POLYTECHNIQUE MONTRÉAL

LE GÉNIE EN PREMIÈRE CLASSE

TraceCompare: Automatic Identification of Differences between Executions

François Doray

Tracing Summit - August 2015

Introduction

Performance

is a critical requirement

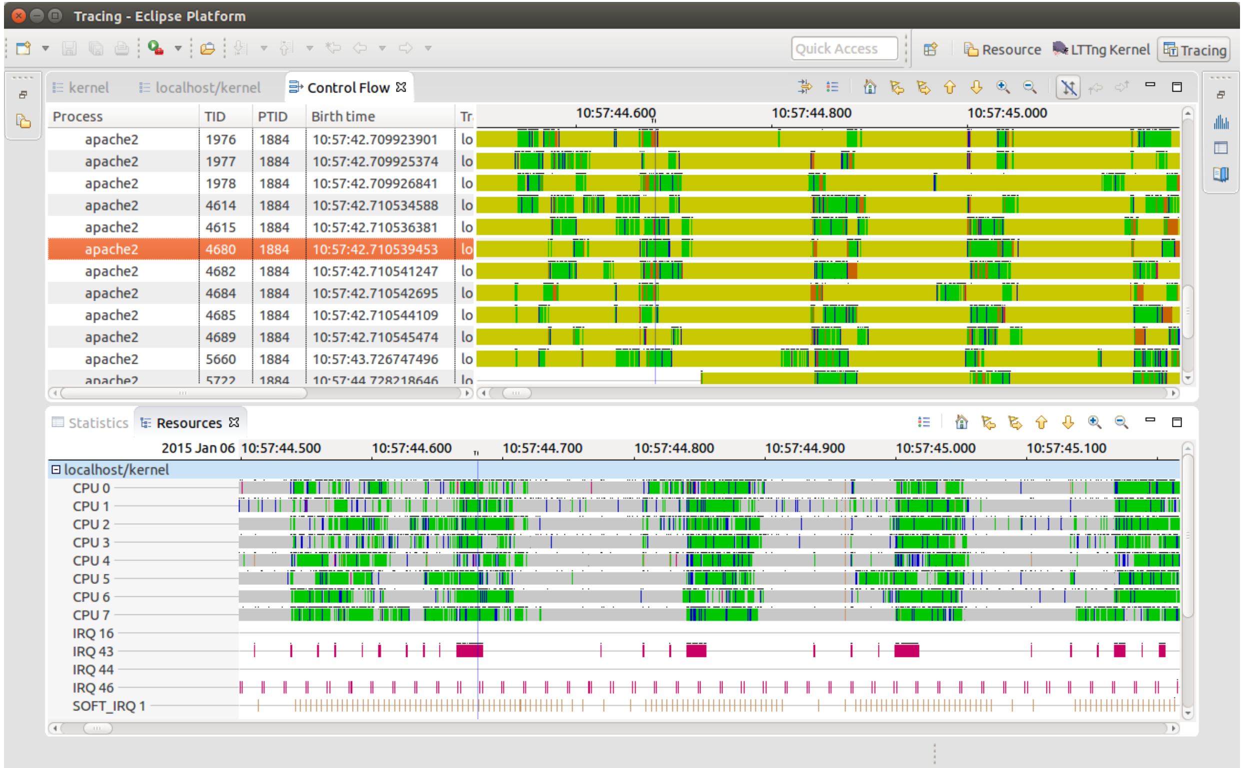
Sources of performance variations

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

Developers don't understand 100% of the systems they develop.

Tracing: Record events that occur during the execution of a system.

Introduction



View a trace in **TraceCompass**

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

1.
Related Work

2.
Solution

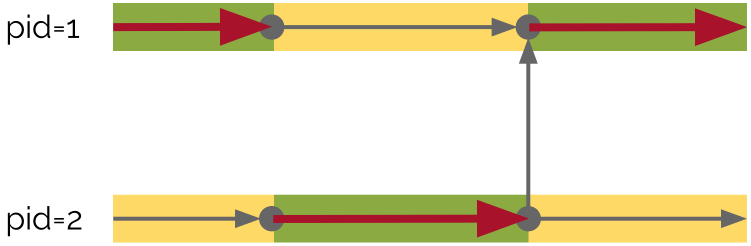
3.
Case Studies

4.
Performance Evaluation

1. Related Work / Extracting Task Executions

Approximation of Critical Path Giraldeau & Dagenais

- Heuristic that uses kernel events to build:
  - Graph of dependencies between threads.
  - List of segments that belong to the critical path of an execution.

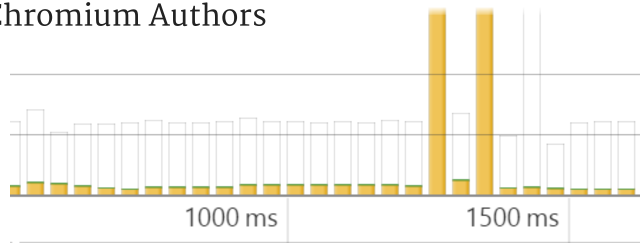


Alternate solution: **Dapper** Sigelman & al. (2010)

1. Related Work / Comparing Task Executions

"Frames" mode of Chrome

Chromium Authors



Differential Flame Graphs Gregg (2014)

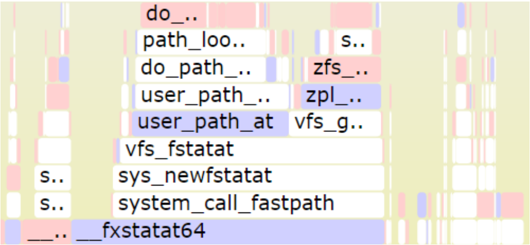
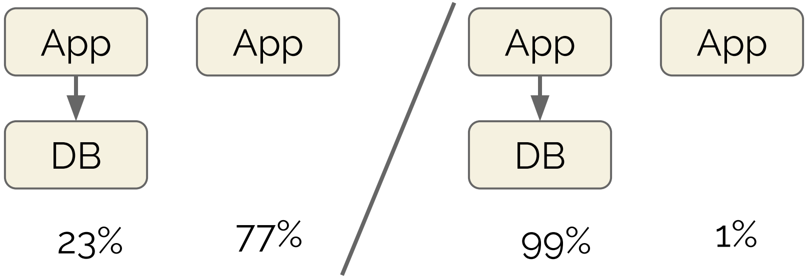


Image credit: Brendan Gregg / With permission.

Spectroscope Sambasivan & al. (2007)



TraceDiff Trumper & al. (2013)

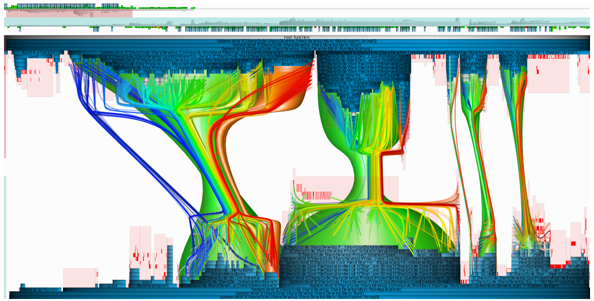


Image credit: Jonas Trümper / With permission.

1.
Related Work

2.
Solution

3.
Case Studies

4.
Performance Evaluation

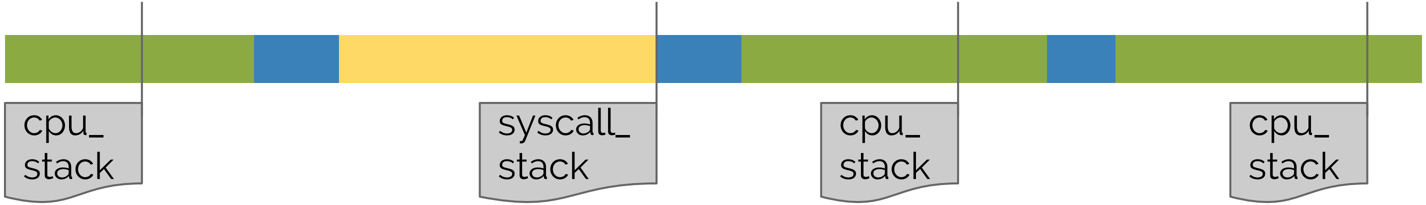
2. Solution / Required Events

cpu_stack

- Generated periodically when a thread is on the CPU.
- Uses 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 / Required Events

cpu_stack

- Generated periodically when a thread is on the CPU.
- Uses ITIMER_PROF.

syscall_stack

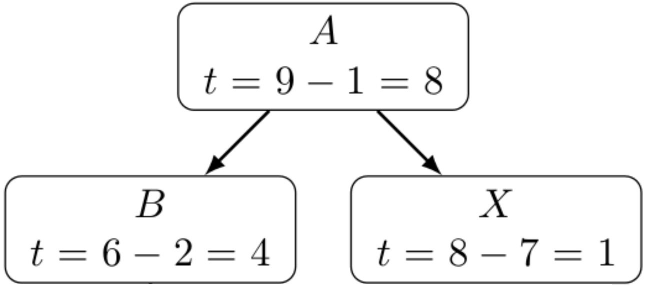
- Generated on long system calls.
- Duration of system calls tracked in a kernel module.
- Stack captured from a signal handler.

Kernel Events

- To compute the critical path of executions.

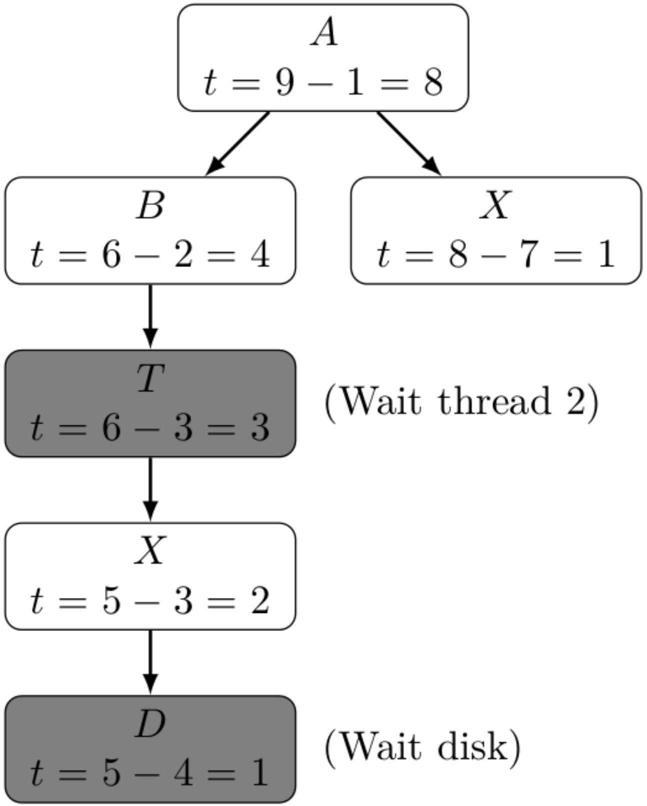
2. Solution / Enhanced Calling Context Tree

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

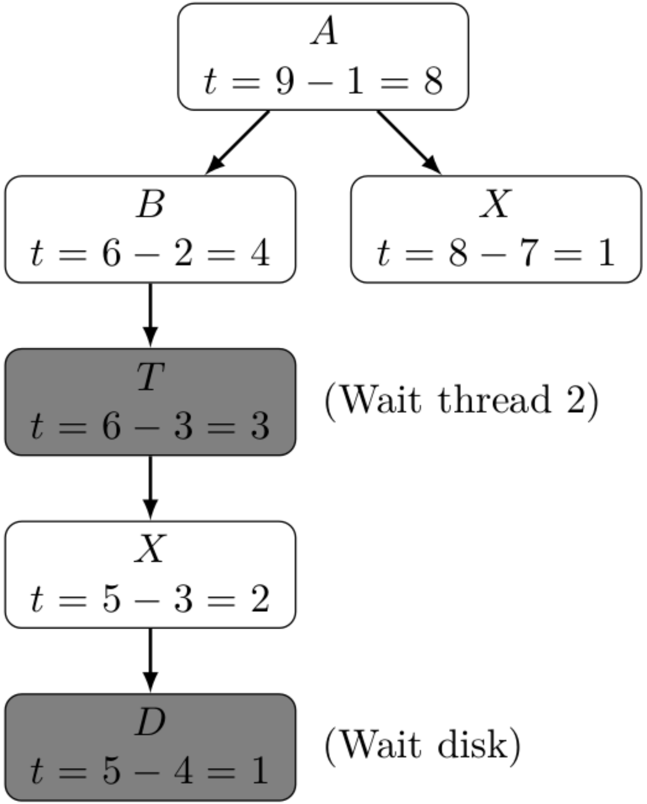
<table>
<tr><th>Time</th><th>Thread 1</th><th>Thread 2</th></tr>
<tr><td>1</td><td>Call A</td><td></td></tr>
<tr><td>2</td><td>Call B</td><td></td></tr>
<tr><td>3</td><td rowspan="3">Wait thread 2</td><td>Call X</td></tr>
<tr><td>4</td><td>Wait disk</td></tr>
<tr><td>5</td><td>Return X</td></tr>
<tr>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

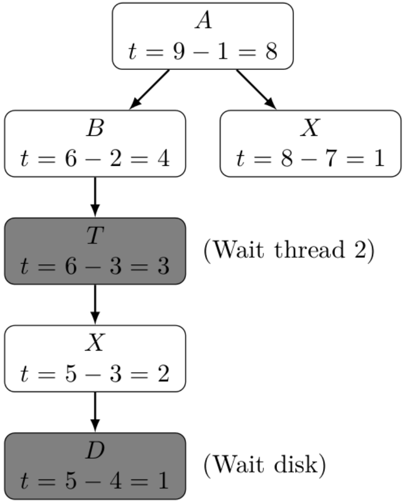
- Any type of latency.
  - CPU usage
  - Disk / network
  - Dependencies between threads

- Context of each latency.



2. Solution / Enhanced Calling Context Tree

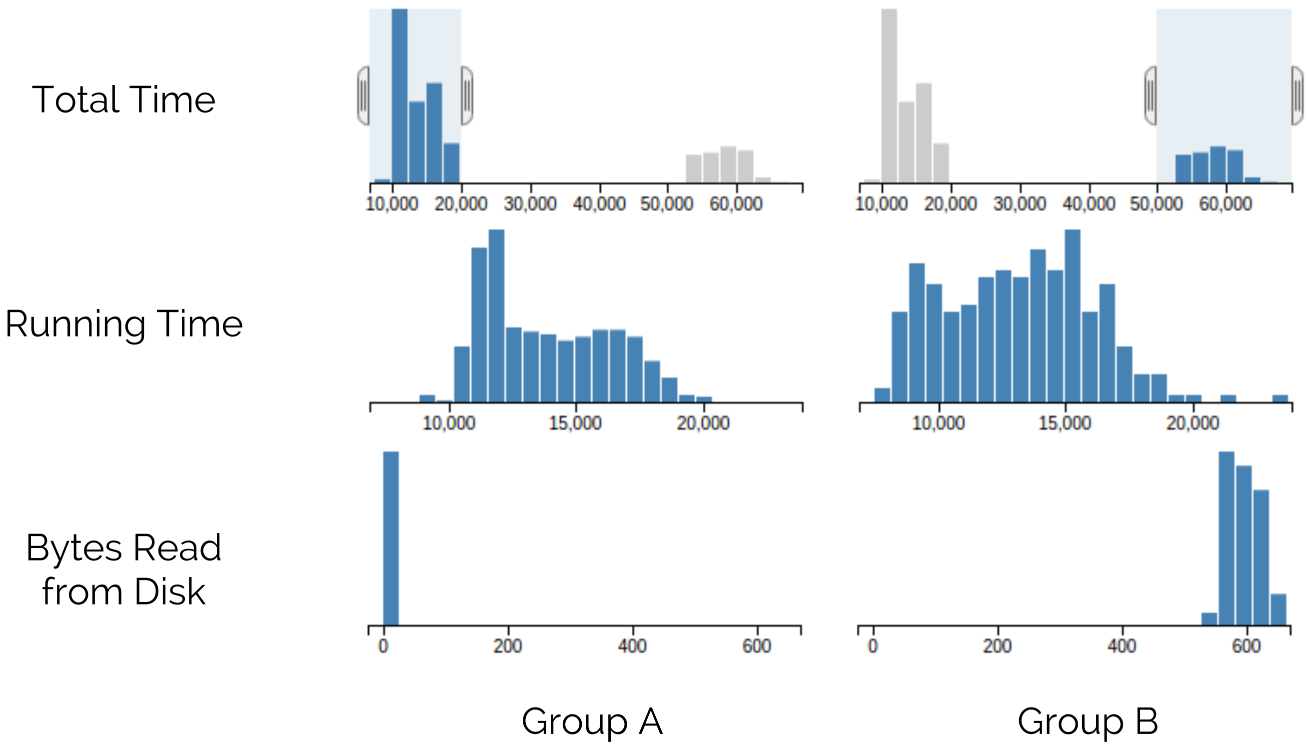
```
[18:10:27.684] sys_write_entry: { cpu_id = 0 },
{ fd = 4, count = 1024 }

[18:10:27.783] sys_write_exit: { cpu_id = 0 },
{ ret = 0 }

[18:10:28.093] sched_switch: { cpu_id = 0 },
{ prev_tid = 4, next_tid = 10 }

[18:10:28.689] app:hello: { cpu_id = 0 },
{ str = "Hello World!" }
```



- State History Tree

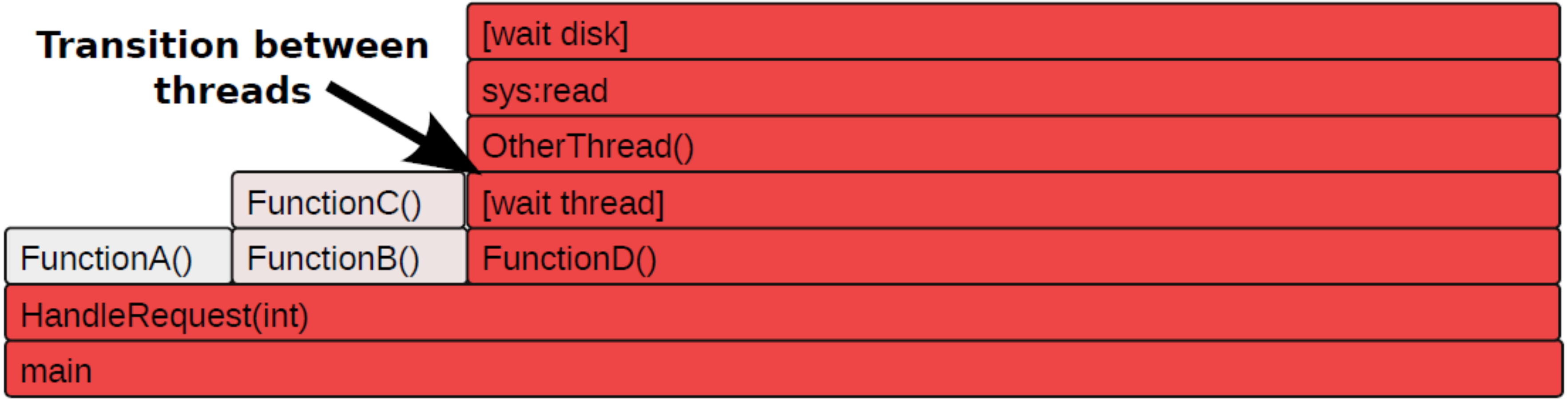
2. Solution / Comparison View

Filters to build groups of executions.



2. Solution / Comparison View

« Enhanced» Differential Flame Graph



- Red = time difference between compared groups.

1.
Related Work

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.

http://fdoray.github.io/tracecompare/tracecompare.html?data=mongowrite

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.

ZZZZZZZzzz

PREEMPTION

Critical operation preempted by a low priority thread.

DISK

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

http://fdoray.github.io/tracecompare/tracecompare.html?data=preempt

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.

1.
Related Work

2.
Solution

3.
Case Studies

4.
Performance Evaluation

4. Performance Evaluation / Overhead

Application	LTTng overhead
Prime CPU only.	0.2%
Find Long disk requests.	5%
Mongod Interactions between threads.	9%

* Quad-core Intel® Core™i7-3770 CPU @ 3.4 GHz, 16 GB RAM, 7200 RPM hard drive.

4. Performance Evaluation / Overhead Comparison

Application	LTTng Overhead (Linux)	DTrace Overhead (Mac)	ETW Overhead (Windows)
Prime CPU only.	-0.1% ±0.3%	1.0% ±0.1%	0.0% ±0.1%
Mongod Interactions between threads.	8% ±1%	24% ±0%	24% ±1%

* 95% confidence intervals.

* **MacBook Pro** Quad-core Intel® Core i7™-3720QM @ 2.6 GHz, 8 GB RAM, SSD.

Conclusion

Summary

- Trace **call stacks**.
- **Enhanced calling context trees**.
- Compare groups of executions using **filters** and **flame graphs**.

- **Works** with open-source and enterprise apps.

Future Work

- Support more interactions:
  - VMs
  - GPUs
- Dynamic languages / JIT
- Support code refactoring

Thanks!

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 25, 2015.

F. Giraldeau and M. R. Dagenais, “Approximation of critical path using low-level system events”, to be published.

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