Request Tracking in DROPS

(Diplomverteidigung)

Björn Döbel
<doebel@os.inf.tu-dresden.de>

23.06.2006

Contents

1. Introduction
2. Instrumentation
3. Applications
4. Evaluation
5. Conclusion and Outlook

Motivation

Donald Knuth:

"Premature optimization is the root of all evil (or at least most of it) in programming."

Motivation (2)

- Performance optimizations mostly inside the kernel

Motivation (2)

- Performance optimizations mostly inside the kernel
- What about the components on top of Fiasco?

Motivation (2)

- Performance optimizations mostly inside the kernel
- What about the components on top of Fiasco?
- Enable evaluation and verification of
  - (inter-)component behavior
  - (inter-)component resource usage
  - (inter-)component performance
- Use offline monitoring for evaluation
- Use online monitoring for verification

Definitions

Event

An event contains data representing an action that happens at a certain point in time.

Definitions

Event

An event contains data representing an action that happens at a certain point in time.

Sensor

A sensor is a means to produce events, mostly some piece of software.

Definitions

Event

An event contains data representing an action that happens at a certain point in time.

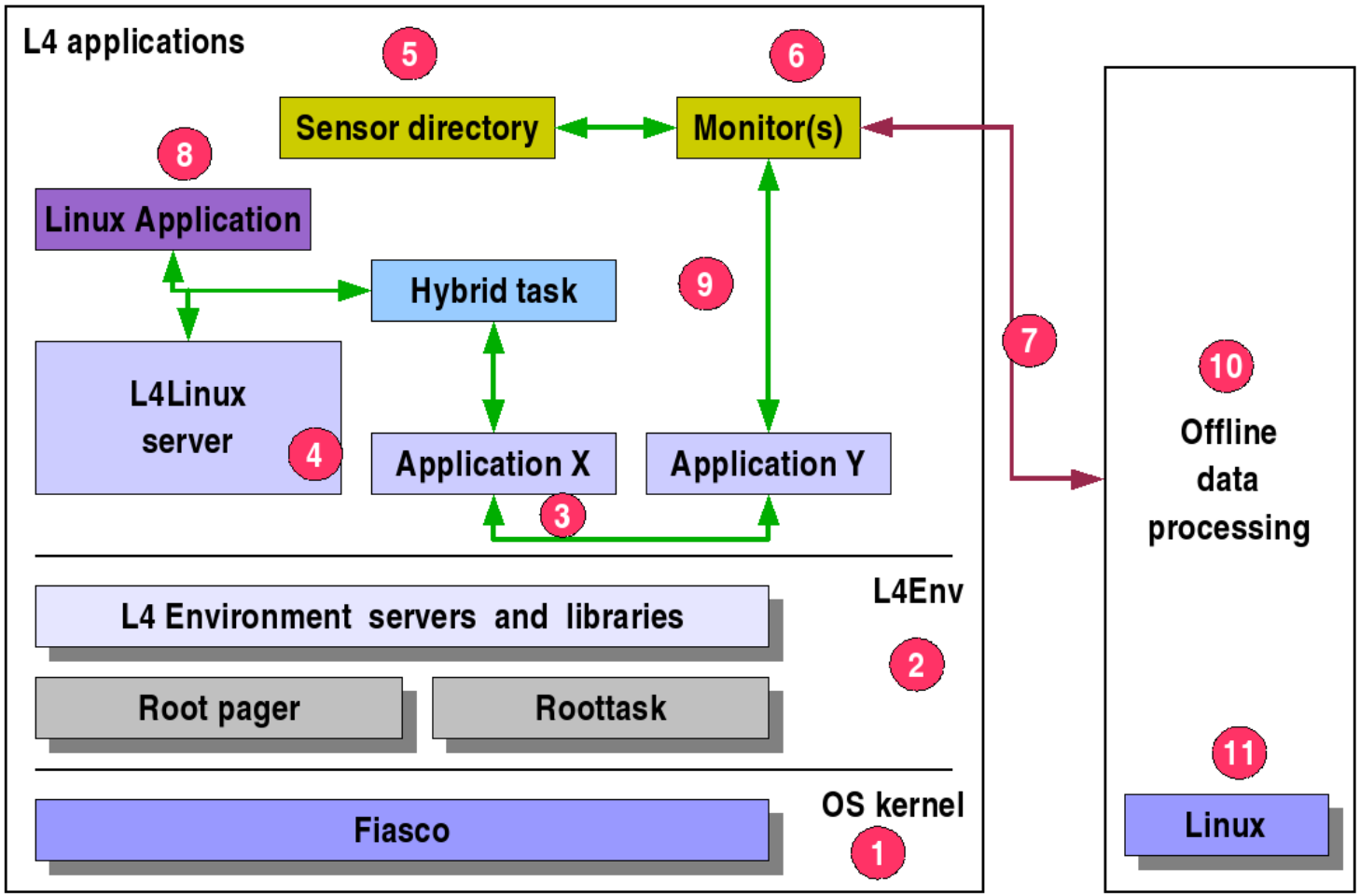
Sensor

A sensor is a means to produce events, mostly some piece of software.

Request

A request is a structured set of events. Requests are interesting with respect to resource usage.

Where to instrument?



What do we have?

- Fiasco tracebuffer
  - needed for kernel events
  - inflexible

What do we have?

- Fiasco tracebuffer
  - needed for kernel events
  - inflexible
- Ferret monitoring framework
  - sensor directory
  - predefined sensors
    - scalars
    - histograms
    - lists

Instrumentation rules

1. Deterministic event layout

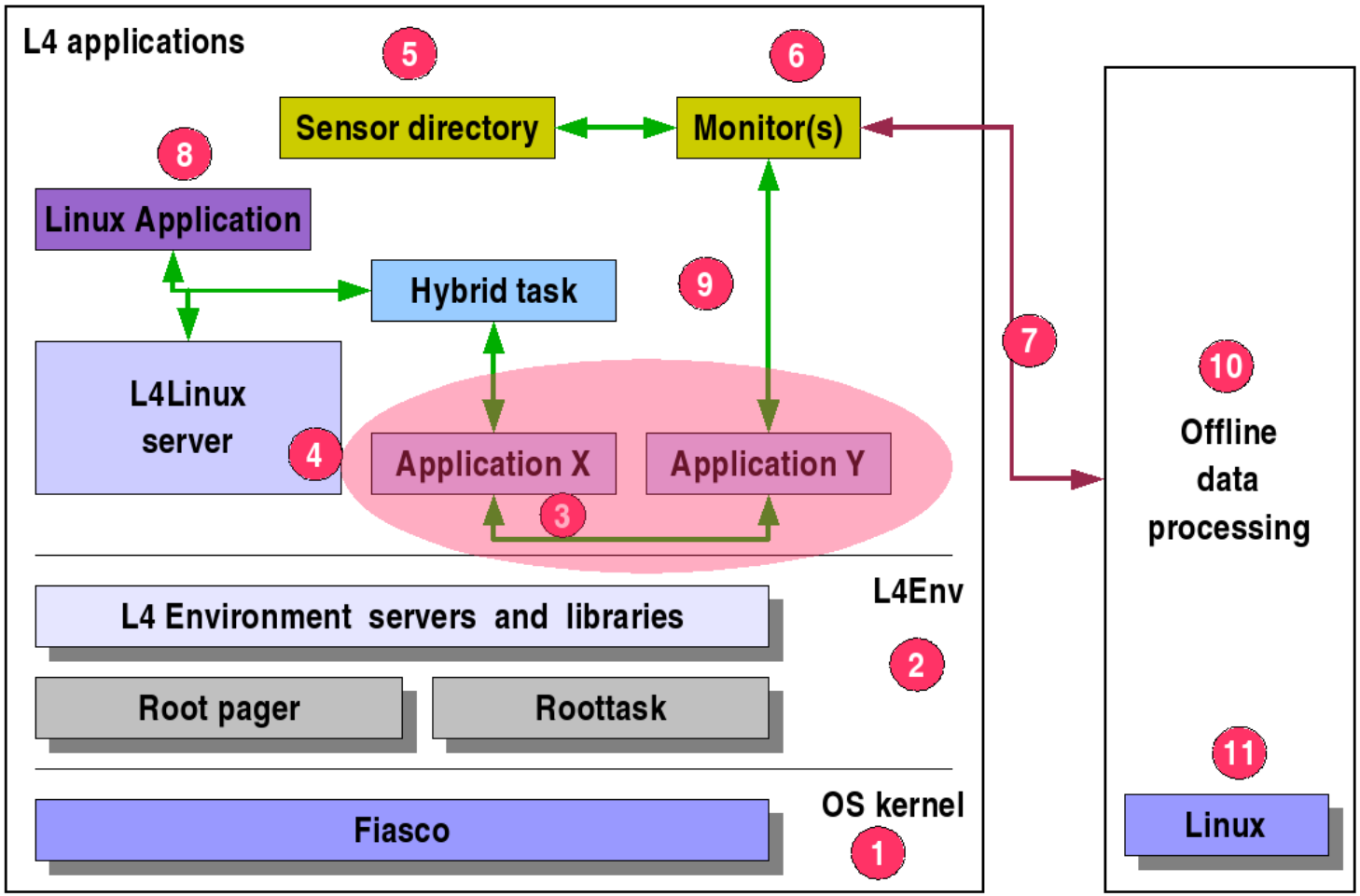
Instrumentation rules

1. Deterministic event layout
2. Use large events if possible
   - less calls to monitoring framework

Instrumentation rules

1. Deterministic event layout
2. Use large events if possible
   - less calls to monitoring framework
3. Keep out monitoring overhead
   - need to extract overhead from measurements
   - might be necessary to obtain more data

IPC tracing


L4 applications
5
6
Sensor directory
Monitor(s)
8
Linux Application
Hybrid task
9
7
L4Linux
server
4
Application X
Application Y
3
10
Offline
data
processing
L4Env
L4 Environment servers and libraries
2
Root pager
Roottask
11
OS kernel
Fiasco
1
Linux

IPC tracing

- Automated instrumentation of IPC using a Dice tracing plugin: `dice_trace`
  - covers most L4 applications
- Sensor support library
- Global list sensor for Dice events
  - initialized during application startup

IPC tracing (2)

- Client-side: one event for each call

IPC tracing (2)

- Client-side: one event for each call
- Server-side: one event for each client-side call
  - tricky: `allow_reply_only`
    - store pending events at server
    - efficient lookup

IPC tracing (2)

- Client-side: one event for each call
- Server-side: one event for each client-side call
  - tricky: `allow_reply_only`
    - store pending events at server
    - efficient lookup
  - alternative: split server-side event into multiple ones
    - multiple times of monitoring overhead

IPC tracing: event layout

<table>
<tr><td colspan="2">Timestamp
Major number
Minor number
Instance number
CPU ID</td><td>Event head</td></tr>
<tr><td>Producer ID</td><td>Producer ID</td><td rowspan="7">Additional data</td></tr>
<tr><td>Start timestamp</td><td>Start timestamp</td></tr>
<tr><td>Stop timestamp</td><td>Stop timestamp</td></tr>
<tr><td>Send size</td><td rowspan="2">Component start</td></tr>
<tr><td>Receive size</td></tr>
<tr><td>Dice opcode</td><td rowspan="2">Component stop</td></tr>
<tr><td>-</td></tr>
<tr><td>Client-side</td><td>Server-side</td><td></td></tr>
</table>

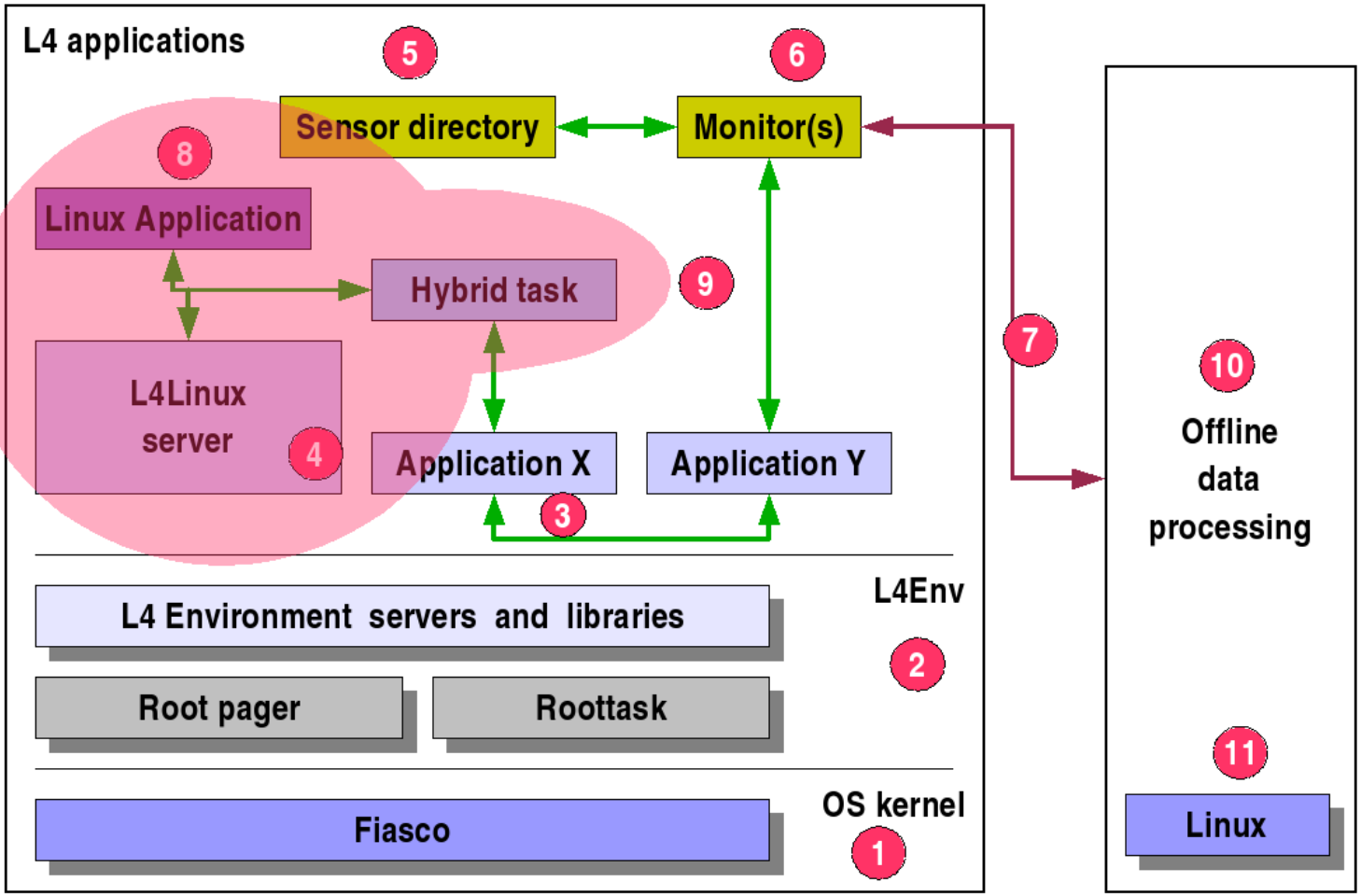
L4Env tracing

- L4Env services used by nearly all applications
- Built-in instrumentation can help with profiling and debugging
- Global L4Env sensor

L4Env tracing

- L4Env services used by nearly all applications
- Built-in instrumentation can help with profiling and debugging
- Global L4Env sensor
- Example: semaphores
  1. Critical sections:
     - How much contention?
     - How long are critical sections?
  2. Consumer–producer scenarios:
     - How long are consumers/producers waiting for a semaphore?

Tracing L4Linux



Tracing the L4Linux kernel

- Several approaches to Linux kernel tracing:
  - `printk()`
    - manual instrumentation
    - flexible

Tracing the L4Linux kernel

- Several approaches to Linux kernel tracing:
  - `printk()`
    - manual instrumentation
    - flexible
  - LTT / LTTng
    - fixed amount of predefined events
    - visualization backend

Tracing the L4Linux kernel

- Several approaches to Linux kernel tracing:
  - `printk()`
    - manual instrumentation
    - flexible
  - LTT / LTTng
    - fixed amount of predefined events
    - visualization backend
  - kProbes
    - instrument arbitrary locations inside the kernel by patching instructions with `int3`
    - flexible

Tracing the L4Linux kernel (2)

- Adam Lackorzynski ported kProbes to L4Linux:
  - cannot use `int3` as break instruction
  - use `hlt` instead
- Ferret sensor inside the L4Linux kernel
- kProbes overhead higher than in native Linux, but constant
- Differential analysis: run the same kProbe in L4Linux and Linux
  - `FerretLX` kernel module for Linux

Tracing L4Linux processes

- No user-space kProbes exist yet
- Manual instrumentation possible:
  - map a Ferret sensor into task's memory
    1. direct request from sensor directory
    2. map to a fixed area in task's address space
    3. use L4Linux as a proxy for Ferret sensors
  - 2) implemented by Martin Pohlack
- Need to modify source code

Inspecting fork/vfork in (L4)Linux

- kProbes at certain locations inside `do_fork()` routine:
  - call to `alloc_pid()`
  - call to `wake_up_new_task()`
  - function return

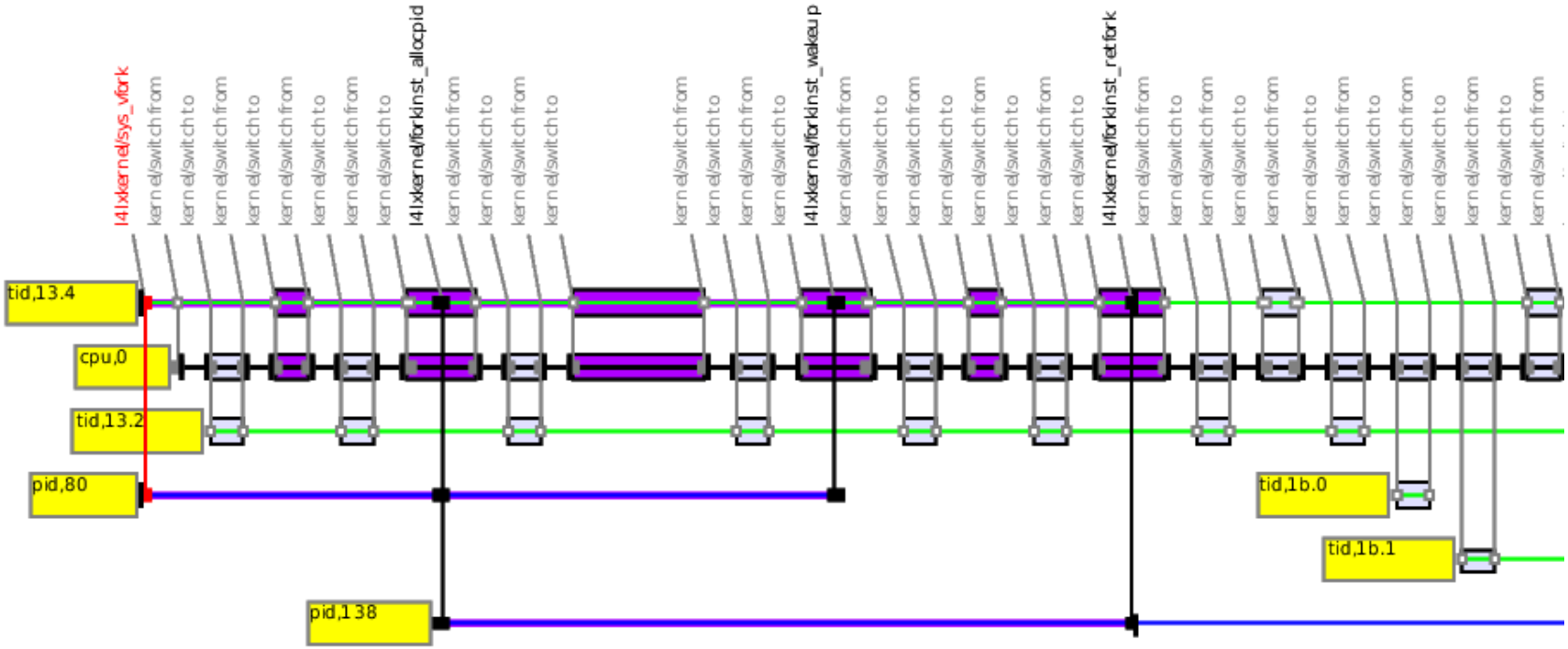
Inspecting fork/vfork in (L4)Linux

- kProbes at certain locations inside `do_fork()` routine:
  - call to `alloc_pid()`
  - call to `wake_up_new_task()`
  - function return
- More probes:
  - detect system calls:
    - `l4x_dispatch_syscall()` (inline...)
    - `system_call`
  - `ret_from_fork` - start instruction of child process
  - `exit` system call

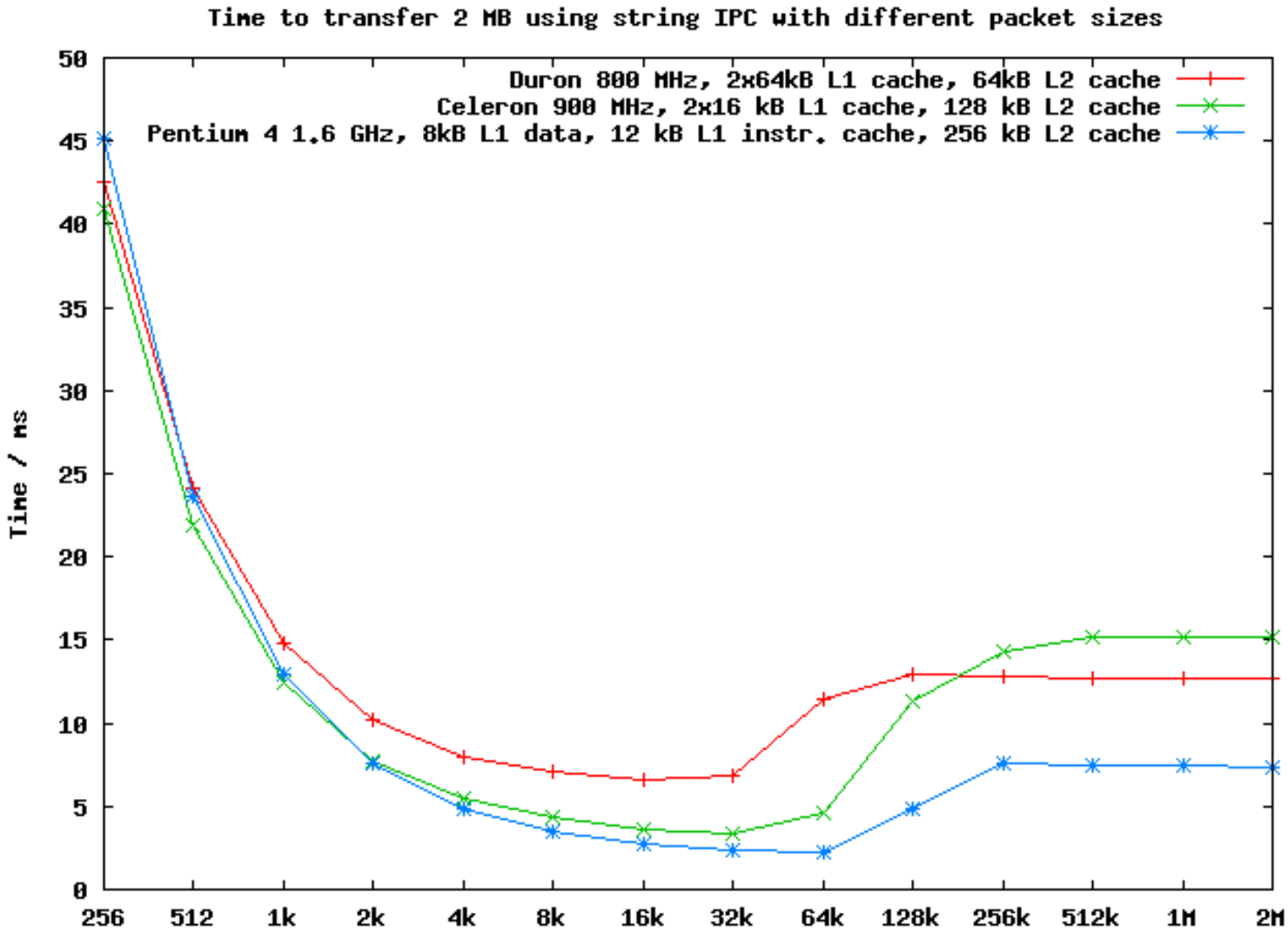
Inspecting fork/vfork in (L4)Linux

- kProbes at certain locations inside `do_fork()` routine:
  - call to `alloc_pid()`
  - call to `wake_up_new_task()`
  - function return
- More probes:
  - detect system calls:
    - `l4x_dispatch_syscall()` (inline...)
    - `system_call`
  - `ret_from_fork` - start instruction of child process
  - `exit` system call
- Context switches (Fiasco tracebuffer, `__switch_to()`)
- Defined Magpie schema for visualization
- Found a peculiarity in L4Linux

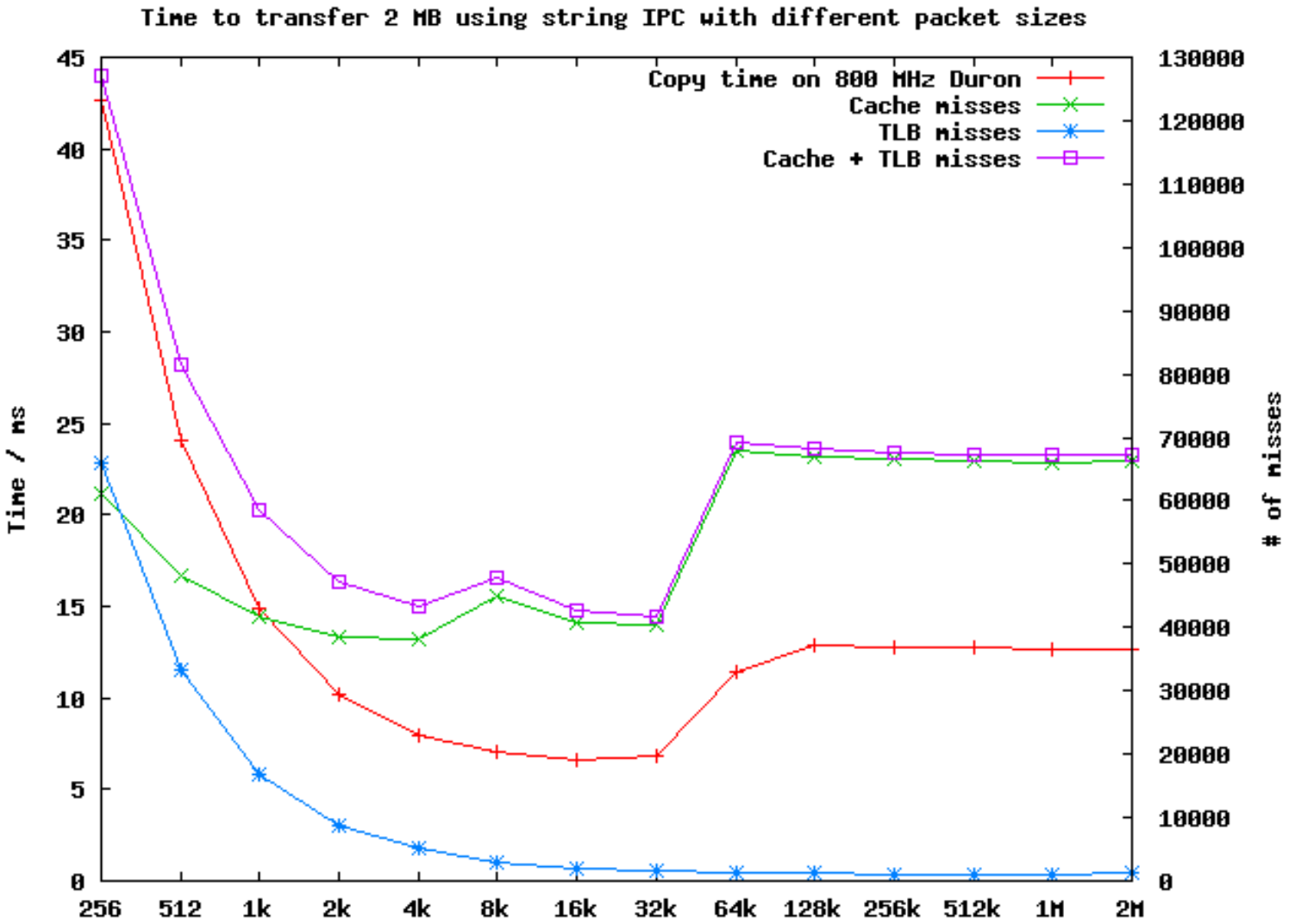
L4Linux scenario



String IPC performance



String IPC performance (2)



Evaluating L4Linux

- UnixBench on L4Linux and native Linux:
  - system call overhead: 8x slower
  - process creation: 5x slower
  - pipe throughput: 3x slower

Evaluating L4Linux

- UnixBench on L4Linux and native Linux:
  - system call overhead: 8x slower
  - process creation: 5x slower
  - pipe throughput: 3x slower
- System call execution time (in cycles):

	native Linux	L4Linux
`sys_brk`	1,800	2,800
`sys_llseek`	1,089	1,276
`sys_open`	20,865	21,579
`sys_clone`	62,604	161,775

Evaluating L4Linux

- UnixBench on L4Linux and native Linux:
  - system call overhead: 8x slower
  - process creation: 5x slower
  - pipe throughput: 3x slower
- System call execution time (in cycles):

	native Linux	L4Linux
`sys_brk`	1,800	2,800
`sys_llseek`	1,089	1,276
`sys_open`	20,865	21,579
`sys_clone`	62,604	161,775

- Two main reasons:
  - two context switches per system call
  - interaction with L4 servers

Context switch issues

Hypothesis: Context switch overhead decreases performance.

- 100,000 executions of `sys_getpid()`

	native Linux	L4Linux
User-space execution time in cycles (avg)	274	3,778
Instruction cache misses for 100,000 calls	544	1,703,332
Data Cache misses for 100,000 calls	286	402,398
ITLB misses for 100,000 calls	35	501,679
DTLB misses for 100,000 calls	141	1,052,520

L4Linux task creation

Hypothesis: Cooperation with L4 servers can be optimized.

- implemented transfer of chief rights to L4Linux server
- implemented a task cache in L4Linux

	L4Linux unmodified	+ new task management	+ task caching
`fork()`	330,425	292,857	255,087
`vfork()`	45,502	44,274	45,510
`execve()`	237,014	243,574	207,376

Conclusion

Request tracking throughout DROPS components

- Applicable for online and offline monitoring
- Useful for performance evaluation

Future work

- LTTng for L4Linux
- Autonomic Computing
- Model-carrying code
- Regression performance evaluation
- Inspect larger scenarios

Thank you for attending.
Any Questions?