POLYTECHNIQUE MONTRÉAL

LE GÉNIE EN PREMIÈRE CLASSE

Diagnosing Performance Variations by Comparing Execution Traces

François Doray

Progress Report Meeting – May 2015

Introduction

Performance

is a critical requirement

Sources of performance variations

- Update to a program, library or OS
- Interaction between tasks
- Programming error
- Different system load

Developers are not aware of this

Tracing

- Lots of details

Can we facilitate the diagnosis of performance variations with an algorithm that automatically identifies differences between groups of execution traces?

1.
Literature Review

2.
Solution

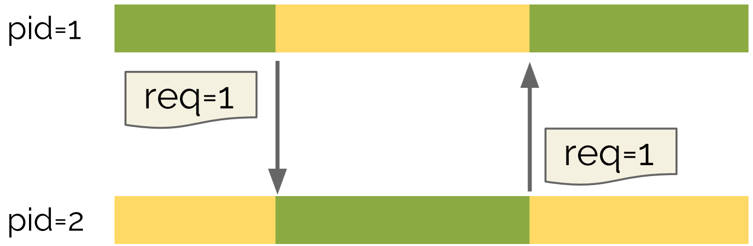
3.
Case Studies

4.
Performance Evaluation

1. Literature Review / Extracting Task Executions

Dapper Sigelman & al. (2010)

- Associate an identifier to incoming requests.
- Propagate the identifier.



Critical Path in TraceCompass

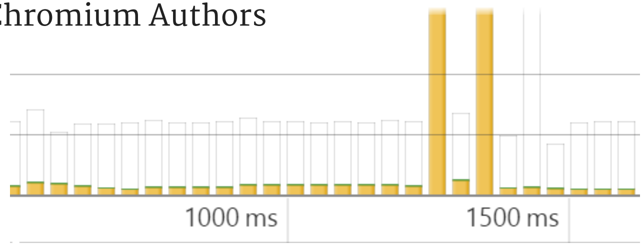
Giraldeau & Dagenais

- Heuristic based on kernel events.

1. Literature Review / Comparing Task Executions

“Frames” mode of Chrome

Chromium Authors



Differential Flame Graphs Gregg (2014)

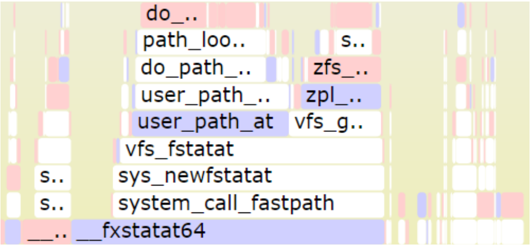
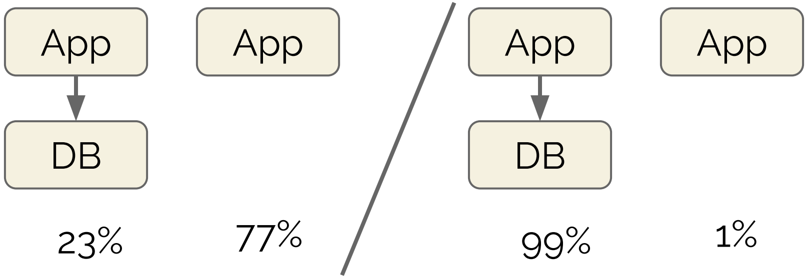


Image credit: Brendan Gregg / With permission.

Spectroscope Sambasivan & al. (2007)



TraceDiff Trumper & al. (2013)

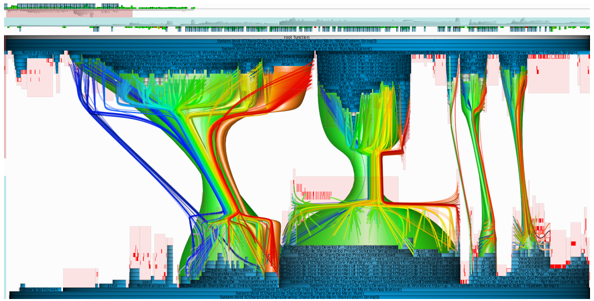
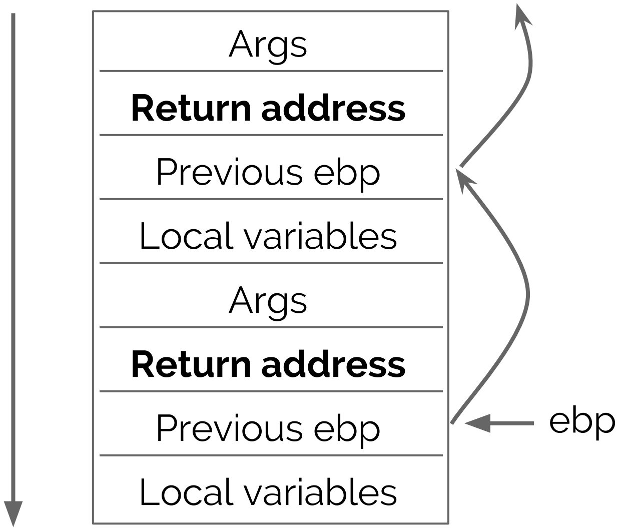


Image credit: Jonas Trümper / With permission.

With Frame Pointer

- Traverse a linked list.



Without Frame Pointer

Oakley and Bratus (2011)

- Extract rules from the .eh_frame section of ELF.
- Implemented by libunwind.

IP	CFA	ebp	eip
0x0001	...	...	...
0x0002	...	...	...

1.
Literature Review

2.
Solution

3.
Case Studies

4.
Performance Evaluation

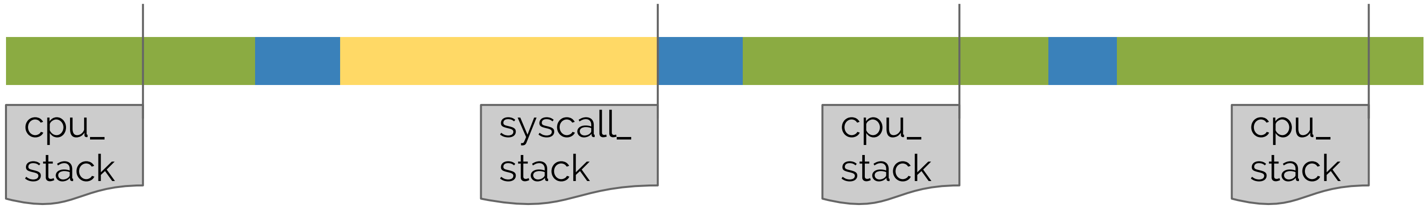
2. Solution / Tracing call stacks

cpu_stack

- Generated periodically when a thread is running.
- Using ITIMER_PROF.

syscall_stack

- Generated on long system calls.
- Duration of system calls tracked in a kernel module.
- Stack captured from a signal handler.



2. Solution / Tracing call stacks

cpu_stack

- Generated periodically when a thread is running.
- Using ITIMER_PROF.

syscall_stack

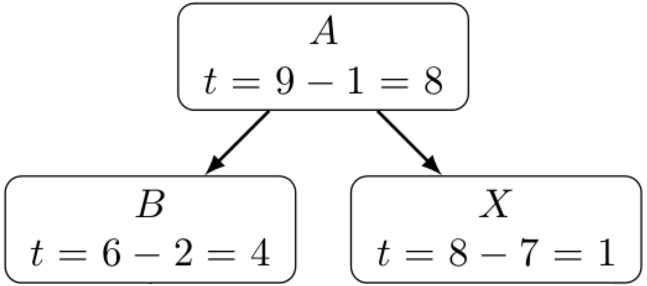
- Generated on long system calls.
- Duration of system calls tracked in a kernel module.
- Stack captured from a signal handler.

+

Kernel events to compute the critical path

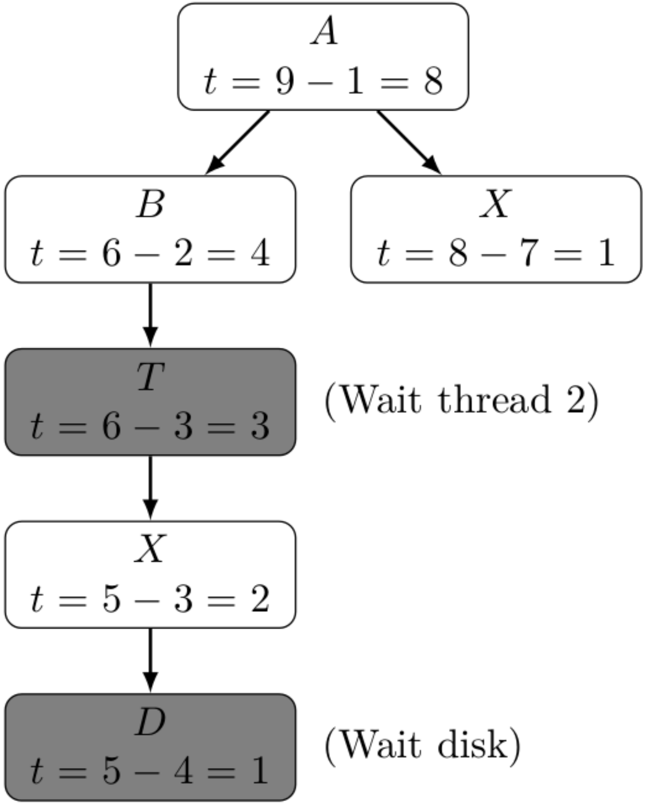
2. Solution / Enhanced Calling Context Tree

Time	Thread 1
1	Call A
2	Call B
3	
4	
5	
6	Return B
7	Call X
8	Return X
9	Return A



2. Solution / Enhanced Calling Context Tree

<table>
<tr><th>Time</th><th>Thread 1</th><th>Thread 2</th></tr>
<tr><td>1</td><td>Call A</td><td></td></tr>
<tr><td>2</td><td>Call B</td><td></td></tr>
<tr><td>3</td><td rowspan="3">Wait thread 2</td><td>Call X</td></tr>
<tr><td>4</td><td>Wait disk</td></tr>
<tr><td>5</td><td>Return X</td></tr>
<tr><td>6</td><td>Return B</td><td></td></tr>
<tr><td>7</td><td>Call X</td><td></td></tr>
<tr><td>8</td><td>Return X</td><td></td></tr>
<tr><td>9</td><td>Return A</td><td></td></tr>
</table>

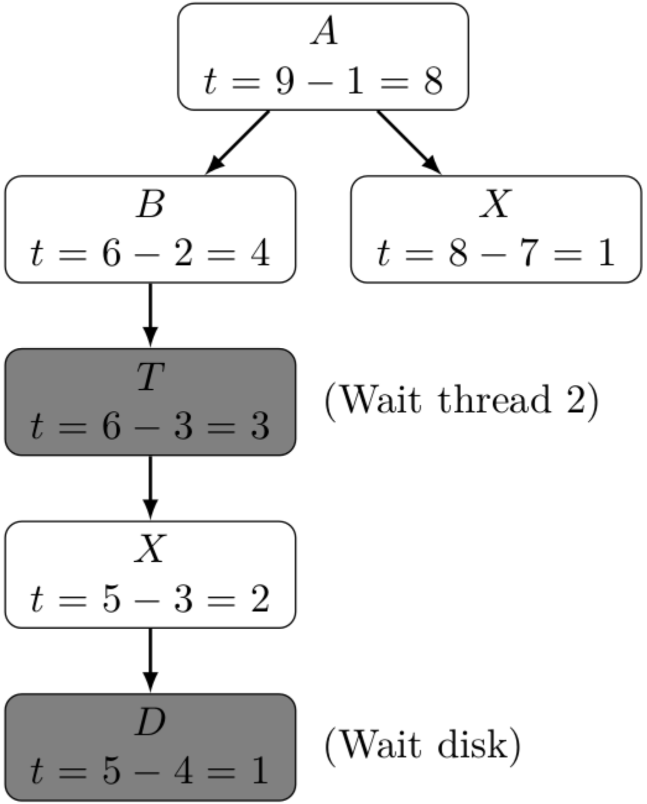


2. Solution / Enhanced Calling Context Tree

- Any type of latency.
  - CPU usage
  - Disk / network
  - Dependencies between threads

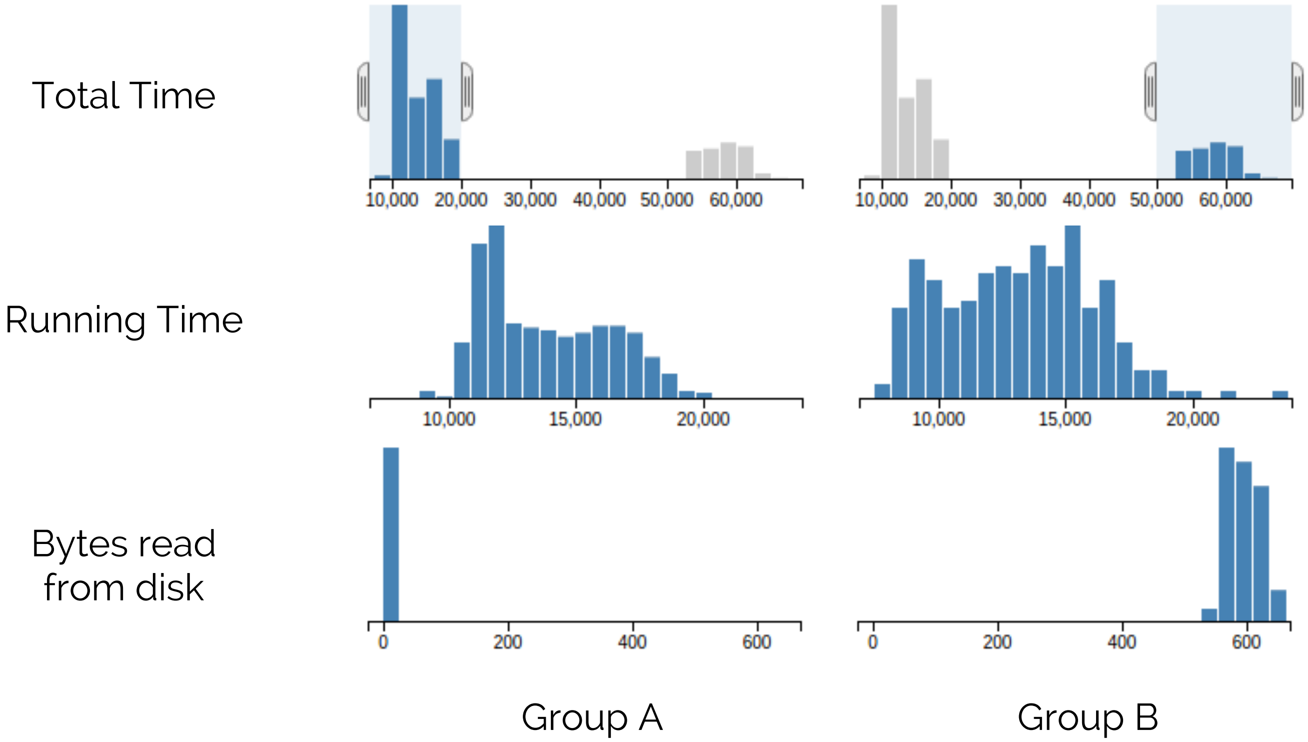
- Context of each latency.

- State History Tree.



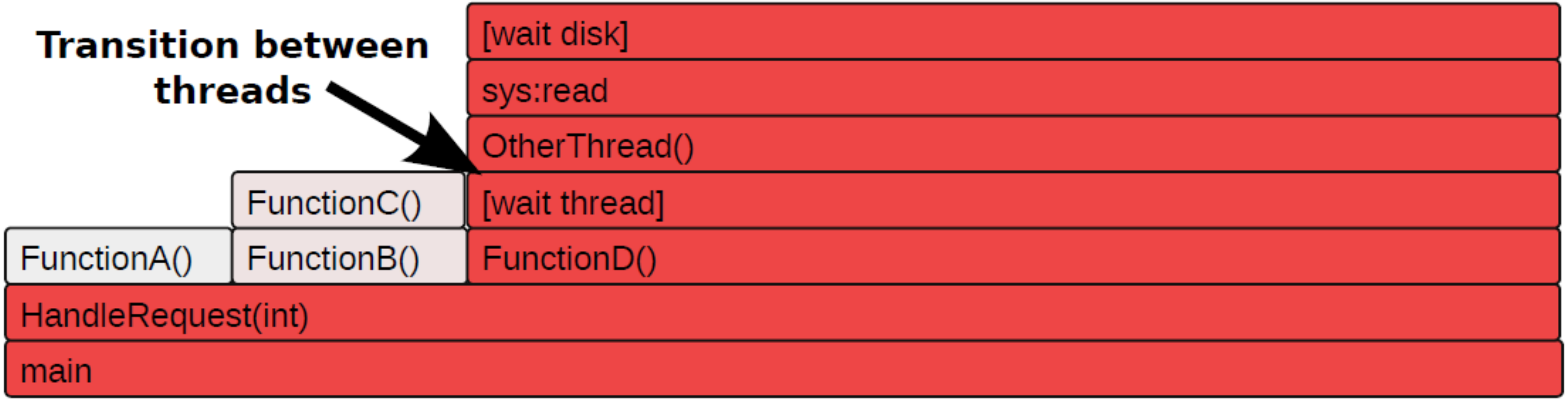
2. Solution / Comparison View

Filters to build groups of executions.



2. Solution / Comparison View

« Enhanced» Differential Flame Graph



1.
Literature Review

2.
Solution

3.
Case Studies

4.
Performance Evaluation

3. Case Studies

MUTEX

Mutex held during a long operation for no reason.

In MongoDB.

SLEEP

Using sleeps to synchronize threads.

In MongoDB.

PREEMPTION

Critical operation preempted by a low priority thread.

DISK

Web request slowed down by the OS committing data to the disk.

3. Case Studies

MUTEX

Mutex held during a long operation for no reason.

In MongoDB.

SLEEP

Using sleeps to synchronize threads.

In MongoDB.

PREEMPTION

Critical operation preempted by a low priority thread.

DISK

Web request slowed down by the OS committing data to the disk.

Demo

Try it yourself in a browser:

fdoray.github.io/tracecompare

1.
Literature Review

2.
Solution

3.
Case Studies

4.
Performance Evaluation

4. Performance Evaluation / Global overhead

PRIME

CPU-Only.

Stacks: 0.1%

Stacks + critical: **0.2%**

BABELTRACE

Short system calls.

Stacks: 1%

Stacks + critical: 1%

FIND

Long disk requests.

Stacks: 2%

Stacks + critical: 5%

MONGOD

Multi-thread.

Stacks: 2%

Stacks + critical: **9%**

4. Performance Evaluation

PRIME

PU-Only.

Stacks: 0.1%

Stacks + critical: **0.2%**

ETW on Windows: 0.0%
DTrace on Mac: 1.0%

BABELTRACE

Short system calls.

Stacks: 1%

Stacks + critical: 1%

FIND

Long disk requests.

Stacks: 2%

Stacks + critical: 5%

MONGOD

Multi-threac

Stacks: 2%

Stacks + critical: **9%**

ETW on Windows: 19%
DTrace on Mac: 22%

* MacBook Pro with Quad-core Intel® Core i7™-3720QM at 2.6 GHz, 8 GB RAM, SSD for Windows and Mac benchmarks.

Conclusion

Summary

- Trace **call stacks**.
- **Enhanced calling context trees**.
- Compare groups of executions using **histograms** and **flame graphs**.
- **Works** with open-source and enterprise apps.

Future Work

- Support more interactions:
  - VMs
  - GPU
  - Application-specific
- Dynamic languages / JIT
- Support code refactoring

Thanks!

ANY QUESTIONS?

Try the demo:

fdoray.github.io/tracecompare

References

The Chromium Authors, “Performance profiling with the timeline”, https://developer.chrome.com/devtools/docs/timeline, consulted on March 24th 2015.

F. Giraldeau and M. R. Dagenais, “Approximation of critical path using low-level system events”, not published yet.

B. Gregg, “Differential flame graphs”, http://www.brendangregg.com/blog/2014-11-09/differential-flame-graphs.html, November 2014, consulted on March 24th, 2015.

J. Oakley and S. Bratus, “Exploiting the hard-working dwarf : Trojan and exploit techniques with no native executable code”, in Proceedings of the 5th USENIX Conference on Offensive Technologies, WOOT’11. Berkeley, CA, USA : USENIX Association, 2011, p. 11.

R. R. Sambasivan, A. X. Zheng, E. Thereska, and G. R. Ganger, “Categorizing and differencing system behaviours”, Hot Topics in Autonomic Computing, p. 2, June 2007.

B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag, “Dapper, a large-scale distributed systems tracing infrastructure”, Google research, 2010.

J. Trumper, J. Dollner, and A. Telea, “Multiscale visual comparison of execution traces”, in IEEE 21st International Conference on Program Comprehension (ICPC), May 2013, pp. 53–62. DOI : 10.1109/ICPC.2013.6613833.

Credits

Presentation by François Doray, master's student at the Distributed open reliable systems analysis lab (DORSAL) of Polytechnique Montreal.

Special thanks to SlidesCarnival for releasing this presentation template for free (CC BY 4.0).