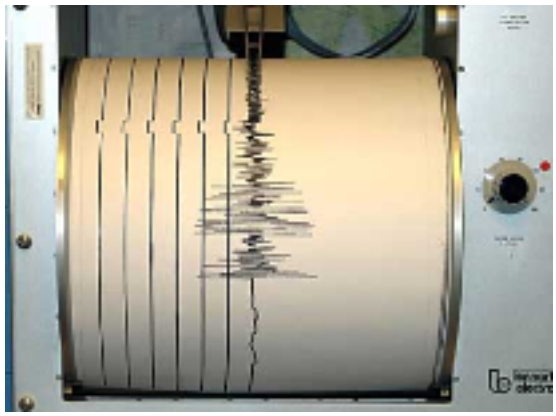
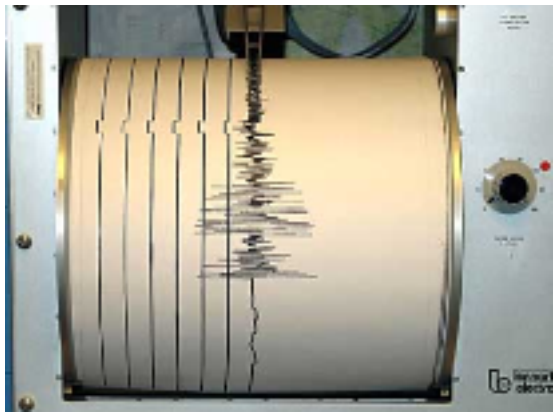
App Trace Roll: Users Guide



Version 4.1 Edition

App Trace Roll: Users Guide : 

Version 4.1 Edition

Published Dec 2005

Table of Contents

Preface ..... **??**

1. Requirements ..... **??**

1.1. Rocks Version ..... **??**

1.2. Hardware Requirements ..... **??**

1.3. Software Requirements ..... **??**

2. Installing the App Trace Roll ..... **??**

2.1. Adding the Roll ..... **??**

3. Using the App Trace Roll ..... **??**

3.1. Starting and Stopping Tracing on Compute Nodes ..... **??**

3.2. Analyzing App Trace Data at Your Site ..... **??**

3.3. Uploading Trace Files to SDSC ..... **??**

4. Tuning the App Trace Roll ..... **??**

4.1. Tuning the App Trace Roll ..... **??**

5. Internals ..... **??**

5.1. SystemTap Internals ..... **??**

5.2. Upload Internals ..... **??**

6. Copyrights ..... **??**

6.1. App Trace ..... **??**

Preface

The App Trace Roll installs and configures system tracing tools that records the size and offset of every read/write system call for all processes that are running on a node.

It only records size and offset information. User data is never accessed.

We see the App Trace Roll being used in two modes.

In the first mode, users will use the App Trace Roll for their personal use to better understand the I/O behavior of their applications. They will use the collected data to help them to tune their application, tune their storage subsystem or both. The Rocks Group will benefit from this usage mode as users will provide feedback on how to make the App Trace Roll more useful.

In the second mode, users will use the App Trace Roll for their personal use *and* upload their trace files to SDSC's server. In the near term, Greg Bruno from the Rocks Group will use these trace files to support his Ph.D. research into parallel file systems. In the long term, the Rocks Group will use these trace files to gain insight on how to provide better storage systems for the Rocks Community.

Chapter 1. Requirements

1.1. Rocks Version

The App Trace Roll is for use with Rocks version 4.1 or later.

This requirement is strict as the App Trace Roll requires *SystemTap* which was introduced in RedHat Enterprise Linux Server 4 update 2 (which Rocks version 4.1 is based upon).

1.2. Hardware Requirements

This roll can only be used with i386 and x86_64 releases of Rocks.

As of 11/15/2005, RedHat has not released *SystemTap* for ia64.

1.3. Software Requirements

The App Trace Roll works out-of-the-box with RedHat kernel version 2.6.9-22. If you update your RedHat kernel, you'll need to download a new *kernel-debuginfo* package from RedHat's FTP site. The version of the *kernel-debuginfo* package must match that of the kernel that is running on the compute nodes.

The *kernel-debuginfo* packages can be found under: *ftp://ftp.redhat.com/pub/redhat/linux/updates/enterprise/4WS/en/os/Debuginfo*. If you download a new *kernel-debuginfo* package, follow the procedure Adding Packages to Compute Nodes[1] in order to install this package onto your compute nodes.

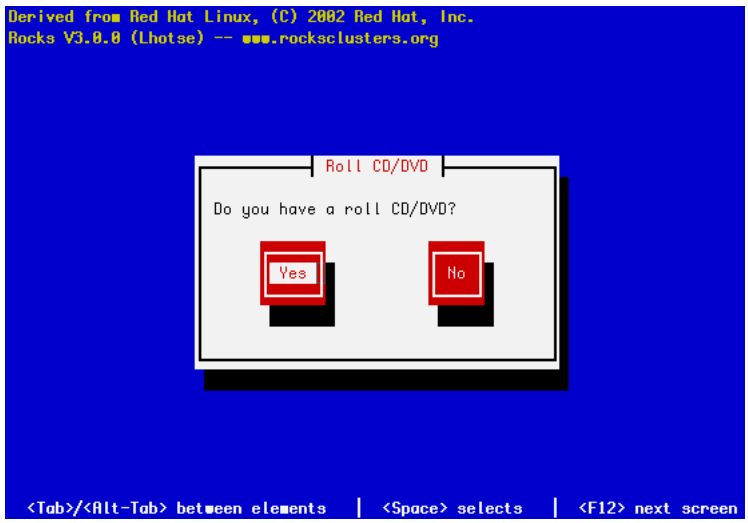
Notes

1. http://www.rocksclusters.org/rocks-documentation/4.1/customization-adding-packages.html

Chapter 2. Installing the App Trace Roll

2.1. Adding the Roll

The App Trace Roll can be installed during the frontend installation step of your cluster (refer to section 1.2 of the Rocks usersguide). If this method is chosen, then the App Trace Roll is added to a frontend installation in exactly the same manner as the required OS Roll. Specifically, after the OS Roll is added, the installer will once again ask if you have a Roll (see below). Select 'Yes' and insert the App Trace Roll.



2.1.1. Adding the Roll to a Running System

The App Trace Roll can also be added to running frontend. Here's the procedure for an x86_64 cluster (for an i386 cluster, substitute *x86_64* with *i386*).

```
# cd /tmp
# wget http://www.rocksclusters.org/ftp-site/pub/rocks/extras/4.1/x86_64/app-trace-4.1-0.x86_64.disk
# mount -o loop app-trace-4.1-0.x86_64.disk1.iso /mnt/cdrom
# cd /home/install
# rocks-dist --install copyroll
# umount /mnt/cdrom
# rocks-dist dist
# kroll app-trace > /tmp/app-trace-install.sh
# sh /tmp/app-trace-install.sh
```

Now install your compute nodes (or re-install your compute nodes if you added the roll to a running system).

Chapter 3. Using the App Trace Roll

3.1. Starting and Stopping Tracing on Compute Nodes

After you install the App Trace Roll on your frontend and compute nodes, the compute nodes will be running a SystemTap script that is ready to collect data about each read and write I/O for an application (or set of applications).

To start colleting data about the program *iozone*, execute:

```
# cluster-fork 'echo "iozone" > /proc/sys/debug/traced-apps'
```

After the above command completes, when you run *iozone* on any compute node, the SystemTap script will record its I/O activity to a log file.

To trace multiple programs, just separate them by a comma. For example, to trace both *iozone* and *dd*:

```
# cluster-fork 'echo "iozone,dd" > /proc/sys/debug/traced-apps'
```

To stop tracing, execute:

```
# cluster-fork 'echo "" > /proc/sys/debug/traced-apps'
```

3.2. Analyzing App Trace Data at Your Site

If you wish analyze the data, follow the procedure in this section. If not and you would like to upload your trace files to SDSC, skip to the next section Uploading Trace Files to SDSC.

To download all the traces from all the compute nodes, execute:

```
# make -f /opt/app-trace/collect/Makefile local
```

The above command will store all traces in */state/partition1/traces*.

Collecting the trace files stops the trace function on each node. If you wish to trace more applications after collecting the trace data, you'll need to reexecute a command like:

```
# cluster-fork 'service rocks-app-trace start'
# cluster-fork 'echo "iozone" > /proc/sys/debug/traced-apps'
```

Then to build plots for specific executables, first go to the directory that has the recently downloaded traces:

```
# cd /state/partition1/traces
```

Then, determine the directory that was created by the above collection. Below is an example listing:

```
# ls | sort
```

```
1132195141.948881000
1132195260.675707000
1132195264.174861000
```

The last directory is the most recently created one. Now, let's build some I/O trace graphs from the recently created collection:

```
# cd 1132195264.174861000
# make -f /opt/app-trace/analyze/bin/Makefile PROGRAMS=iozone,dd
```

The above will build graphs for every instance of the programs *iozone* and *dd* that were running on all the compute nodes over the trace interval.

The resulting graphs are put under */var/www/html/rocks-app-trace/analyze/*. This directory contains a subdirecties that are labeled for each node that trace data was collected from.

Point your web browser at *http://10.1.1.1/rocks-app-trace/analyze* and navigate into the node directory names. In the example above, there are the files:

```
trace-disk-iozone-8974-read-0.plot.png
trace-disk-iozone-8974-read-1.plot.png
trace-disk-iozone-8974-read-2.plot.png
trace-disk-iozone-8974-read-3.plot.png
trace-disk-iozone-8974-read-4.plot.png
trace-disk-iozone-8974-read,write,seek.plot.png
trace-disk-iozone-8974-seek-0.plot.png
trace-disk-iozone-8974-seek-1.plot.png
trace-disk-iozone-8974-seek-2.plot.png
trace-disk-iozone-8974-seek-3.plot.png
trace-disk-iozone-8974-write-0.plot.png
trace-disk-iozone-8974-write-1.plot.png
trace-disk-iozone-8974-write-2.plot.png
trace-disk-iozone-8974-write-3.plot.png
```

For traces that have many data points (e.g., traces that have over 50,000 events), the plot is broken up into subplots. In the example above, *trace-disk-iozone-8974-read-3.plot.png* is the 3rd graph for the iozone program and it plots read disk events.

An example graph that contains read, write and seek disk events would be *trace-disk-iozone-6470-read,write,seek.plot.png* and it looks like:

In addition to disk I/Os, all network reads and writes are also captured. To view the network I/O activity for a process, click on the file with the form 'trace-net-*program-pid-op-graphnumber*.plot.pdf'.

Here is an example plot from a Linpack process (xhpl):

3.3. Uploading Trace Files to SDSC

After you have collected traces for your application(s), you may wish to upload your traces files to our server at SDSC (*traces.rocksclusters.org*). Once uploaded, then we (the Rocks Group) will use these traces to help us understand the I/O characteristics of parallel applications.

Specifically, in the near term, these traces will be used to support Greg Bruno's Ph.D. research regarding parallel file systems. So in effect, you'll be helping a student to graduate. :-)

To upload the trace files from each compute node to SDSC's server, on the frontend execute:

```
# make -f /opt/app-trace/collect/Makefile sdsc
```

This invokes the program */opt/app-trace/bin/collect.sh* on each compute node which uses HTTPS POST to copy the file to SDSC's server. See the Internals Chapter for details regarding the mechanisms that are invoked when uploading trace files.

Chapter 4. Tuning the App Trace Roll

4.1. Tuning the App Trace Roll

This section discusses how to tune the App Trace Roll to fit your needs.

The App Trace Roll is tuned through replacing the XML node file *app-trace-client.xml*.

First, create a *replace-app-trace-client.xml* file:

```
# cd /home/install/site-profiles/4.1/nodes/
# cp /home/install/rocks-dist/lan/*/build/nodes/app-trace-client.xml replace-app-trace-client.xml
```

There are 4 tunable values: COLLECTHOSTLOCAL, COLLECTHOSTSDSC, TRACEDIR and FILESIZE.

- *COLLECTHOSTLOCAL* - This is the host that trace files are uploaded to when executing 'make -f /opt/app-trace/collect/Makefile local'. The default value is the private IP address of your frontend.
- *COLLECTHOSTSDSC* - This is the host that trace files are uploaded to when executing 'make -f /opt/app-trace/collect/Makefile sdsc'. The default value is the *traces.rocksclusters.org* which is a host at SDSC.
- *TRACEDIR* - This is the directory on the compute node where the trace file is stored. The default value is */state/partition1*.
- *FILESIZE* - This is the maximum size of the trace file. The default value is 10% of the partition on which TRACEDIR resides.

After modifying the above values, you can apply the values to the compute nodes by rebuilding the distribution and reinstalling the compute nodes:

```
# cd /home/install
# rocks-dist dist
# cluster-fork '/boot/kickstart/cluster-kickstart ; exit'
```

Chapter 5. Internals

5.1. SystemTap Internals

The focus of this chapter to describe the internals of the mechanisms used in the App Trace Roll.

Read and write system calls are traced using SystemTap[1]. Below is the *Tap script* used to record open, socket, read, write, lseek and close events:

```
 1  /*
 2   * $Id: internals.sgml,v 1.2 2005/11/30 21:48:40 bruno Exp $
 3   *
 4   * @Copyright@
 5   *
 6   *                               Rocks
 7   *                         www.rocksclusters.org
 8   *                           version 4.1 (fuji)
 9   *
10   * Copyright (c) 2005 The Regents of the University of California. All
11   * rights reserved.
12   *
13   * Redistribution and use in source and binary forms, with or without
14   * modification, are permitted provided that the following conditions are
15   * met:
16   *
17   * 1. Redistributions of source code must retain the above copyright
18   * notice, this list of conditions and the following disclaimer.
19   *
20   * 2. Redistributions in binary form must reproduce the above copyright
21   * notice, this list of conditions and the following disclaimer in the
22   * documentation and/or other materials provided with the distribution.
23   *
24   * 3. All advertising materials mentioning features or use of this
25   * software must display the following acknowledgement:
26   *
27   *      "This product includes software developed by the Rocks
28   *      Cluster Group at the San Diego Supercomputer Center and
29   *      its contributors."
30   *
31   * 4. Neither the name or logo of this software nor the names of its
32   * authors may be used to endorse or promote products derived from this
33   * software without specific prior written permission.  The name of the
34   * software includes the following terms, and any derivatives thereof:
35   * "Rocks", "Rocks Clusters", and "Avalanche Installer".
36   *
37   * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS “AS IS”
38   * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
39   * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
40   * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS
41   * BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
42   * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
```

```
43   * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
44   * BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
45   * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
46   * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN
47   * IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
48   *
49   * @Copyright@
50   *
51   * $Log: internals.sgml,v $
51   * Revision 1.2  2005/11/30 21:48:40  bruno
51   * touch ups
51   *
52   * Revision 1.5  2005/11/29 18:50:54  bruno
53   * needed new method to get file descriptor for i386
54   *
55   * Revision 1.3  2005/11/23 03:47:53  bruno
56   * major overhaul of the systemtap file
57   *
58   * Revision 1.1  2005/11/17 16:55:56  bruno
59   * first rev
60   *
61   *
62   */
63
64  /*
65   * this code is used to put a file into the proc file system that is used
66   * by this systemtap program to determine which programs to trace.
67   *
68   * the file is: /proc/sys/debug/traced-apps
69   *
70   * for example, to trace 'iozone', after starting this systemtap file,
71   * on the command line execute:
72   *
73   *      echo "iozone" > /proc/sys/debug/traced-apps
74   */
75
76  %{
77  #include <linux/sysctl.h>
78
79  /*
80   * a 'handle' used to deallocate the proc fs table
81   */
82  struct ctl_table_header *app_trace_proc_fs;
83
84  #define MAX_TRACED_APPS_SIZE    1024
85
86
87  char traced_apps[MAX_TRACED_APPS_SIZE] = "";
88  char working_traced_apps[MAX_TRACED_APPS_SIZE];
89
90  static ctl_table app_trace_table[] = {
91          {
92                  .ctl_name       = 1,
```

```
 93                 .procname       = "traced-apps",
 94                 .data           = &traced_apps,
 95                 .maxlen         = MAX_TRACED_APPS_SIZE,
 96                 .mode           = 0644,
 97                 .proc_handler   = &proc_dostring,
 98                 .strategy       = &sysctl_string,
 99         },
100
101         { .ctl_name = 0 }
102 };
103
104 static ctl_table app_trace_root_table[] = {
105         {
106                 .ctl_name = CTL_DEBUG,
107                 .procname = "debug",
108                 .mode = 0555,
109                 .child = app_trace_table,
110         },
111
112         { .ctl_name = 0 }
113 };
114 %}
115
116
117 function get_returnvalue:long () %{
118         if (CONTEXT->regs) {
119 #if defined (__x86_64__)
120                 THIS->__retvalue = CONTEXT->regs->rax;
121 #elif defined (__i386__)
122                 THIS->__retvalue = CONTEXT->regs->eax;
123 #endif
124         } else {
125                 THIS->__retvalue = 0;
126         }
127 %}
128
129 /*
130  * get_fd() is called from llseek which is only called from i386 programs.
131  * but, there is an __x86_64__ clause to make sure the program compiles
132  * correctly.
133  */
134 function get_fd:long (fd) %{
135         if (CONTEXT->regs) {
136 #if defined (__x86_64__)
137                 THIS->__retvalue = THIS->fd;
138 #elif defined (__i386__)
139                 THIS->__retvalue = CONTEXT->regs->ebx;
140 #endif
141         } else {
142                 THIS->__retvalue = 0;
143         }
144 %}
145
```

```
146  function is_traced_call:long (name:string) %{
147          char    *a;
148          char    *b;
149
150          strcpy(working_traced_apps, traced_apps);
151          b = working_traced_apps;
152
153          while ((a = strsep(&b, ","))) {
154                  if (!strcmp(a, THIS->name)) {
155                          THIS->__retvalue = 1;
156                          return;
157                  }
158          }
159
160          THIS->__retvalue = 0;
161  %}
162
163  function output_header(op, name, pid) {
164          print(string(gettimeofday_us()) . " op: " . op);
165          print(" execname: " . name);
166          print(" pid: " . pid);
167  }
168
169  probe kernel.function("sys_open").return
170  {
171          if (is_traced_call(execname())) {
172                  output_header("open", execname(), string(pid()));
173
174                  fd = get_returnvalue();
175                  print(" fd: " . string(fd) . "\n");
176          }
177  }
178
179  probe kernel.function("sys_socket").return
180  {
181          if (is_traced_call(execname())) {
182                  output_header("socket", execname(), string(pid()));
183
184                  fd = get_returnvalue();
185                  print(" fd: " . string(fd) . "\n");
186          }
187  }
188
189  probe kernel.function("sys_read")
190  {
191          if (is_traced_call(execname())) {
192                  output_header("read", execname(), string(pid()));
193
194                  fd = get_fd($fd);
195                  print(" fd: " . string(fd));
196                  print(" count: " . string($count) .  "\n");
197          }
198  }
```

```
199
200 probe kernel.function("sys_write")
201 {
202         if (is_traced_call(execname())) {
203                 output_header("write", execname(), string(pid()));
204
205                 fd = get_fd($fd);
206                 print(" fd: " . string(fd));
207                 print(" count: " . string($count) . "\n");
208         }
209 }
210
211 probe kernel.function("sys_lseek")
212 {
213         if (is_traced_call(execname())) {
214                 output_header("seek", execname(), string(pid()));
215
216                 fd = get_fd($fd);
217                 print(" fd: " . string(fd));
218         }
219 }
220
221 /*
222  * llseek is only called on i386
223  */
224 probe kernel.function("sys_llseek")
225 {
226         if (is_traced_call(execname())) {
227                 output_header("seek", execname(), string(pid()));
228
229                 fd = get_fd($fd);
230                 print(" fd: " . string(fd));
231         }
232 }
233
234 probe kernel.function("vfs_llseek").return
235 {
236         if (is_traced_call(execname())) {
237                 offset = get_returnvalue();
238                 print(" offset: " . string(offset) . "\n");
239         }
240 }
241
242 probe kernel.function("sys_close")
243 {
244         if (is_traced_call(execname())) {
245                 output_header("close", execname(), string(pid()));
246
247                 fd = get_fd($fd);
248                 print(" fd: " . string(fd) . "\n");
249         }
250 }
251
```

```
252
253  function init_proc_fs() %{
254          app_trace_proc_fs = register_sysctl_table(app_trace_root_table, 1);
255  %}
256
257  function deinit_proc_fs() %{
258          unregister_sysctl_table(app_trace_proc_fs);
259  %}
260
261  probe begin
262  {
263          log("trace-io: starting probe");
264          init_proc_fs();
265  }
266
267  probe end
268  {
269          log("trace-io: ending probe");
270          deinit_proc_fs();
271  }
272
```

Notice in *probe kernel.function("sys_read")* (line 189) and *probe kernel.function("sys_write")* (line 200) only the following information is recorded:

- Time the event occurred
- Name of the program that is issuing the system call
- Process Id for the program
- File descriptor the system call will be performed upon
- I/O size (or the absolute offset in the case of seek)


Note that no user data is ever examined.

Here is a snippet from a trace when running the I/O benchmark *iozone*:

```
1133384755854320 op: close execname: iozone pid: 3693 fd: 3
1133384755854365 op: write execname: iozone pid: 3693 fd: 1 count: 8
1133384755854513 op: write execname: iozone pid: 3693 fd: 1 count: 8
1133384755854629 op: open execname: iozone pid: 3693 fd: 3
1133384755854672 op: read execname: iozone pid: 3693 fd: 3 count: 4096
1133384755854710 op: seek execname: iozone pid: 3693 fd: 3 offset: 0
```

The Tap script above is loaded with the init script */etc/rc.d/init.d/rocks-app-trace*. Below is the line that starts the tracing:

```
ulimit -f $FILESIZE && /usr/bin/stap \
        -g /opt/app-trace/systemtap/trace-io.stp > \
                $TRACEDIR/‘hostname‘-app-trace.out 2>&1 &
```

5.2. Upload Internals

When a user executes the file upload procedure, the trace file from the compute nodes are copied to a server at SDSC. The user executes *make -C /opt/app-trace/collect/Makefile sdsc* and the Makefile has the contents:

```
TRACEID = $(shell date +%s.%N)

local:
        cluster-fork ’/opt/app-trace/bin/collect.sh $(TRACEID) local’

sdsc:
        cluster-fork ’/opt/app-trace/bin/collect.sh $(TRACEID) sdsc’
```

The *TRACEID* is the unique value that will be used to create a directory on SDSC’s server to hold all the trace files. It is important to note that this value is just the date (to the nanosecond) and doesn’t contain any information about the cluster (e.g., there is no IP or MAC address information in *TRACEID*).

In the Makefile above, file upload is carried out by a script on the compute node called */opt/app-trace/bin/collect.sh*. Note that the *TRACEID* is passed to the next stage in the *--id* parameter.

```
 1  #!/bin/bash
 2  #
 3  # $Id: internals.sgml,v 1.2 2005/11/30 21:48:40 bruno Exp $
 4  #
 5  # @Copyright@
 6  #
 7  #                               Rocks
 8  #                         www.rocksclusters.org
 9  #                           version 4.1 (fuji)
10  #
11  # Copyright (c) 2005 The Regents of the University of California. All
12  # rights reserved.
13  #
14  # Redistribution and use in source and binary forms, with or without
15  # modification, are permitted provided that the following conditions are
16  # met:
17  #
18  # 1. Redistributions of source code must retain the above copyright
19  # notice, this list of conditions and the following disclaimer.
20  #
21  # 2. Redistributions in binary form must reproduce the above copyright
22  # notice, this list of conditions and the following disclaimer in the
23  # documentation and/or other materials provided with the distribution.
```

```
24  #
25  # 3. All advertising materials mentioning features or use of this
26  # software must display the following acknowledgement:
27  #
28  #       "This product includes software developed by the Rocks
29  #       Cluster Group at the San Diego Supercomputer Center and
30  #       its contributors."
31  #
32  # 4. Neither the name or logo of this software nor the names of its
33  # authors may be used to endorse or promote products derived from this
34  # software without specific prior written permission.  The name of the
35  # software includes the following terms, and any derivatives thereof:
36  # "Rocks", "Rocks Clusters", and "Avalanche Installer".
37  #
38  # THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS “AS IS”
39  # AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
40  # THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
41  # PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS
42  # BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
43  # CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
44  # SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
45  # BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
46  # WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
47  # OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN
48  # IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
49  #
50  # @Copyright@
51  #
52  # $Log: internals.sgml,v $
52  # Revision 1.2  2005/11/30 21:48:40  bruno
52  # touch ups
52  #
53  # Revision 1.3  2005/11/30 01:41:35  bruno
54  # a bit more
55  #
56  # Revision 1.2  2005/11/18 00:54:54  bruno
57  # only upload app trace files
58  #
59  # Revision 1.1  2005/11/17 16:55:56  bruno
60  # first rev
61  #
62  #
63  #
64
65  . /opt/app-trace/conf/variables
66
67  echo "" > /proc/sys/debug/traced-apps
68
69  for i in ‘find $TRACEDIR -type f -name "*app-trace.out*"‘
70  do
71          file $i | grep ’bzip2 compressed’ > /dev/null 2>&1
72          notcompressed=$?
73
```

```
74          if [ $notcompressed == 1 ]
75          then
76                  bzip2 --force $i
77                  filename=$i.bz2
78          else
79                  filename=$i
80          fi
81
82          if [ $2 == 'local' ]
83          then
84                  CHOST=$COLLECTHOSTLOCAL
85          else
86                  CHOST=$COLLECTHOSTSDSC
87          fi
88
89          /opt/app-trace/bin/upload.py \
90                  --upload-server=$CHOST --filename=$filename --id=$1
91  done
92
```

For each file in TRACEDIR (line 69), it compresses the file (if not already compressed), then calls */opt/app-trace/bin/upload.py* which uses HTTPS POST to copy the compressed file to your frontend (when executing *make -f /opt/app-trace/collect/Makefile local*) or to the the SDSC server (when executing *make -f /opt/app-trace/collect/Makefile sdsc*).

Here is the contents of */opt/app-trace/bin/upload.py*:

```
 1  #!/opt/rocks/usr/bin/python
 2  #
 3  # $Id: internals.sgml,v 1.2 2005/11/30 21:48:40 bruno Exp $
 4  #
 5  # @Copyright@
 6  #
 7  #                               Rocks
 8  #                         www.rocksclusters.org
 9  #                           version 4.1 (fuji)
10  #
11  # Copyright (c) 2005 The Regents of the University of California. All
12  # rights reserved.
13  #
14  # Redistribution and use in source and binary forms, with or without
15  # modification, are permitted provided that the following conditions are
16  # met:
17  #
18  # 1. Redistributions of source code must retain the above copyright
19  # notice, this list of conditions and the following disclaimer.
20  #
21  # 2. Redistributions in binary form must reproduce the above copyright
22  # notice, this list of conditions and the following disclaimer in the
23  # documentation and/or other materials provided with the distribution.
24  #
25  # 3. All advertising materials mentioning features or use of this
26  # software must display the following acknowledgement:
```

```
27  #
28  #       "This product includes software developed by the Rocks
29  #       Cluster Group at the San Diego Supercomputer Center and
30  #       its contributors."
31  #
32  # 4. Neither the name or logo of this software nor the names of its
33  # authors may be used to endorse or promote products derived from this
34  # software without specific prior written permission.  The name of the
35  # software includes the following terms, and any derivatives thereof:
36  # "Rocks", "Rocks Clusters", and "Avalanche Installer".
37  #
38  # THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS “AS IS”
39  # AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
40  # THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
41  # PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS
42  # BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
43  # CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
44  # SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
45  # BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
46  # WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
47  # OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN
48  # IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
49  #
50  # @Copyright@
51  #
52  # $Log: internals.sgml,v $
52  # Revision 1.2  2005/11/30 21:48:40  bruno
52  # touch ups
52  #
53  # Revision 1.1  2005/11/17 16:55:56  bruno
54  # first rev
55  #
56  #
57
58  import getopt
59  import sys
60  import httplib
61  import os
62  import os.path
63
64  boundary = '------------Rocks-Trace-File-Upload------------'
65
66
67  def file_header(filename):
68          str = '--' + boundary + '\n'
69          str += 'Content-Disposition: form-data; '
70          str += 'name="filename"; '
71          str += 'filename="%s"\n\n' % (os.path.basename(filename))
72          return str
73
74
75  def file_trailer():
76          str = '\n--' + boundary + '\n\n\r\n'
```

```
 77         return str
 78
 79
 80 def encode_fields(fields):
 81         str = ”
 82
 83         for (key, value) in fields:
 84                 str += ’--’ + boundary + ’\n’
 85                 str += ’Content-Disposition: form-data; ’ + \
 86                                                 ’name="%s"\n\n’ % (key)
 87                 str += value + ’\n’
 88
 89         return str
 90
 91
 92
 93 def send_file(host, req, fields, filename):
 94         str_fields = encode_fields(fields)
 95         header = file_header(filename)
 96         trailer = file_trailer()
 97         filesize = os.path.getsize(filename)
 98
 99         h = httplib.HTTPSConnection(host)
100
101         h.putrequest(’POST’, req)
102
103         h.putheader(’content-type’,
104                         ’multipart/form-data; boundary=%s’ % (boundary))
105
106         bodysize = len(str_fields) + len(header) + len(trailer) + filesize
107         h.putheader(’content-length’, ’%d’ % (bodysize))
108
109         h.endheaders()
110
111         #
112         # send the body of the message
113         #
114         h.send(str_fields)
115         h.send(header)
116
117         file = open(filename, ’r’)
118         line = file.readline()
119         while line:
120                 h.send(line)
121                 line = file.readline()
122         file.close()
123
124         h.send(trailer)
125
126         response = h.getresponse()
127
128         return response
129
```

```
130  #
131  # main
132  #
133
134  #
135  # get the command line arguments
136  #
137  opts, args = getopt.getopt(sys.argv[1:], '', ['upload-server=',
138          'filename=', 'id=' ])
139
140  #
141  # IP address of the node we care about
142  #
143  upload_server = ''
144  filename = ''
145  id = ''
146
147  for c in opts:
148          if c[0] == '--upload-server':
149                  upload_server = c[1]
150          if c[0] == '--filename':
151                  filename = c[1]
152          if c[0] == '--id':
153                  id = c[1]
154
155
156  req = '/traces/sbin/store-trace.cgi'
157  fields = [ ('username', 'rocks'), ('id', id) ]
158
159  send_file(upload_server, req, fields, filename)
```

Line 99 shows the HTTPS connection and line 101 shows the POST command.

It's important to note that all trace files are transferred using a secure link. That is, if an outside program is eavesdropping on the connection, they will only see ciphertext and not cleartext.

The program */opt/app-trace/bin/upload.py* calls the CGI program */traces/sbin/store-trace.cgi* on SDSC's server. The contents of the CGI is:

```
 1  #!/opt/rocks/usr/bin/python
 2
 3  import cgi
 4  import os
 5
 6  tracedir = '/state/partition1/traces/'
 7
 8
 9  def savefile(infile, outfilename):
10          outfile = open('%s' % (outfilename), 'w+')
11
12          if infile:
```

```
13                  line = infile.readline()
14                  while line:
15                          outfile.write(line)
16                          line = infile.readline()
17          else:
18                  outfile.write(f.value)
19
20          outfile.close()
21          return
22
23  #
24  # main
25  #
26  form = cgi.FieldStorage()
27
28  username = ”
29  filename = ”
30  id = ”
31
32  file = open(’/tmp/store-file.cgi’, ’w+’)
33  for key in form.keys():
34
35          if key == ’username’:
36                  username = form.getvalue(key)
37                  file.write(’username (%s)\n’ % (username))
38                  file.write(’\n’)
39          elif key == ’id’:
40                  id = form.getvalue(key)
41                  file.write(’id (%s)\n’ % (id))
42                  file.write(’\n’)
43          elif key == ’filename’ and form[key].filename:
44                  filename = form[key].filename
45
46  file.close()
47
48  if username == ’rocks’ and id != ” and filename != ”:
49          dirname = os.path.join(’/state/partition1/traces/’, id)
50          if not os.path.exists(dirname):
51                  os.system(’mkdir -p %s’ % (dirname))
52
53          outfilename = os.path.join(dirname, filename)
54
55          savefile(form[’filename’].file, outfilename)
56
57  print ’Content-type: application/octet-stream’
58  print ’Content-length: %d’ % (len(”))
59  print ”
60  print ”
```

The CGI above creates a new directory based on the *id* passed to it by *upload.py*.


Again, no cluster-specific information is stored on the SDSC server.

Notes

1. http://sourceware.org/systemtap/

Chapter 6. Copyrights

6.1. App Trace