Show the IRQ list
--irqs *IRQ-LIST*	Specify the list of IRQs that are to be affected by commands. The list will remain in effect until a new list is specified. IRQs can be added to the list by using **+** and removed from the list by using **-**
--save *FILENAME*	Save the kernel threads schedules to a file called *FILENAME*
--sockets=*CPU-SOCKET-LIST*	The CPU sockets to be controlled by Tuna. This option takes into account the CPU topology, such as the cores that share a single processor cache, and that are on the same physical chip.
--threads=*THREAD-LIST*	The threads to be controlled by Tuna. The list will remain in effect until a new list is specified. Threads can be added to the list by using **+** and removed from the list by using **-**
--no_uthreads	Operation will not affect user threads
--version	Show the current version of the Tuna package
--what_is	To see further help on selected entities
--spread	Spread the specified threads evenly between the selected CPUs

3.1. Reviewing the System

Tuna can show what is happening currently on the system, before changes are made.

Procedure 3.1. Reviewing the System in the CLI

1. Use the **--show_threads** command to view the current policies and priorities:

```
# tuna --show_threads
    thread
pid SCHED_ rtpri affinity            cmd
1      OTHER     0      0,1           init
2       FIFO    99        0     migration/0
3      OTHER     0        0     ksoftirqd/0
4       FIFO    99        0      watchdog/0
```

2. Use the **--show_irqs** command to view the current interrupts and their affinity:

```
# tuna --show_irqs
# users            affinity
```

```
0 timer                  0
1 i8042                  0
7 parport0               0
```

3.2. CPU Tuning

Procedure 3.2. CPU Tuning in the CLI

To tune CPUs on the command line in Tuna, first specify the list of CPUs to be affected, and then give the action to be performed.

1. Specify the list of CPUs to be affected by the command:

```
# tuna --cpus=CPU-LIST --COMMAND
```

2. To isolate a CPU:

```
# tuna --cpus=CPU-LIST --isolate
```

This command will cause all tasks currently running on that CPU to move to the next available CPU.

3. To include a CPU:

```
# tuna --cpus=CPU-LIST --include
```

This command will allow threads to run on the specified CPU.

3.3. IRQ Tuning

Procedure 3.3. IRQ Tuning in the CLI

1. Specify the list of IRQs to be affected by the command:

```
# tuna --irqs=IRQ-LIST --COMMAND
```

2. To move an interrupt to a specified CPU:

```
# tuna --irqs=IRQ-LIST --cpus=CPU --move
```

3.4. Task Tuning

Procedure 3.4. Task Tuning in the CLI

1. To change policy and priority information on threads, use the **`--priority=[POLICY]:RTPRIO`** command, where *`POLICY`* is the new policy and *`RTPRIO`* is the new priority:

```
# tuna --threads 7861 --priority=RR:40
```

Policy can be either **RR** for round-robin, **FIFO** for first in/first out, or **OTHER** for the default policy.

Priority is a number between 1 (lowest priority) and 99 (highest priority).

For more information on scheduler policy and priority, see the *MRG Realtime Reference Guide*.

2. Use the **--show_threads** command to check the changes:

```
# tuna --threads 7861 --show_threads

pid  SCHED_ rtpri affinity voluntary nonvoluntary   cmd
7861     RR    40     0xff     33318        16957   IRQ-4 serial
```

3.5. Examples for Using Tuna with the Command Line Interface

Example 3.1. Using Tuna with the Command Line Interface

This example uses a system with four or more processors. All **ssh** threads need to run on CPUs 0 and 1. All **http** threads need to run on CPUs 2 and 3.

```
# tuna --cpus=0,1 --threads ssh* --move --cpus 2,3 --threads http* --move
```

This command will:

1. Select CPUs 0 and 1
2. Select all threads that begin with **ssh**
3. Move the selected threads to the selected CPUs. Tuna does this by setting the affinity mask of threads starting with **ssh** to the appropriate CPUs. The CPUs can be expressed numerically as 0 and 1; hex mask as 0x3; binary as 11
4. Reset the CPU list to 2 and 3
5. Select all threads that begin with **http**
6. Move the selected threads to the selected CPUs. Tuna does this by setting the affinity mask of threads starting with **http** to the appropriate CPUs. The CPUs can be expressed numerically as 2 and 3; hex mask as 0xC; binary as 1100

Example 3.2. Using the **show_threads** Command to View the Current Configurations

This example uses the **show_threads** command to display the current configuration, and test if the requested changes have worked as expected.

```
# tuna -t gnome-sc* -P -c0 -mP -c1 -mP -c+0 -mP
                       thread       ctxt_switches
     pid SCHED_ rtpri affinity voluntary nonvoluntary             cmd
    3861   OTHER     0      0,1     33997           58 gnome-screensav
                       thread       ctxt_switches
     pid SCHED_ rtpri affinity voluntary nonvoluntary             cmd
    3861   OTHER     0        0     33997           58 gnome-screensav
                       thread       ctxt_switches
```

```
  pid SCHED_ rtpri affinity voluntary nonvoluntary            cmd
3861   OTHER     0        1     33997           58 gnome-screensav
                   thread       ctxt_switches
  pid SCHED_ rtpri affinity voluntary nonvoluntary            cmd
3861   OTHER     0      0,1     33997           58 gnome-screensav
```

This command will:

1. Select all threads that begin with **gnome-sc**
2. Show the selected threads, to check their affinity mask and RT priority
3. Select CPU 0
4. Move the **gnome-sc** threads to the selected CPU (CPU 0)
5. Show the result of the move
6. Reset the CPU list to CPU 1
7. Move the **gnome-sc** threads to the selected CPU (CPU 1)
8. Show the result of the move
9. Add CPU 0 to the CPU list
10. Move the **gnome-sc** threads to the selected CPUs (CPUs 0 and 1)
11. Show the result of the move

3.6. Saving Changes

Procedure 3.5. Saving Changes in the CLI

- Use the **--save** or **-s** parameter with a descriptive filename to save the current configuration:

```
# tuna --save=FILENAME
```

Important

This method will not save every option that is able to be changed with Tuna. This will save the kernel thread changes only. Any processes that are not currently running when they are changed will not be saved.

Using Testing Tools with Tuna

Tuna's functionality is enhanced and expanded by the addition of several testing tools. The most important of these is Cyclictest, which is designed specifically to locate and identify latencies in a real-time system. Oscilloscope uses data provided to it and presents it in graph form. By feeding data to the oscilloscope from cyclictest, it graphically displays latencies as they occur.

Cyclictest is available in the **rt-tests** package. Ensure you are registered with the Red Hat Network, and subscribed to the appropriate MRG Realtime channel. See *Chapter 1, Installing Tuna*. The package can then be installed using the following command:

```
# yum install rt-tests
```

The oscilloscope is available in the **oscilloscope** package. It requires the following dependencies:

- **pygtk2**
- **python-matplotlib**
- **python-numeric**

```
# yum install oscilloscope
```

4.1. Cyclictest

Cyclictest is used to measure the maximum latency of certain events over time. Ideally, the tool would be run over a period of time, under a variety of different stress levels, to determine where the highest latencies lie.

Use the **--help** option to see all the available options:

```
# cyclictest --help
cyclictest V 0.66
Usage:
cyclictest <options>

-a [NUM] --affinity        run thread #N on processor #N, if possible
                           with NUM pin all threads to the processor NUM
-b USEC  --breaktrace=USEC send break trace command when latency > USEC
-B       --preemptirqs     both preempt and irqsoff tracing (used with -b)
-c CLOCK --clock=CLOCK     select clock
                           0 = CLOCK_MONOTONIC (default)
                           1 = CLOCK_REALTIME
-C       --context         context switch tracing (used with -b)
-d DIST  --distance=DIST   distance of thread intervals in us default=500
-D       --duration=t      specify a length for the test run
                           default is in seconds, but 'm', 'h', or 'd' maybe added
                           to modify value to minutes, hours or days
-E       --event           event tracing (used with -b)
-f       --ftrace          function trace (when -b is active)
-h       --histogram=US    dump a latency histogram to stdout after the run
                           (with same priority about many threads)
                           US is the max time to be be tracked in microseconds
-i INTV  --interval=INTV   base interval of thread in us default=1000
-I       --irqsoff         Irqsoff tracing (used with -b)
-l LOOPS --loops=LOOPS     number of loops: default=0(endless)
-m       --mlockall        lock current and future memory allocations
-M       --refresh_on_max  delay updating the screen until a new max latency is hit
```

```
-n       --nanosleep       use clock_nanosleep
-N       --nsecs           print results in ns instead of us (default us)
-o RED   --oscope=RED      oscilloscope mode, reduce verbose output by RED
-O TOPT  --traceopt=TOPT   trace option
-p PRIO  --prio=PRIO       priority of highest prio thread
-P       --preemptoff      Preempt off tracing (used with -b)
-q       --quiet           print only a summary on exit
-r       --relative        use relative timer instead of absolute
-s       --system          use sys_nanosleep and sys_setitimer
-t       --threads         one thread per available processor
-t [NUM] --threads=NUM     number of threads:
                           without NUM, threads = max_cpus
                           without -t default = 1
-T TRACE --tracer=TRACER   set tracing function
    configured tracers: unavailable (debugfs not mounted)
-u       --unbuffered      force unbuffered output for live processing
-v       --verbose         output values on stdout for statistics
                           format: n:c:v n=tasknum c=count v=value in us
-w       --wakeup          task wakeup tracing (used with -b)
-W       --wakeuprt        rt task wakeup tracing (used with -b)
-y POLI  --policy=POLI     policy of realtime thread (1:FIFO, 2:RR)
                           format: --policy=fifo(default) or --policy=rr
-S       --smp             Standard SMP testing (equals -a -t -n -m -d0)
                           same priority on all threads.
-U       --numa            Standard NUMA testing (similar to SMP option)
                           thread data structures allocated from local node
```

1. Cyclictest must be run as the root user.

2. Running cyclictest without any parameters will create one test thread with a 1ms interval:

```
# cyclictest
policy: other: loadavg: 0.07 0.19 0.29 3/260 27939

T: 0 (27939) P: 0 I:1000 C:   3279 Min:   1538 Act:1059544 Avg:881375 Max: 1059876
```

   The final column displays the maximum latency.

3. Use the following command to run one test thread per CPU:

```
# cyclictest --smp -p75 -m
policy: fifo: loadavg: 0.01 0.05 0.08 1/338 30074

T: 0 (30073) P:75 I:1000 C:    821 Min:      6 Act:   39 Avg:   22 Max:      44
T: 1 (30074) P:75 I:1500 C:    542 Min:      7 Act:   64 Avg:   48 Max:      73
```

4. Use this command on a NUMA system (an AMD system with more than one memory node):

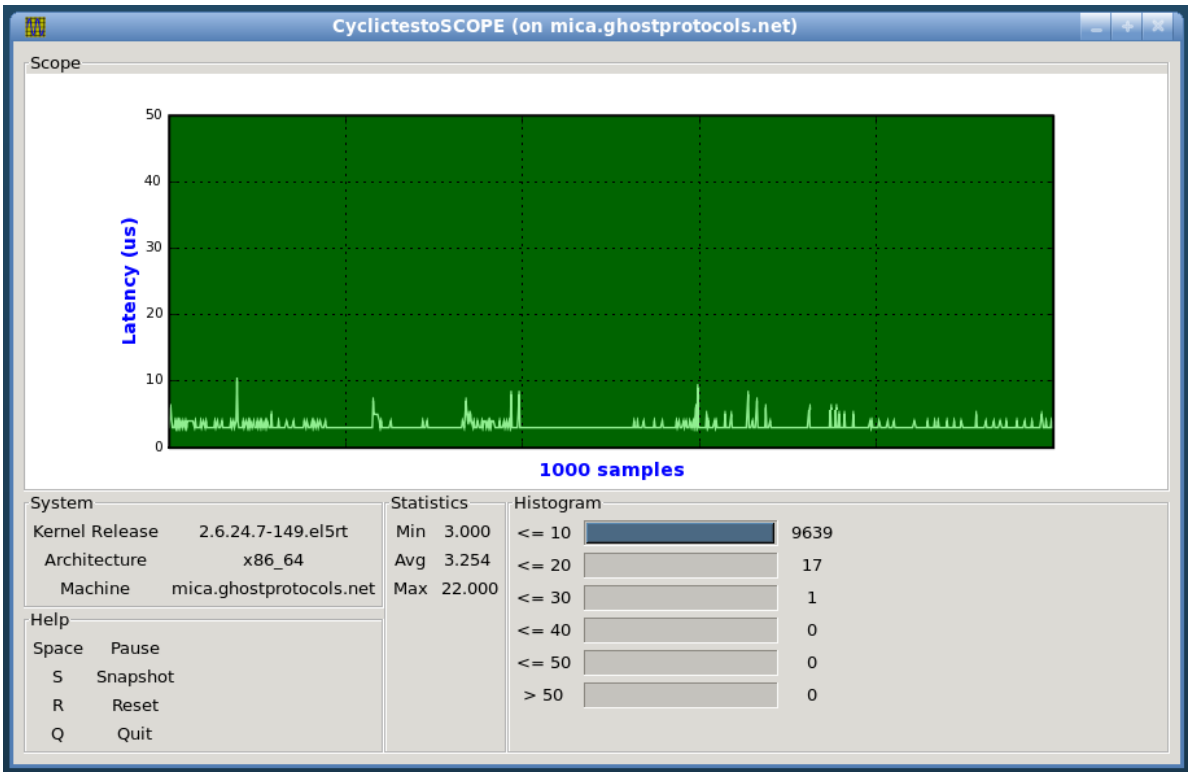
```
# cyclictest --numa -p75 -m
policy: fifo: loadavg: 0.00 0.00 0.00 1/173 25319

T: 0 (25318) P:75 I:1000 C:   2046 Min:      7 Act:    9 Avg:    8 Max:      12
T: 1 (25319) P:75 I:1500 C:   1363 Min:      8 Act:   10 Avg:    9 Max:      24
```

4.2. Oscilloscope

The oscilloscope uses the data produced by cyclictest and pipes it to a continuously updated graph.

1. Start cyclictest with the **-v** (verbose) parameter. Then use a **|** (pipe) to send the output to the oscilloscope:

```
# cyclictest -t1 -n -p99 -v | oscilloscope >/dev/null
```



2. Use the keyboard controls listed in the help section of the oscilloscope to control the output:

   - **space**: Pause the feed, and display a static graph
   - **s**: Create a snapshot of the graph. The image will be saved as a PNG in the current directory.
   - **r**: Reset the oscilloscope
   - **q**: Quit the program

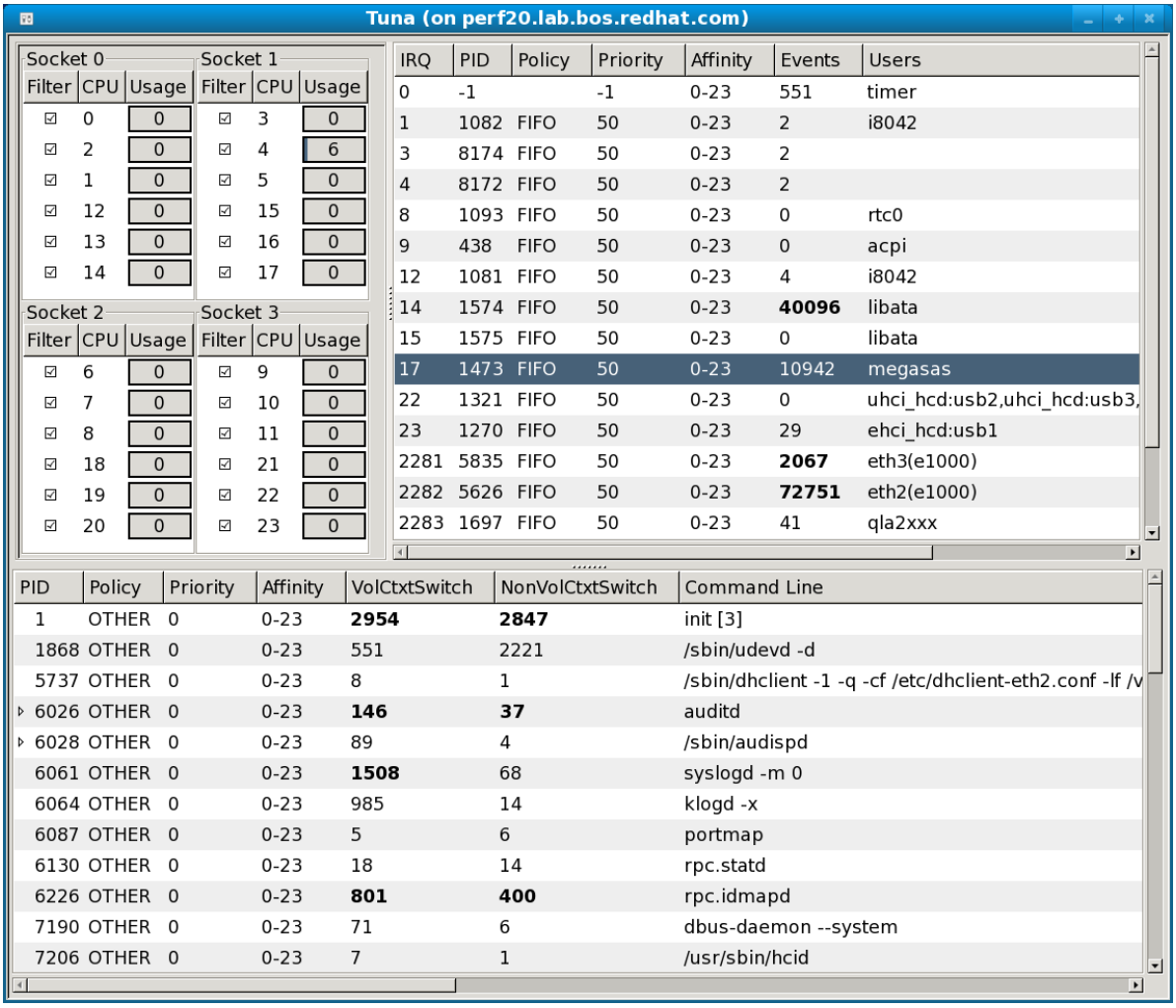
Frequently Asked Questions

Q: How can I save my configuration for threads other than kernel threads?

A: The command line interface can be used to add a series of operations to the startup script of any program. Develop the series of commands for Tuna to run at startup, and pass it to the program as a single command. Threads created after the initial command will inherit the affinity and scheduling policy of the thread that creates it. For an example of an appropriate startup script, see *Chapter 3, Using the Command Line Interface*

Q: Can Tuna handle multiple sockets and multiple cores?

A:



Tuna supports multiple sockets and sockets with multiple cores. If there are multiple cores on a socket, they will often share the cache on that socket.

The Tuna interface groups multiple sockets within a frame, so that operations can be done on whole sockets or on specific cores.

More Information

6.1. Reporting Bugs

If you have determined that the bug is specific to MRG Realtime follow these instructions to enter a bug report:

1. You will need a *Bugzilla*[1] account. You can create one at *Create Bugzilla Account*[2].
2. Once you have a Bugzilla account, log in and click on *Enter A New Bug Report*[3].
3. You will need to identify the product in which the bug occurs. MRG Realtime appears under Red Hat Enterprise MRG in the Red Hat products list. It is important that you choose the correct product that the bug occurs in.
4. Continue to enter the bug information by designating the appropriate component and giving a detailed problem description.

6.2. Further Reading

Red Hat Enterprise MRG and MRG Realtime Product Information
http://www.redhat.com/mrg

MRG Realtime and other Red Hat Enterprise MRG manuals
http://docs.redhat.com/docs/en- US/index.html

Red Hat Knowledgebase
https://access.redhat.com/knowledge/search

Appendix A. Revision History

Revision 1-0 Thu Jun 23 2011 **Alison Young** *alyoung@redhat.com*

Prepared for publishing

Revision 0.1-1 Wed Feb 23 2011 **Alison Young** *alyoung@redhat.com*

Minor XML updates

Revision 0.1-0 Wed Feb 23 2011 **Alison Young** *alyoung@redhat.com*

Fork from 1.3