Enhanced Statistical Debugging for Adaptive Monitoring

Mohammed Adib Khan
ak19qp@brocku.ca

Dr. Morteza Noferesti
mnoferesti@brocku.ca

Dr. Naser Ezzati-Jivan
nezzati@brocku.ca

Department Of Computer Science

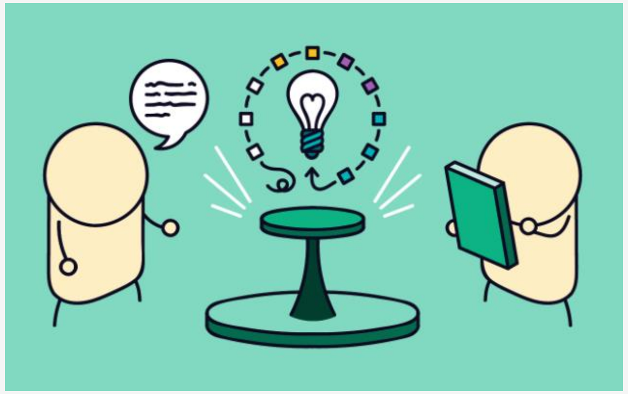
Brock University, Canada

Introduction

- ❑ Monitoring and debugging functions is essential, particularly when kernel-level operations experience long wait times.
- ❑ Some examples of kernel-level waiting time metrics could drive from:
    - ➢ Syscalls
    - ➢ Block_rqs
    - ➢ Sched_Switch
    - ➢ Irq_handler
- ❑ Prolonged delays in kernel-level operations can severely impact overall system performance and could potentially lead to system failures.
- ❑ Statistical debugging techniques offer powerful solutions.

Problem Statement & Objectives

- The primary goal is to identify candidate functions and function paths for adaptive or selective tracing, crucial for diagnosing performance issues.
- To achieve this, we have two key objectives:
    - Identifying problematic functions
    - Determining potential problematic function paths



Objective 1 – Identifying Problematic Functions

- The first step involves pinpointing the functions responsible for kernel-level activity delays by examining correlation.
- Our focus lies on detecting application-related bugs within specific functions.
- If a function and a performance bug shows a high correlation, our method will identify that function.

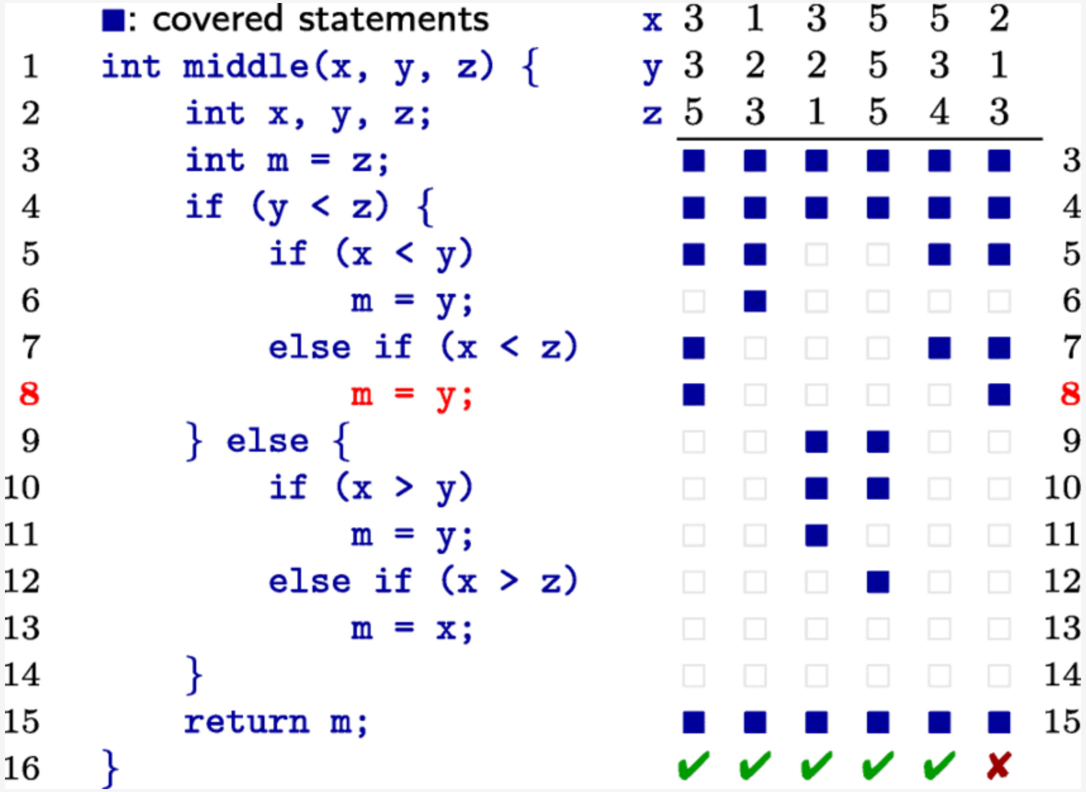
Objective 2 – Determining Function Paths

- Simply identifying the functions does not suffice for our ultimate objective.
- The next step is to ascertain the function path leading to these problematic or intriguing functions.
- This path is indispensable for adaptive tracing, as merely tracing the function itself would lack sufficient context for effective analysis.

$$Path^{i} = \langle F^{main}, F^{1}, F^{2}, ..., F^{i} \rangle$$

Background – Statistical Debugging

- ❑ Statistical debugging is a method that identifies defects by examining the correlation between program states and failures.
- ❑ It collects program execution data, isolating elements most related to failures by comparing 'successful' and 'failing' runs.
- ❑ Typically, it focuses on line-code level and conditions within the code.
- ❑ It is effective for consistently reproducible issues, aiding in bug localization in complex and large codebases.
- ❑ It provides vital clues to developers to understand and fix the defects.



■: covered statements

		x	3	1	3	5	5	2	
		y	3	2	2	5	3	1	
		z	5	3	1	5	4	3	
1	`int middle(x, y, z) {`								
2	`int x, y, z;`								
3	`int m = z;`		■	■	■	■	■	■	3
4	`if (y < z) {`		■	■	■	■	■	■	4
5	`if (x < y)`		■	■	□	□	■	■	5
6	`m = y;`		□	■	□	□	□	□	6
7	`else if (x < z)`		■	□	□	□	■	■	7
8	`m = y;`		■	□	□	□	□	■	8
9	`} else {`		□	□	■	■	□	□	9
10	`if (x > y)`		□	□	■	■	□	□	10
11	`m = y;`		□	□	■	□	□	□	11
12	`else if (x > z)`		□	□	□	■	□	□	12
13	`m = x;`		□	□	□	□	□	□	13
14	`}`		□	□	□	□	□	□	14
15	`return m;`		■	■	■	■	■	■	15
16	`}`		✓	✓	✓	✓	✓	✗	

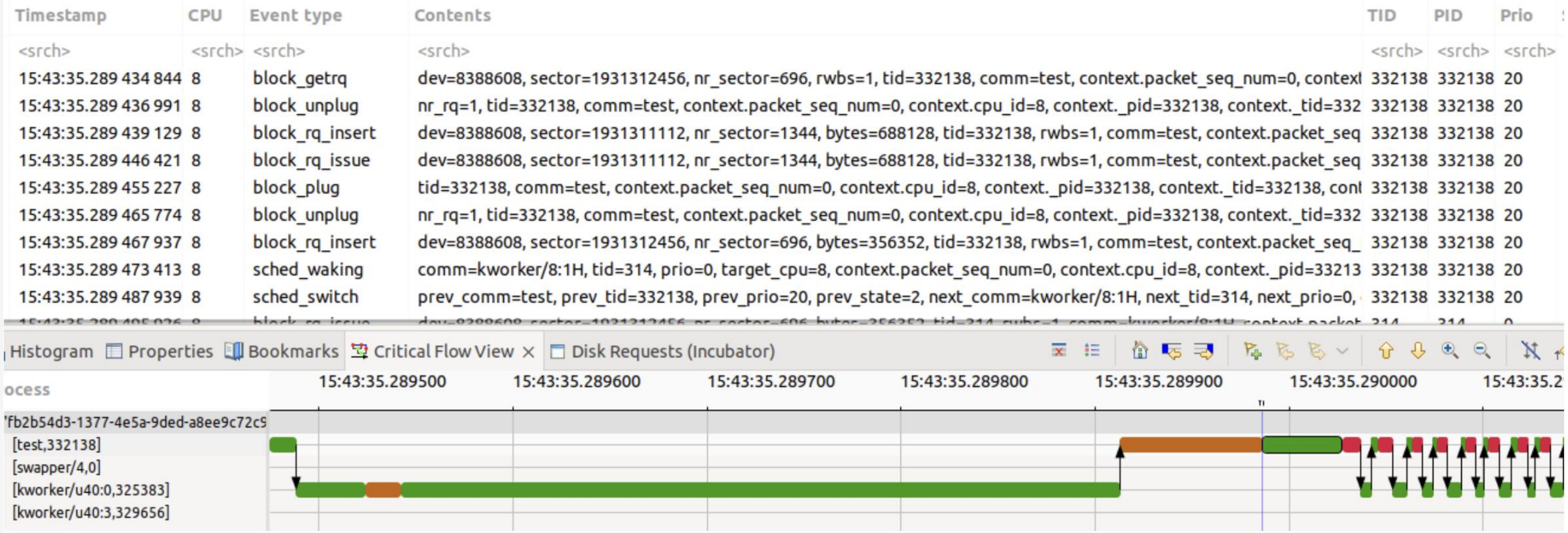
Image credits: Jones and Harrold 2005

Background – Enhanced Statistical Debugging

- Enhanced Statistical Debugging is our refined version of traditional statistical debugging.
- Traditional statistical debugging analyses bug correlations at a line-code level, examining conditional constructs within the program code.
- In contrast, our Enhanced Statistical Debugging shifts focus to a function level, examining the correlation between specific function execution and performance issues.
- This function-level analysis aids in identifying potential bottlenecks, offering a more precise tool for diagnosing and resolving performance concerns.

Methodology – Monitoring Kernel Trace

- ❑ Our approach includes monitoring prolonged activities in the kernel trace of our application, such as waiting times (for net, disk, CPU, etc.), syscalls, interrupts, etc.
- ❑ For each protracted activity, we ensure to gather enough stack trace data to perform meaningful analysis on them.



Timestamp	CPU	Event type	Contents	TID	PID	Prio
<srch>	<srch>	<srch>	<srch>	<srch>	<srch>	<srch>
15:43:35.289 434 844	8	block_getrq	dev=8388608, sector=1931312456, nr_sector=696, rwbs=1, tid=332138, comm=test, context.packet_seq_num=0, context	332138	332138	20
15:43:35.289 436 991	8	block_unplug	nr_rq=1, tid=332138, comm=test, context.packet_seq_num=0, context.cpu_id=8, context._pid=332138, context._tid=332	332138	332138	20
15:43:35.289 439 129	8	block_rq_insert	dev=8388608, sector=1931311112, nr_sector=1344, bytes=688128, tid=332138, rwbs=1, comm=test, context.packet_seq	332138	332138	20
15:43:35.289 446 421	8	block_rq_issue	dev=8388608, sector=1931311112, nr_sector=1344, bytes=688128, tid=332138, rwbs=1, comm=test, context.packet_seq	332138	332138	20
15:43:35.289 455 227	8	block_plug	tid=332138, comm=test, context.packet_seq_num=0, context.cpu_id=8, context._pid=332138, context._tid=332138, con	332138	332138	20
15:43:35.289 465 774	8	block_unplug	nr_rq=1, tid=332138, comm=test, context.packet_seq_num=0, context.cpu_id=8, context._pid=332138, context._tid=332	332138	332138	20
15:43:35.289 467 937	8	block_rq_insert	dev=8388608, sector=1931312456, nr_sector=696, bytes=356352, tid=332138, rwbs=1, comm=test, context.packet_seq_	332138	332138	20
15:43:35.289 473 413	8	sched_waking	comm=kworker/8:1H, tid=314, prio=0, target_cpu=8, context.packet_seq_num=0, context.cpu_id=8, context._pid=33213	332138	332138	20
15:43:35.289 487 939	8	sched_switch	prev_comm=test, prev_tid=332138, prev_prio=20, prev_state=2, next_comm=kworker/8:1H, next_tid=314, next_prio=0,	332138	332138	20

Methodology – Enhanced Statistical Debugging

- These stack trace samples undergo analysis using statistical debugging.
- We accumulate ample data for both faulty and successful activities.
- Then, via statistical debugging, we strive to identify any correlation between the application's functions and these lengthy or faulty activities.
- We define fail runs as wait times exceeding mean + std, and the opposite for success. Other thresholds could also be used depending on the activity.
- Success/fail predicates correspond to the last function in the call stack.
- Success(observed) and failed(observed) corresponds to the functions which are observable at any place in the call stack.

$$Failure(P) = F(P)/(S(P) + F(P))$$

$$Observation(P) = \frac{F(Pobserved)}{S(Pobserved) + F(Pobserved)}$$

$$Increase(P) = Failure(P) - Observation(P)$$

Methodology – Function Path Sequence Mining

- ❑ Upon identification of the functions, we conduct further analysis (path sequence mining) to locate the most common paths leading to these kernel-level metrics delays.
- ❑ These functions, together with their associated paths, become candidates for tracing/logging.
- ❑ This targeted approach to tracing allows us to effectively concentrate our resources, negating the need for comprehensive tracing.

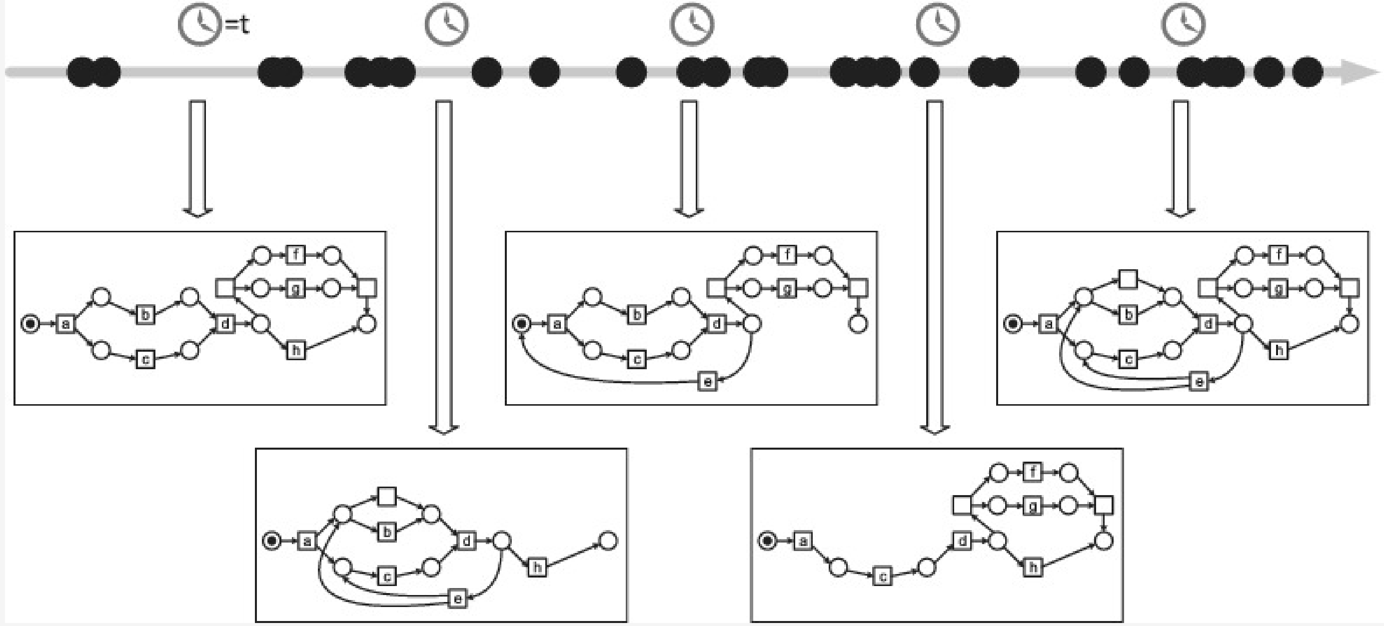


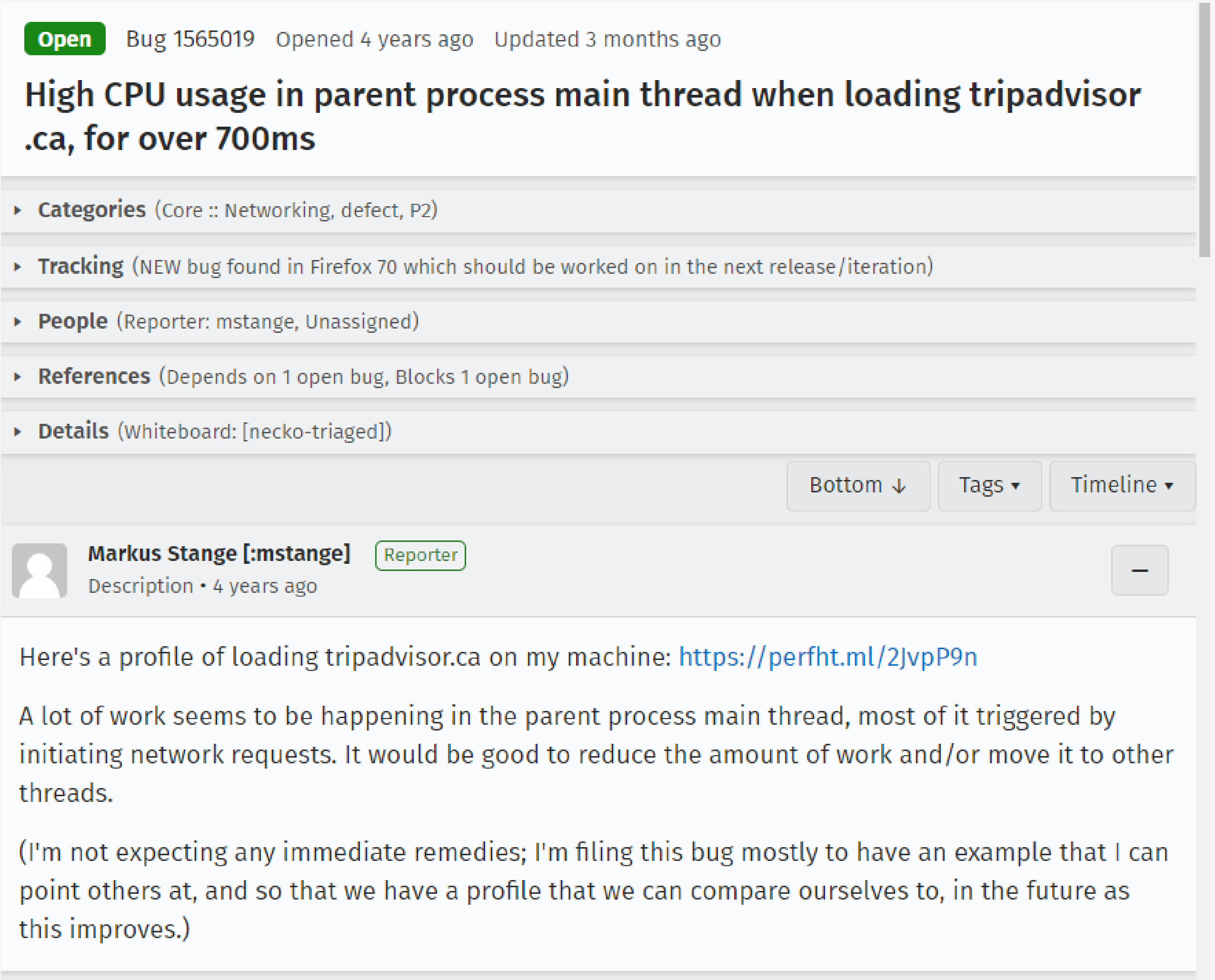
Image credits: Hassani, Marwan, et al. 2019

Methodology – Adaptive Monitoring

- ❑ Adaptive monitoring uses a dynamic, targeted approach to system tracing and debugging.
- ❑ It builds on insights from statistical debugging and path mining to focus on problematic functions and paths.
- ❑ Unlike exhaustive logging, this method saves resources by tracing only areas of interest.
- ❑ By reducing data volume and overhead, it prevents crucial information from being overlooked.

Case Study – Firefox

- ❑ We applied our methodology to a case study of a performance bug in Firefox.
- ❑ Our method successfully identified the correct function paths causing the bug.



Bug 1565019 Opened 4 years ago Updated 3 months ago

High CPU usage in parent process main thread when loading tripadvisor .ca, for over 700ms

Categories (Core :: Networking, defect, P2)

Tracking (NEW bug found in Firefox 70 which should be worked on in the next release/iteration)

People (Reporter: mstange, Unassigned)

References (Depends on 1 open bug, Blocks 1 open bug)

Details (Whiteboard: [necko-triaged])

Markus Stange [:mstange] Reporter

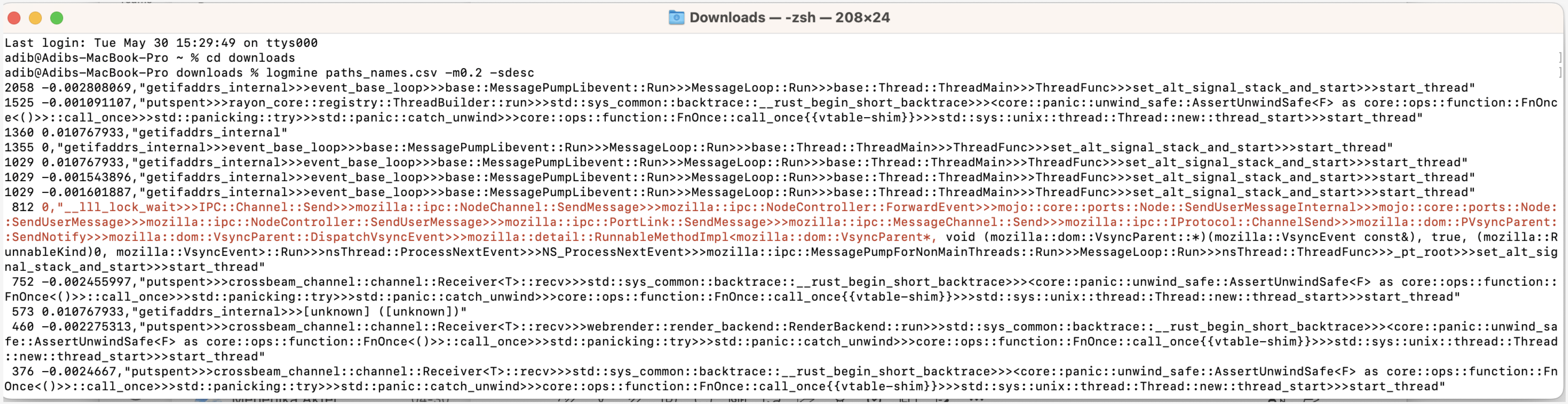
Description • 4 years ago

Here's a profile of loading tripadvisor.ca on my machine: https://perfht.ml/2JvpP9n

A lot of work seems to be happening in the parent process main thread, most of it triggered by initiating network requests. It would be good to reduce the amount of work and/or move it to other threads.

(I'm not expecting any immediate remedies; I'm filing this bug mostly to have an example that I can point others at, and so that we have a profile that we can compare ourselves to, in the future as this improves.)

Case Study – Firefox



```
Last login: Tue May 30 15:29:49 on ttys000
adib@Adibs-MacBook-Pro ~ % cd downloads
adib@Adibs-MacBook-Pro downloads % logmine paths_names.csv -m0.2 -sdesc
2058 -0.002808069,"getifaddrs_internal>>>event_base_loop>>>base::MessagePumpLibevent::Run>>>MessageLoop::Run>>>base::Thread::ThreadMain>>>ThreadFunc>>>set_alt_signal_stack_and_start>>>start_thread"
1525 -0.001091107,"putspent>>>rayon_core::registry::ThreadBuilder::run>>>std::sys_common::backtrace::__rust_begin_short_backtrace>>><core::panic::unwind_safe::AssertUnwindSafe<F> as core::ops::function::FnOnc
e<()>>::call_once>>>std::panicking::try>>>std::panic::catch_unwind>>>core::ops::function::FnOnce::call_once{{vtable-shim}}>>>std::sys::unix::thread::Thread::new::thread_start>>>start_thread"
1360 0.010767933,"getifaddrs_internal"
1355 0,"getifaddrs_internal>>>event_base_loop>>>base::MessagePumpLibevent::Run>>>MessageLoop::Run>>>base::Thread::ThreadMain>>>ThreadFunc>>>set_alt_signal_stack_and_start>>>start_thread"
1029 0.010767933,"getifaddrs_internal>>>event_base_loop>>>base::MessagePumpLibevent::Run>>>MessageLoop::Run>>>base::Thread::ThreadMain>>>ThreadFunc>>>set_alt_signal_stack_and_start>>>start_thread"
1029 -0.001543896,"getifaddrs_internal>>>event_base_loop>>>base::MessagePumpLibevent::Run>>>MessageLoop::Run>>>base::Thread::ThreadMain>>>ThreadFunc>>>set_alt_signal_stack_and_start>>>start_thread"
1029 -0.001601887,"getifaddrs_internal>>>event_base_loop>>>base::MessagePumpLibevent::Run>>>MessageLoop::Run>>>base::Thread::ThreadMain>>>ThreadFunc>>>set_alt_signal_stack_and_start>>>start_thread"
 812 0,"__lll_lock_wait>>>IPC::Channel::Send>>>mozilla::ipc::NodeChannel::SendMessage>>>mozilla::ipc::NodeController::ForwardEvent>>>mojo::core::ports::Node::SendUserMessageInternal>>>mojo::core::ports::Node:
:SendUserMessage>>>mozilla::ipc::NodeController::SendUserMessage>>>mozilla::ipc::PortLink::SendMessage>>>mozilla::ipc::MessageChannel::Send>>>mozilla::ipc::IProtocol::ChannelSend>>>mozilla::dom::PVsyncParent:
:SendNotify>>>mozilla::dom::VsyncParent::DispatchVsyncEvent>>>mozilla::detail::RunnableMethodImpl<mozilla::dom::VsyncParent*, void (mozilla::dom::VsyncParent::*)(mozilla::VsyncEvent const&), true, (mozilla::R
unnableKind)0, mozilla::VsyncEvent>::Run>>>nsThread::ProcessNextEvent>>>NS_ProcessNextEvent>>>mozilla::ipc::MessagePumpForNonMainThreads::Run>>>MessageLoop::Run>>>nsThread::ThreadFunc>>>_pt_root>>>set_alt_sig
nal_stack_and_start>>>start_thread"
 752 -0.002455997,"putspent>>>crossbeam_channel::channel::Receiver<T>::recv>>>std::sys_common::backtrace::__rust_begin_short_backtrace>>><core::panic::unwind_safe::AssertUnwindSafe<F> as core::ops::function::
FnOnce<()>>::call_once>>>std::panicking::try>>>std::panic::catch_unwind>>>core::ops::function::FnOnce::call_once{{vtable-shim}}>>>std::sys::unix::thread::Thread::new::thread_start>>>start_thread"
 573 0.010767933,"getifaddrs_internal>>>[unknown] ([unknown])"
 460 -0.002275313,"putspent>>>crossbeam_channel::channel::Receiver<T>::recv>>>webrender::render_backend::RenderBackend::run>>>std::sys_common::backtrace::__rust_begin_short_backtrace>>><core::panic::unwind_sa
fe::AssertUnwindSafe<F> as core::ops::function::FnOnce<()>>::call_once>>>std::panicking::try>>>std::panic::catch_unwind>>>core::ops::function::FnOnce::call_once{{vtable-shim}}>>>std::sys::unix::thread::Thread
::new::thread_start>>>start_thread"
 376 -0.0024667,"putspent>>>crossbeam_channel::channel::Receiver<T>::recv>>>std::sys_common::backtrace::__rust_begin_short_backtrace>>><core::panic::unwind_safe::AssertUnwindSafe<F> as core::ops::function::Fn
Once<()>>::call_once>>>std::panicking::try>>>std::panic::catch_unwind>>>core::ops::function::FnOnce::call_once{{vtable-shim}}>>>std::sys::unix::thread::Thread::new::thread_start>>>start_thread"
```

Conclusion

- ❑ Our study demonstrated the effectiveness of statistical debugging techniques in identifying performance issues.
- ❑ The results have significant implications for improving system call performance and can be applied to other systems.

Future Work & Questions

- ❑ Future work includes refining these techniques, advancing sequence pattern mining and exploring broader applications.
- ❑ We are looking forward to discussing and receiving use-cases from our industrial partners.
- ❑ Special thanks to CINEA for their sponsoring of this project.
- ❑ Any questions or feedback?

Mohammed Adib Khan
ak19qp@brocku.ca

Dr. Morteza Noferesti
mnoferesti@brocku.ca

Dr. Naser Ezzati-Jivan
nezzati@brocku.ca