Beyond kernel tracing

Pierre-Marc Fournier

ÉCOLE
POLYTECHNIQUE
MONTRÉAL

LTTng Tracers

- Kernel tracer
  - Record events in the kernel.
  - Project started in 2005.

- Userspace tracer
  - Port to userspace of kernel tracer.
  - Project started in 2009.
  - Preceded by several generations of experimental userspace tracers.

Why userspace tracing?

- The kernel knows about: scheduling, blocking, IRQs, softIRQs, traps, system calls, signals.
- But it's not enough.

- Essential for application that:
  - Talk directly to hardware
  - Handle lots of sessions
  - Are complex in general

Objectives

- Lowest performance impact possible.
- Optimized for huge event rates and traces.
- Low intrusiveness.
- Maximize traceability.

Existing userspace tracers

- DTrace, SystemTap
- Historical LTTng userspace tracing methods
- Ftrace
- “printf”

- **Need for better performance.**

Performance

- No system call, no trap.
- Minimize impact on instruction cache.
- Minimize impact on data cache.
- Never copy events.
- Markers / tracepoints.

Scalability

- Use of userspace RCU.
- Non-blocking atomic operations.
- Per-CPU buffers.

Trace format

- Binary.
- Saved in system byte order.
- Compact.
- Same format as LTTng kernel tracer.

Combined Tracing

- Tracing both the kernel and userspace at the same time.
  - Using a kernel tracer.
  - And using a userspace tracer.

- This provides a full tracing solution that produces events covering the whole system.

- Traces are merged at analysis time.

Maximize traceability

- Early tracing.
    - Early activation of instrumentation.
- Crash recovery.
- Support multi-threaded applications.
- Signal safe.
- Follow forks.
- Do not require initialization. (URCU BP)
- Trace executable, dynamically linked libs, even dynamically loaded

Time

- Timestamp must be accurate and fast to read.
- Combined tracing: must be synchronized with kernel time source.
- x86: using cycle counter (TSC)
- Fallback: gettimeofday()
- Need userspace trace clock API

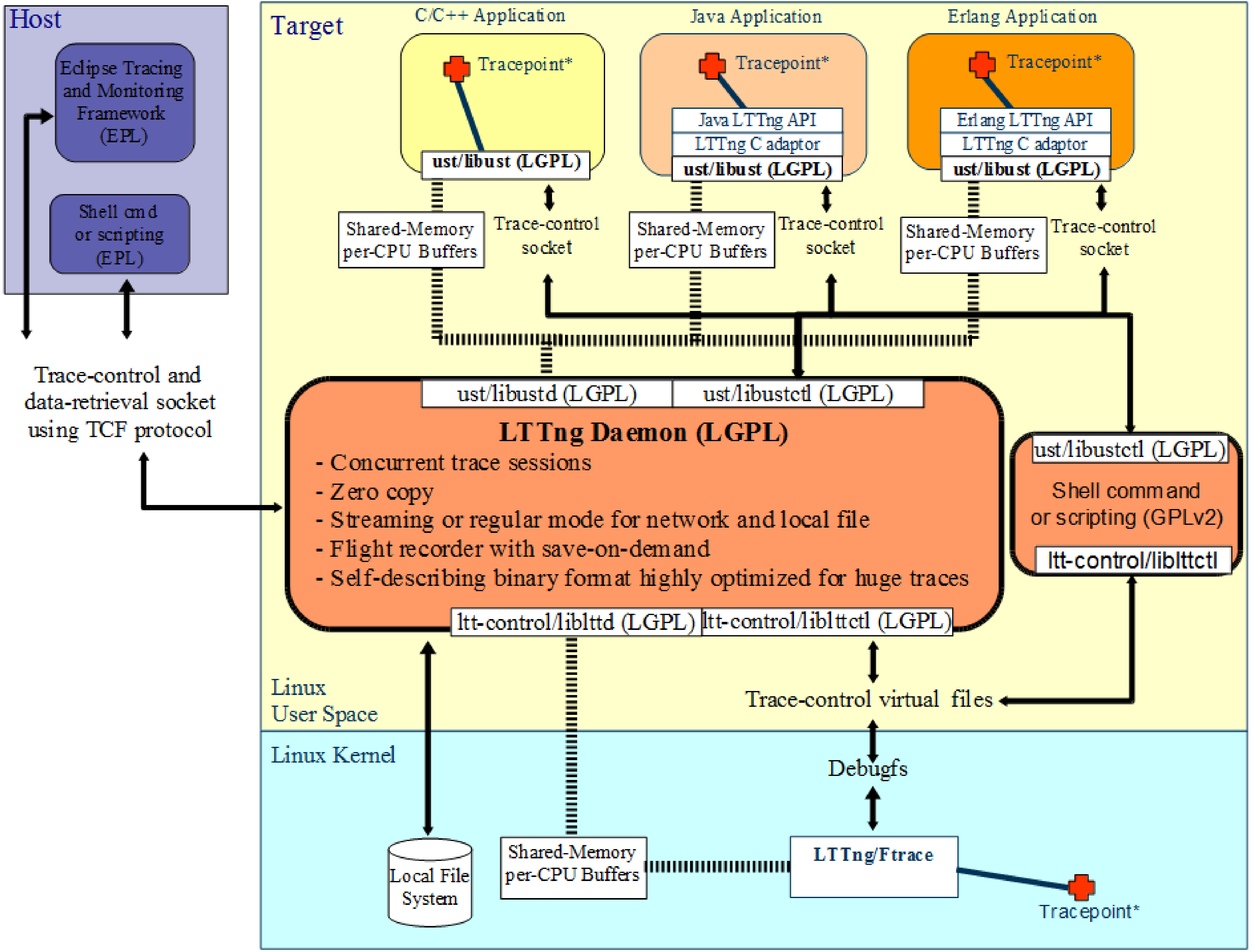
Language support

- Native API in C (markers, tracepoints)
- Trivial port to any language that can call C code.

- Special support needed for
  - Passing fundamental types in payload.
  - Efficient individual activation/deactivation.

System awareness

- Quotas
- Permissions
- Multi-process sessions



Host
Eclipse Tracing and Monitoring Framework (EPL)
Shell cmd or scripting (EPL)
Trace-control and data-retrieval socket using TCF protocol
Target
C/C++ Application
Java Application
Erlang Application
Tracepoint*
Tracepoint*
Tracepoint*
ust/libust (LGPL)
Java LTTng API
LTTng C adaptor
ust/libust (LGPL)
Erlang LTTng API
LTTng C adaptor
ust/libust (LGPL)
Shared-Memory per-CPU Buffers
Trace-control socket
Shared-Memory per-CPU Buffers
Trace-control socket
Shared-Memory per-CPU Buffers
Trace-control socket
ust/libustd (LGPL)
ust/libustctl (LGPL)
LTTng Daemon (LGPL)
- Concurrent trace sessions
- Zero copy
- Streaming or regular mode for network and local file
- Flight recorder with save-on-demand
- Self-describing binary format highly optimized for huge traces
ltt-control/liblttd (LGPL)
ltt-control/liblttctl (LGPL)
ust/libustctl (LGPL)
Shell command or scripting (GPLv2)
ltt-control/liblttctl
Linux User Space
Trace-control virtual files
Linux Kernel
Debugfs
Local File System
Shared-Memory per-CPU Buffers
LTTng/Ftrace
Tracepoint*

Performance

- Early performance measurements.
  - Dtrace: 5 us / event
  - UST: 698 ns / event
- Not yet in optimization phase.

Status

- UST 0.1 to be released soon.
- Working with Suse Linux for integration.
- GDB static tracepoints integration in progress.
- Git for userspace tracer: **http://git.dorsal.polymtl.ca/?p=ust.git**