Seminar: Leistungsanalyse unter Linux

Performance analysis using GPerfTools and LTTng

Heye Vöcking

March 27, 2012
Contents

• Great Performance Tools
  – Overview and History
  – Tools offered for profiling and analysis

• Linux Trace Toolkit next generation
  – Overview and History
  – Tracing using Tracepoints
  – Analysis

• Live demonstration
Profiling and Tracing

- What?
  - Collecting information during execution

- Why?
  - Understanding software for debugging / tuning (esp. on multicore systems)

- How?
  - “Great Performance Tools” and “Linux Trace Toolkit next generation”
Great Performance Tools

- Distributed under BSD license
- Community run
- Libraries are available for Windows, Linux, Solaris, and Mac
- Supports C/C++ and all languages that can call C code
GPerfTools – History

- Originally called “Google Performance Tools”
- Started in March 2005
- Version 1.0 early 2009
- Version 2.0 since February 2012
GPerfTools – History

- Main contributor “csilvers” stepped down since v. 2.0
- Google withdrew ownership → now completely community run
- Renaming to “gperftools”, where “g” stands for “great”
- Main developer from now on is David Chappelle
GPerfTools – Overview

- CPU Profiler – performance of functions
- TCMalloc – fast, thread aware malloc
- Heap Leak Checker – memory leak detector
- Heap Profiler – record program stack
GPerfTools – CPU Profiler

- To use the CPU Profiler you have to
  - Link the library with -lprofiler
  - Set $CPUPROFILE to the path where to save the profile
  - Surround the code to be profiled with
    - ProfilerStart("profile name")
    - ProfilerStop()

- Output can be analyzed with pprof (we'll get back to pprof later)
GPerfTools – TCMalloc

- To use TCMalloc simply link the library with -ltcmalloc
- Faster than clib malloc
- Low overhead on small objects
- Reduces thread lock contention in multithreaded environments
- The Heap Leak Checker and Heap Profiler work with TCMalloc
GPerfTools – Heap Leak Checker

• To use the Heap Leak Checker you have to
  – Link the library with `-ltcmalloc`
  – Set `$HEAPCHECK` to the desired mode
    • minimal, normal, strict, draconian, as-is, local
  • Output can be analyzed with `pprof` (we'll get back to `pprof` later)
GPerfTools – Heap Profiler

• To use the Heap Profiler you have to

  – Link the library with `-ltcmalloc`

  – Set `$HEAPPROFILE` to the path where to save the profile

  – Surround the code to be profiled with
    • `HeapProfilerStart("prefix name")`
    • `HeapProfilerStop()`

• Output can be analyzed with `pprof` (we'll get back to `pprof` later)
GPerfTools – Profiler Insights

- Function of GetStackFrames():

```c
main() { foo(); }
foo() { bar(); }
bar() {
    void* result[10];
    int sizes[10];
    int depth = GetStackFrames(result, sizes, 10, 1); }
```

- Result:
  - depth = 2
  - result[0] = foo, sizes[0] = 16
GPerfTools – pprof

• Analyzes profiles generated by
  
  – CPU Profiler – Weighted call graph with timing information
  
  – Heap Leak Checker – call graph of reported leaks
  
  – Heap Profiler – weighed directed graph of memory allocations

• Text, postscript (gv), dot, pdf, gif, source-code listings, & disassembly
Linux Trace Toolkit next generation

- Distributed under GPLv2
- Community run
- Offers Kernel and User Space Tracing
- Only available for Linux (32 & 64 bit)
- Supports C/C++ and all languages that can call C code
LTtng – History

- Successor of Linux Trace Toolkit
- Launched in 2005
- Version 2.0 since March 2012
- Written and maintained by Mathieu Desnoyers
LTTng – Overview

- Consists of 3 parts:
  - Kernel part → Kernel tracing
  - User space command-line application (lttcl)
  - User space daemon (lttd)
- Modular: 5 modules
  - ltt-core, generates events, controls:
    - ltt-heartbeat, ltt-facilities, ltt-statedump
  - ltt-base, built in kernel object, keeps symbols & data structures
LTTng – Tracing

- Observes operating system kernel events such as:
  - system calls, interrupt requests, scheduling & network activities

- Multiple traces can be recorded simultaneously (on mult. CPUs)

- Events are recorded by so called “Tracepoints”
LTTng – Tracepoints

- Tracepoints are small pieces of code, like this:

  ```c
  if (tracepoint_1_active)
  (*tracepoint_1_probe)(arg1, arg2);
  ```

- Almost no overhead when inactive

- Overhead when active comparable to a C function call
LTTng – Tracepoints

Kernel

Core (instrumented code)

Modules (instrumented code)

Probe

reads

calls

calls

writes (synchronized)

Trace Session

Channels

* one trace session contains multiple channels
LTTng – Storage & Tracemodes

• Data can be written to disk or sent to a network stream

• Available tracemodes are:
  – Write-to-disk, flight recorder
  – Still under development: stream to remote disk & live monitoring
# LTTng – Overhead

<table>
<thead>
<tr>
<th>Load size</th>
<th>Test Series</th>
<th>CPU time (%)</th>
<th>Data rate (MiB/s)</th>
<th>Events/s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>load</td>
<td>probes</td>
<td>lttd</td>
</tr>
<tr>
<td>Small</td>
<td>mozilla (browsing)</td>
<td>1.15</td>
<td>0.053</td>
<td>0.27</td>
</tr>
<tr>
<td>Medium</td>
<td>find</td>
<td>15.38</td>
<td>1.150</td>
<td>0.39</td>
</tr>
<tr>
<td>High</td>
<td>find + gcc</td>
<td>63.79</td>
<td>1.720</td>
<td>0.56</td>
</tr>
<tr>
<td>Very high</td>
<td>find + gcc + ping flood</td>
<td>98.60</td>
<td>8.500</td>
<td>0.96</td>
</tr>
</tbody>
</table>

*Source: Desnoyers: A low impact performance and behavior monitor for GNU/Linux.*
LTTng – Post-Processing

- Babeltrace
  - Text based, LTTng 2.0 traces in CTF (Common Trace Format)

- LTT Viewer
  - Visual analyzer, written in C, extendable with plug-ins

- Eclipse
  - TMF (Trace Monitoring Framework) plug-in, displays:
    - Control Flow, Resources, and Statistics
Areas of application

• GPerfTools
  – Google, ...

• LTTng
  – IBM, Siemens, Autodesk, Ericsson, ...
  – Included in packages of Montavista, Wind River, STLlinux, & Suse
Summary

• GPerfTools:
  – CPU Profiler, TCMalloc, Heap Leak Checker, Heap Profiler
  – Pprof for analysis

• LTTng:
  – Kernel & user space tracer → tracepoint → probe → channel → I/O
  – Babeltrace, LTT Viewer, and eclipse for analysis
Sources

- **GperfTools**: [http://gperftools.googlecode.com](http://gperftools.googlecode.com)

- **Desnoyers**:
  - *Low-Impact Operating System Tracing*
  - *A low impact performance and behavior monitor for GNU/Linux*

- **LTTng**: [http://www.lttng.org](http://www.lttng.org)

- **Chakraborty, Anjoy**: *Efficiency of LTTng as a Kernel and Userspace Tracer on Multicore Environment*