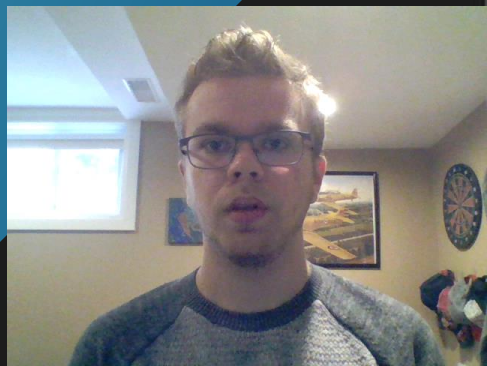
N-LANE BRIDGE PERFORMANCE ANTIPATTERN ANALYSIS USING SYSTEM-LEVEL EXECUTION TRACING

Riley VanDonge
Naser Ezzati-Jivan
Brock University



THE PROBLEM

- Multithreading can add latency to an application when used incorrectly
- The cause of this latency can be difficult to diagnose
  - Many possible causes: lock contention, resource pools, CPU preemption, etc.
  - Non-deterministic nature means that it is not trivial to follow the program's execution

BAD SMELLS

- Surface indication of a deeper problem within the system
- Closely related to static analysis: the bad smell must be directly identifiable in source code
- Common examples: duplicate code, feature envy
- Previously, we worked on detecting runtime smells using execution tracing
  - Examples: CPU Hog, Priority Inversion
  - https://github.com/riley-v/runtime-bad-smell-trace-metrics

ANTIPATTERNS

- Common problem bundled with detection methods and common solutions
- Used to better describe issues in a piece of software
- Antipattern analysis is performed during the testing phase of an application, before delivery
- One Lane Bridge: a performance antipattern
  - Only one, or a few, threads can execute at once at this place

EXISTING CHALLENGES

One Lane Bridge Performance Antipattern

- Bundles a common multithreading bottleneck problem with detection strategies and refactoring solutions
- Two issues/gaps in existing research:
  1. Do not consider bottlenecks in active resources
     - Active resource: performs a physical action in the real world (eg. CPU)
     - vs. passive resource: exists only virtually (eg. mutex)
  2. Use imprecise metrics
     - Example: overall CPU usage is used to denote application's CPU access

OUR SOLUTION

- Introduce N-Lane Bridge: new category of One Lane Bridge
  - Extends OLB from passive resources to active resources
  - Defines a method to distinguish an NLB in the target application from issues due to an external application
- Introduce detection method using system-level execution tracing to be performed during the testing phase
  - Allows for the gathering of more precise metrics
  - Eliminates the need for manual instrumentation of source code with tracepoints

WHAT IS THE N-LANE BRIDGE ANTIPATTERN?

Definition:

- A performance bottleneck due to the target application's use of an active resource.
- Cases where the bottleneck is due to an external application's use of an active resource are not considered NLB's.

Probable Causes:

- Incorrect configuration of the software.
- Implementation of the software did not account for the system it would be running on.

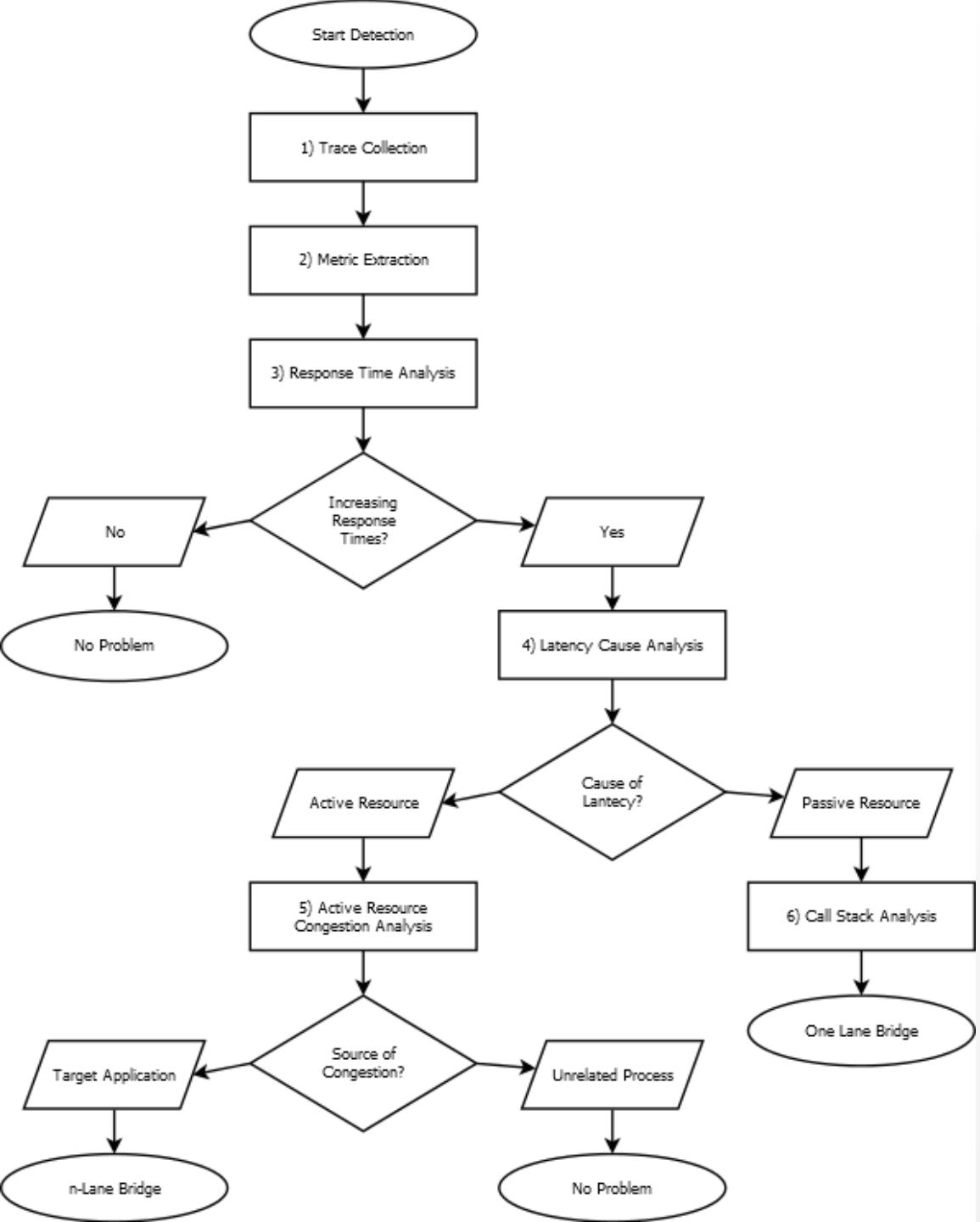
WHY FOCUS ON SYSTEM-LEVEL TRACING?

Benefits for Multithreading:

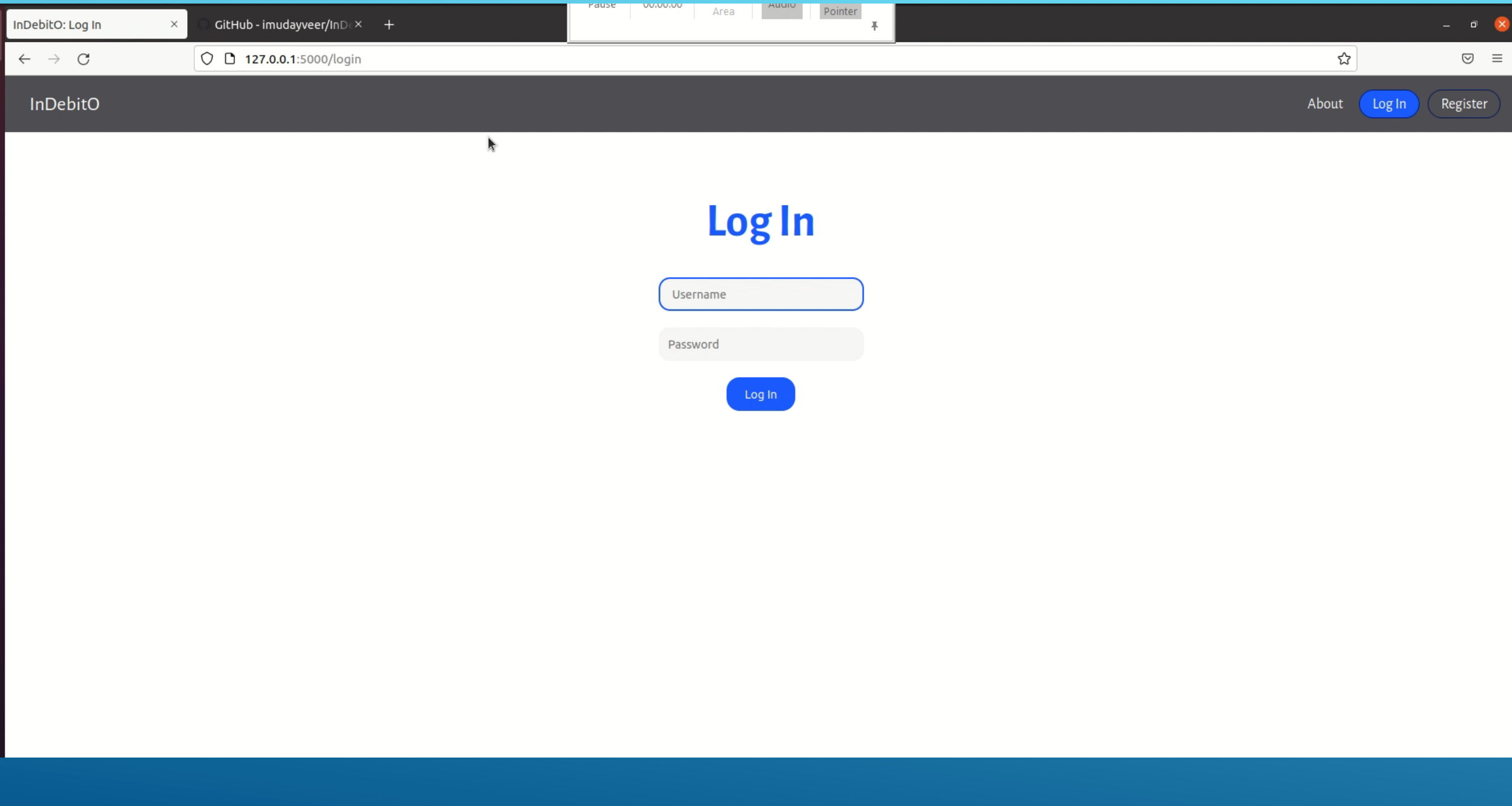
- Concurrency is difficult to predict. Dynamic analysis eliminates the false alarms from abstraction found in static analysis.
- System-level tracing provides the wide visibility of executing threads, processes, and resources needed to accurately analyze multithreaded applications.

Benefits for Ease of Use:

- The tracepoints are predefined (eg. Linux kernel) so no extra effort or domain knowledge is needed.



DETECTION STRATEGY


InDebitO: Log In
GitHub - imudayveer/InDe
Pause
00.00.00
Area
Audio
Pointer
127.0.0.1:5000/login
InDebitO
About
Log In
Register
Log In
Username
Password
Log In

TRACE COLLECTION

Stop Conditions:

1. The response time has increased 10x its original number
2. The application no longer supports adding more users

Required Tracepoints:

- Syscalls: reveal the reason for a blocked thread
  - Interrupt syscalls: irq_handler, softirq, hrtimer_expire
- Request delimiters: reveal the response time for a request
- sched_switch: indicate when a thread occupying a CPU is switched for the new thread
- sched_wakeup: indicate when a previously blocked thread becomes runnable

1ThreadTest.jmx (/home/riley/Documents/apache-jmeter-5.4.1/bin/1ThreadTest.jmx) - Apache JMeter (5.4.1)

File Edit Search Run Options Tools Help

00:00:00 0 0/0

- Test Plan
  - Thread Group
    - HTTP Request Defaults
    - Login
    - Backend Listener

HTTP Request

Name: Login

Comments:

Basic Advanced

Web Server

Protocol [http]: Server Name or IP: Port Number:

HTTP Request

POST Path: /login Content encoding:

Redirect Automatically | Follow Redirects | Use KeepAlive | Use multipart/form-data | Browser-compatible headers

Parameters Body Data Files Upload

Send Parameters With the Request:

Name:	Value	URL Encode?	Content-Type	Include Equals?
usrnm	user	☑	text/plain	☑
pwd	password	☑	text/plain	☑

Detail Add Add from Clipboard Delete Up Down

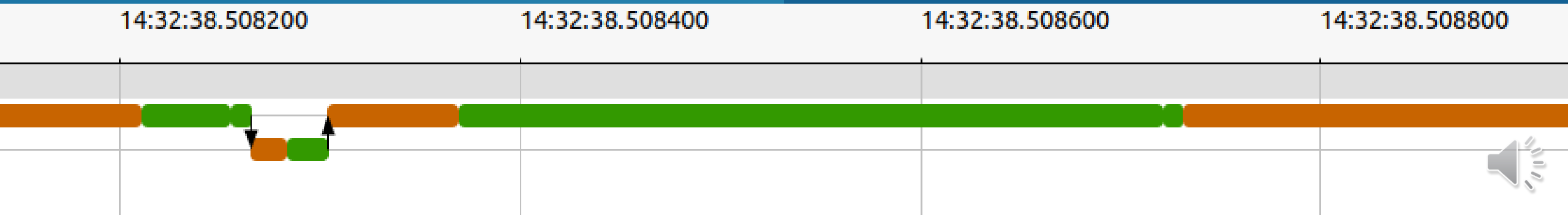
METRIC EXTRACTION

Metrics Needed:

- Response times of requests
- Critical blocking times for threads handling the requests, as well as their causes

Critical Path:

- Longest series of steps before completion
- Can be used to find critical blocking times and their causes

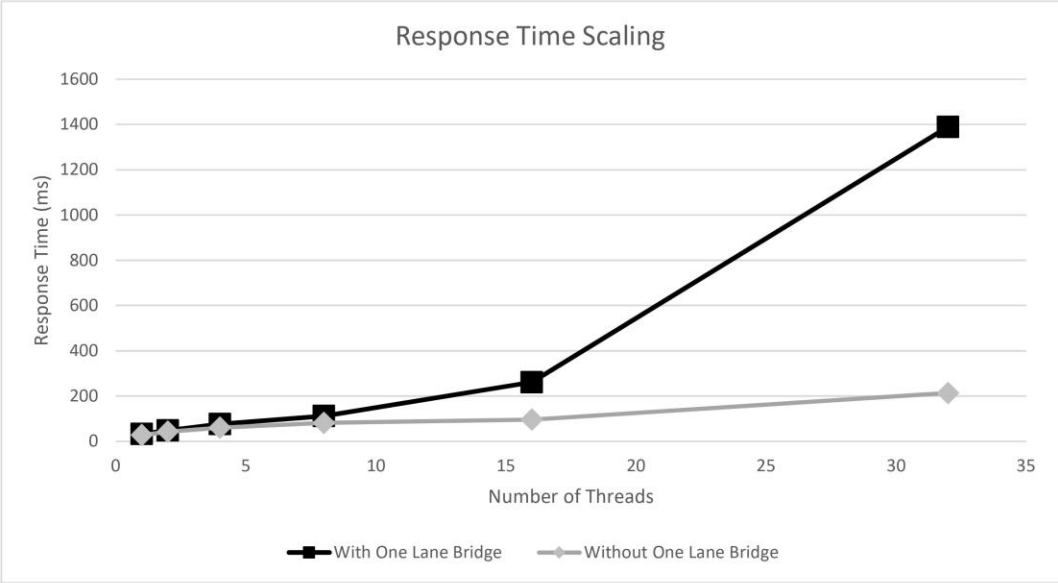


Open | 21.jtl ~/Documents/apache-jmeter-5.4.1/bin | Save

in.csv | 21.jtl

```
1642706038815,89,Login-2,200,OK,Thread Group 1-19,text,true,,3623,173,12,12,http://127.0.0.1:5000/login,89,0,0
1642706034954,3960,Login,200,OK,Thread Group 1-5,text,true,,4678,622,11,11,http://127.0.0.1:5000/login,3821,0,49
1642706034954,3821,Login-0,302,FOUND,Thread Group 1-5,text,true,,558,281,11,11,http://127.0.0.1:5000/login,3821,0,49
1642706038775,56,Login-1,302,FOUND,Thread Group 1-5,text,true,,497,168,11,11,http://127.0.0.1:5000/,56,0,0
1642706038831,83,Login-2,200,OK,Thread Group 1-5,text,true,,3623,173,11,11,http://127.0.0.1:5000/login,82,0,1
1642706034970,3950,Login,200,OK,Thread Group 1-13,text,true,,4678,622,10,10,http://127.0.0.1:5000/login,3841,0,45
1642706034970,3841,Login-0,302,FOUND,Thread Group 1-13,text,true,,558,281,10,10,http://127.0.0.1:5000/login,3841,0,45
1642706034974,3947,Login,200,OK,Thread Group 1-4,text,true,,4678,622,10,10,http://127.0.0.1:5000/login,3824,0,43
1642706034974,3824,Login-0,302,FOUND,Thread Group 1-4,text,true,,558,281,10,10,http://127.0.0.1:5000/login,3824,0,43
1642706038798,96,Login-1,302,FOUND,Thread Group 1-4,text,true,,497,168,10,10,http://127.0.0.1:5000/,96,0,1
1642706038894,27,Login-2,200,OK,Thread Group 1-4,text,true,,3623,173,10,10,http://127.0.0.1:5000/login,27,0,1
1642706034962,3961,Login,200,OK,Thread Group 1-9,text,true,,4678,622,9,9,http://127.0.0.1:5000/login,3835,0,57
1642706034962,3835,Login-0,302,FOUND,Thread Group 1-9,text,true,,558,281,9,9,http://127.0.0.1:5000/login,3835,0,57
1642706038797,95,Login-1,302,FOUND,Thread Group 1-9,text,true,,497,168,9,9,http://127.0.0.1:5000/,95,0,1
1642706038893,30,Login-2,200,OK,Thread Group 1-9,text,true,,3623,173,9,9,http://127.0.0.1:5000/login,30,0,2
1642706038811,85,Login-1,302,FOUND,Thread Group 1-13,text,true,,497,168,10,10,http://127.0.0.1:5000/,85,0,1
1642706038896,24,Login-2,200,OK,Thread Group 1-13,text,true,,3623,173,10,10,http://127.0.0.1:5000/login,24,0,1
1642706034964,3964,Login,200,OK,Thread Group 1-3,text,true,,4678,622,7,7,http://127.0.0.1:5000/login,3886,0,38
1642706034964,3887,Login-0,302,FOUND,Thread Group 1-3,text,true,,558,281,7,7,http://127.0.0.1:5000/login,3886,0,38
1642706034953,3975,Login,200,OK,Thread Group 1-2,text,true,,4678,622,7,7,http://127.0.0.1:5000/login,3899,0,64
1642706038851,52,Login-1,302,FOUND,Thread Group 1-3,text,true,,497,168,7,7,http://127.0.0.1:5000/,52,0,1
1642706034953,3900,Login-0,302,FOUND,Thread Group 1-2,text,true,,558,281,7,7,http://127.0.0.1:5000/login,3899,0,64
1642706038903,25,Login-2,200,OK,Thread Group 1-3,text,true,,3623,173,7,7,http://127.0.0.1:5000/login,25,0,1
1642706038853,54,Login-1,302,FOUND,Thread Group 1-2,text,true,,497,168,7,7,http://127.0.0.1:5000/,54,0,0
1642706038907,21,Login-2,200,OK,Thread Group 1-2,text,true,,3623,173,7,7,http://127.0.0.1:5000/login,21,0,1
1642706034970,3960,Login,200,OK,Thread Group 1-12,text,true,,4678,622,5,5,http://127.0.0.1:5000/login,3896,0,49
1642706034970,3896,Login-0,302,FOUND,Thread Group 1-12,text,true,,558,281,5,5,http://127.0.0.1:5000/login,3896,0,49
1642706038866,41,Login-1,302,FOUND,Thread Group 1-12,text,true,,497,168,5,5,http://127.0.0.1:5000/,41,0,0
1642706038907,23,Login-2,200,OK,Thread Group 1-12,text,true,,3623,173,5,5,http://127.0.0.1:5000/login,23,0,1
1642706034964,3976,Login,200,OK,Thread Group 1-6,text,true,,4678,622,4,4,http://127.0.0.1:5000/login,3913,0,56
1642706034964,3913,Login-0,302,FOUND,Thread Group 1-6,text,true,,558,281,4,4,http://127.0.0.1:5000/login,3913,0,56
1642706038877,35,Login-1,302,FOUND,Thread Group 1-6,text,true,,497,168,4,4,http://127.0.0.1:5000/,35,0,1
1642706038912,28,Login-2,200,OK,Thread Group 1-6,text,true,,3623,173,4,4,http://127.0.0.1:5000/login,28,0,0
1642706034974,3982,Login,200,OK,Thread Group 1-21,text,true,,4678,622,3,3,http://127.0.0.1:5000/login,3932,0,44
1642706034974,3932,Login-0,302,FOUND,Thread Group 1-21,text,true,,558,281,3,3,http://127.0.0.1:5000/login,3932,0,44
1642706038908,25,Login-1,302,FOUND,Thread Group 1-21,text,true,,497,168,3,3,http://127.0.0.1:5000/,25,0,0
1642706034953,4003,Login,200,OK,Thread Group 1-1,text,true,,4678,622,3,3,http://127.0.0.1:5000/login,3924,0,62
1642706038934,22,Login-2,200,OK,Thread Group 1-21,text,true,,3623,173,3,3,http://127.0.0.1:5000/login,22,0,0
1642706034953,3924,Login-0,302,FOUND,Thread Group 1-1,text,true,,558,281,3,3,http://127.0.0.1:5000/login,3924,0,62
1642706038877,36,Login-1,302,FOUND,Thread Group 1-1,text,true,,497,168,3,3,http://127.0.0.1:5000/,36,0,1
1642706038913,43,Login-2,200,OK,Thread Group 1-1,text,true,,3623,173,3,3,http://127.0.0.1:5000/login,43,0,1
1642706034953,4005,Login,200,OK,Thread Group 1-10,text,true,,4678,622,1,1,http://127.0.0.1:5000/login,3960,0,66
1642706034953,3960,Login-0,302,FOUND,Thread Group 1-10,text,true,,558,281,1,1,http://127.0.0.1:5000/login,3960,0,66
1642706038913,36,Login-1,302,FOUND,Thread Group 1-10,text,true,,497,168,1,1,http://127.0.0.1:5000/,35,0,1
1642706038949,9,Login-2,200,OK,Thread Group 1-10,text,true,,3623,173,1,1,http://127.0.0.1:5000/login,9,0,1
```

Plain Text | Tab Width: 8 | Ln 2, Col 20 | INS



RESPONSE TIME ANALYSIS

Motivation:

- Both categories of One Lane Bridge cause increasing response time latency as more users stress the system

Analysis Formula:

$\exists y \, \forall z \, (RT_z < RT_{z+1}) \rightarrow P$

Where:

- RT_z is the average response time for any execution z
- Any execution z has less users than execution z+1
- P holds that response times increase after execution y

Comparison of response times is done using a two-tailed t-test.

Open

in.csv
~/Documents/Scripts/One Lane Bridge Detection/RT Analysis

Save

```
1 Start,289
2 Start,821,1718,1719,1735,1726,1736
3 Start,2130,2132,2148,2184,2180,2169,2176,2176,2159,2126,2226
4 Start,1756,2695,2740,2760,2862,2897,2955,2959,3000,3015,3023,3039,3055,3067,3082,3087
5 Start,2971,3137,3549,3604,3636,3838,3850,3864,3908,3951,3960,3950,3947,3961,3964,3975,3960,3976,3982,4003,4005
```

CSV Tab Width: 8 Ln 5, Col 111 INS

LATENCY CAUSE ANALYSIS

Average Blocking Time Formula:

$$AveBlockTime(Res_x) = \sum_{i=y}^{n-1} (BTx_{\,i+1} - BT_{x\,i})$$

Where:

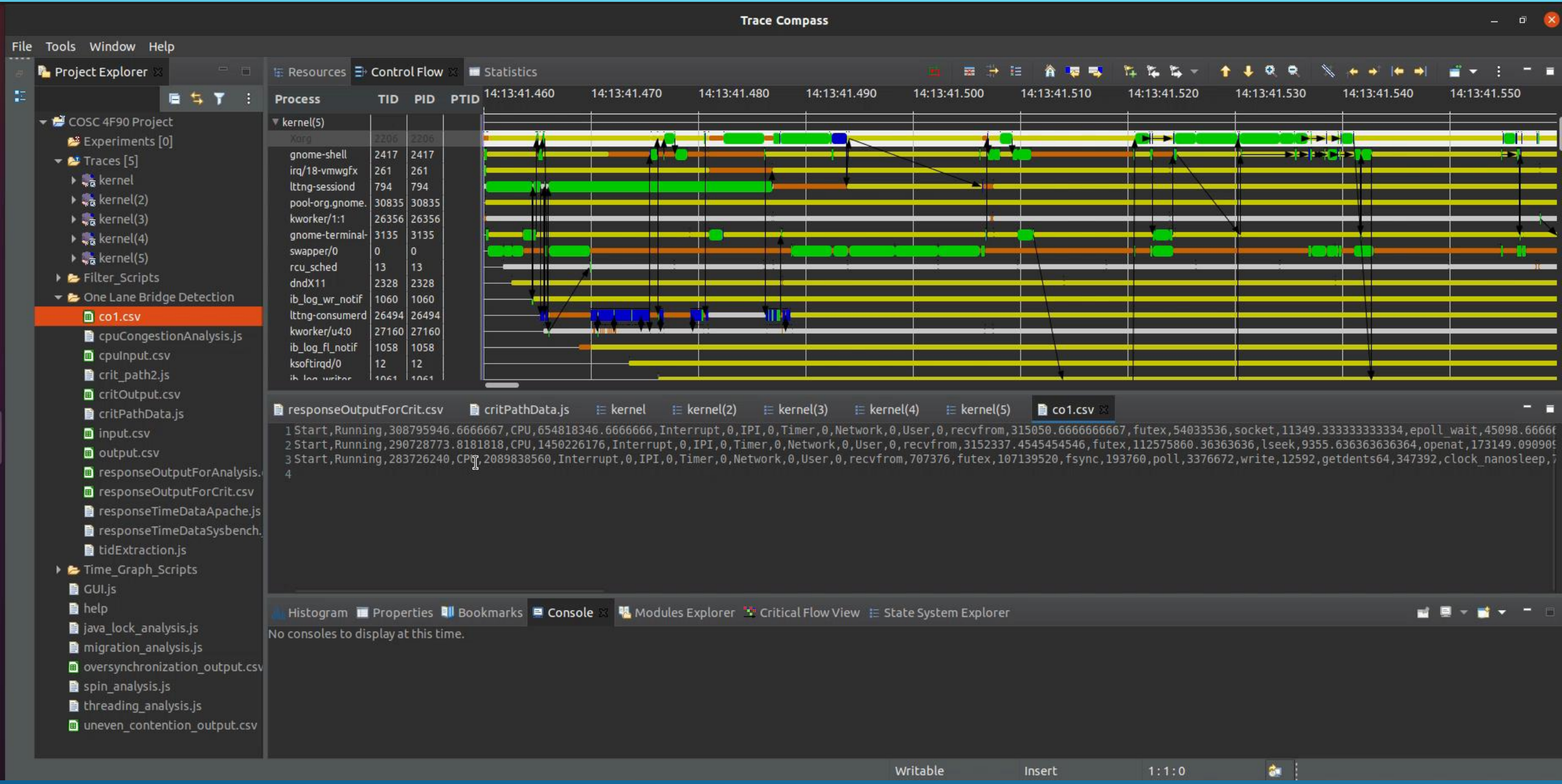
- Res_x is a resource which blocks the target threads
- $BT_{x\,i}$ is the blocking time for resource x in execution I

Analysis Formula:

$$\exists Res_x \; \forall Res_z \; (AveBlockTime(Res_x) > AveBlockTime(Res_z)) \rightarrow Q$$

Where:

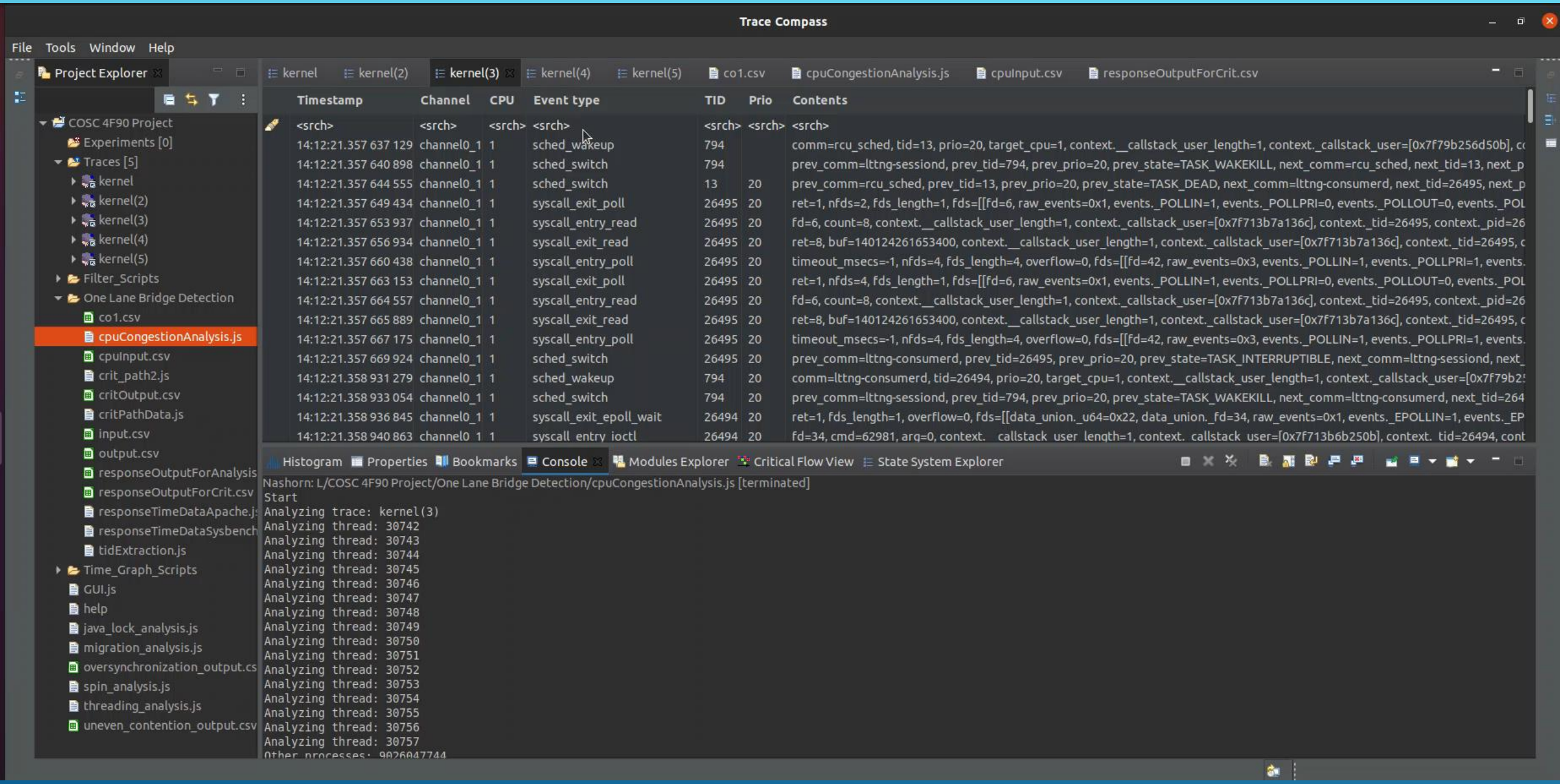
- Q holds that Res_x is the resource which contributes the most to the latency


Trace Compass
File
Tools
Window
Help
Project Explorer
COSC 4F90 Project
Experiments [0]
Traces [5]
kernel
kernel(2)
kernel(3)
kernel(4)
kernel(5)
Filter_Scripts
One Lane Bridge Detection
co1.csv
cpuCongestionAnalysis.js
cpuInput.csv
crit_path2.js
critOutput.csv
critPathData.js
input.csv
output.csv
responseOutputForAnalysis.
responseOutputForCrit.csv
responseTimeDataApache.js
responseTimeDataSysbench.
tidExtraction.js
Time_Graph_Scripts
GUI.js
help
java_lock_analysis.js
migration_analysis.js
oversynchronization_output.csv
spin_analysis.js
threading_analysis.js
uneven_contention_output.csv
Resources
Control Flow
Statistics
Process
TID
PID
PTID
kernel(5)
Xorg
gnome-shell
irq/18-vmwgfx
lttng-sessiond
pool-org.gnome.
kworker/1:1
gnome-terminal-
swapper/0
rcu_sched
dndX11
ib_log_wr_notif
lttng-consumerd
kworker/u4:0
ib_log_fl_notif
ksoftirqd/0
14:13:41.460
14:13:41.470
14:13:41.480
14:13:41.490
14:13:41.500
14:13:41.510
14:13:41.520
14:13:41.530
14:13:41.540
14:13:41.550
responseOutputForCrit.csv
critPathData.js
kernel
kernel(2)
kernel(3)
kernel(4)
kernel(5)
co1.csv
1 Start,Running,308795946.6666667,CPU,654818346.6666666,Interrupt,0,IPI,0,Timer,0,Network,0,User,0,recvfrom,315050.6666666667,futex,54033536,socket,11349.333333333334,epoll_wait,45098.6666
2 Start,Running,290728773.8181818,CPU,1450226176,Interrupt,0,IPI,0,Timer,0,Network,0,User,0,recvfrom,3152337.4545454546,futex,112575860.36363636,lseek,9355.636363636364,openat,173149.090909
3 Start,Running,283726240,CPU,2089838560,Interrupt,0,IPI,0,Timer,0,Network,0,User,0,recvfrom,707376,futex,107139520,fsync,193760,poll,3376672,write,12592,getdents64,347392,clock_nanosleep,
4
Histogram
Properties
Bookmarks
Console
Modules Explorer
Critical Flow View
State System Explorer
No consoles to display at this time.
Writable
Insert
1 : 1 : 0

ACTIVE RESOURCE CONGESTION ANALYSIS

Check which threads are using the congested resource during the analysis period:

1. If over a specific threshold (e.g. 50%) of time, the target application holds the resource: N-Lane Bridge
2. Otherwise: No detected problem in the target application


Trace Compass
File Tools Window Help
Project Explorer
COSC 4F90 Project
Experiments [0]
Traces [5]
kernel
kernel(2)
kernel(3)
kernel(4)
kernel(5)
Filter_Scripts
One Lane Bridge Detection
co1.csv
cpuCongestionAnalysis.js
cpuInput.csv
crit_path2.js
critOutput.csv
critPathData.js
input.csv
output.csv
responseOutputForAnalysis
responseOutputForCrit.csv
responseTimeDataApache.j
responseTimeDataSysbench
tidExtraction.js
Time_Graph_Scripts
GUI.js
help
java_lock_analysis.js
migration_analysis.js
oversynchronization_output.cs
spin_analysis.js
threading_analysis.js
uneven_contention_output.csv
kernel
kernel(2)
kernel(3)
kernel(4)
kernel(5)
co1.csv
cpuCongestionAnalysis.js
cpuInput.csv
responseOutputForCrit.csv
Timestamp Channel CPU Event type TID Prio Contents
<srch> <srch> <srch> <srch> <srch> <srch> <srch>
14:12:21.357 637 129 channel0_1 1 sched_wakeup 794 comm=rcu_sched, tid=13, prio=20, target_cpu=1, context.__callstack_user_length=1, context._callstack_user=[0x7f79b256d50b], c
14:12:21.357 640 898 channel0_1 1 sched_switch 794 prev_comm=lttng-sessiond, prev_tid=794, prev_prio=20, prev_state=TASK_WAKEKILL, next_comm=rcu_sched, next_tid=13, next_p
14:12:21.357 644 555 channel0_1 1 sched_switch 13 20 prev_comm=rcu_sched, prev_tid=13, prev_prio=20, prev_state=TASK_DEAD, next_comm=lttng-consumerd, next_tid=26495, next_p
14:12:21.357 649 434 channel0_1 1 syscall_exit_poll 26495 20 ret=1, nfds=2, fds_length=1, fds=[[fd=6, raw_events=0x1, events._POLLIN=1, events._POLLPRI=0, events._POLLOUT=0, events._POL
14:12:21.357 653 937 channel0_1 1 syscall_entry_read 26495 20 fd=6, count=8, context.__callstack_user_length=1, context._callstack_user=[0x7f713b7a136c], context._tid=26495, context._pid=26
14:12:21.357 656 934 channel0_1 1 syscall_exit_read 26495 20 ret=8, buf=140124261653400, context.__callstack_user_length=1, context._callstack_user=[0x7f713b7a136c], context._tid=26495, c
14:12:21.357 660 438 channel0_1 1 syscall_entry_poll 26495 20 timeout_msecs=-1, nfds=4, fds_length=4, overflow=0, fds=[[fd=42, raw_events=0x3, events._POLLIN=1, events._POLLPRI=1, events.
14:12:21.357 663 153 channel0_1 1 syscall_exit_poll 26495 20 ret=1, nfds=4, fds_length=1, fds=[[fd=6, raw_events=0x1, events._POLLIN=1, events._POLLPRI=0, events._POLLOUT=0, events._POL
14:12:21.357 664 557 channel0_1 1 syscall_entry_read 26495 20 fd=6, count=8, context.__callstack_user_length=1, context._callstack_user=[0x7f713b7a136c], context._tid=26495, context._pid=26
14:12:21.357 665 889 channel0_1 1 syscall_exit_read 26495 20 ret=8, buf=140124261653400, context.__callstack_user_length=1, context._callstack_user=[0x7f713b7a136c], context._tid=26495, c
14:12:21.357 667 175 channel0_1 1 syscall_entry_poll 26495 20 timeout_msecs=-1, nfds=4, fds_length=4, overflow=0, fds=[[fd=42, raw_events=0x3, events._POLLIN=1, events._POLLPRI=1, events.
14:12:21.357 669 924 channel0_1 1 sched_switch 26495 20 prev_comm=lttng-consumerd, prev_tid=26495, prev_prio=20, prev_state=TASK_INTERRUPTIBLE, next_comm=lttng-sessiond, next_
14:12:21.358 931 279 channel0_1 1 sched_wakeup 794 20 comm=lttng-consumerd, tid=26494, prio=20, target_cpu=1, context.__callstack_user_length=1, context._callstack_user=[0x7f79b25
14:12:21.358 933 054 channel0_1 1 sched_switch 794 20 prev_comm=lttng-sessiond, prev_tid=794, prev_prio=20, prev_state=TASK_WAKEKILL, next_comm=lttng-consumerd, next_tid=264
14:12:21.358 936 845 channel0_1 1 syscall_exit_epoll_wait 26494 20 ret=1, fds_length=1, overflow=0, fds=[[data_union._u64=0x22, data_union._fd=34, raw_events=0x1, events._EPOLLIN=1, events._EP
14:12:21.358 940 863 channel0_1 1 syscall_entry_ioctl 26494 20 fd=34, cmd=62981, arg=0, context.__callstack_user_length=1, context._callstack_user=[0x7f713b6b250b], context._tid=26494, cont
Histogram Properties Bookmarks Console Modules Explorer Critical Flow View State System Explorer
Nashorn: L/COSC 4F90 Project/One Lane Bridge Detection/cpuCongestionAnalysis.js [terminated]
Start
Analyzing trace: kernel(3)
Analyzing thread: 30742
Analyzing thread: 30743
Analyzing thread: 30744
Analyzing thread: 30745
Analyzing thread: 30746
Analyzing thread: 30747
Analyzing thread: 30748
Analyzing thread: 30749
Analyzing thread: 30750
Analyzing thread: 30751
Analyzing thread: 30752
Analyzing thread: 30753
Analyzing thread: 30754
Analyzing thread: 30755
Analyzing thread: 30756
Analyzing thread: 30757
Other processes: 9026047744

CALL STACK ANALYSIS

Address translation formula:

$$TracepointAddress - BaseAddress = ELFSymbolTableEntry$$

```
0000000000004040 B lock
000000000000134a T main
                 U printf@@GLIBC_2.2.5
                 U pthread_create@@GLIBC_2.2.5
                 U pthread_join@@GLIBC_2.2.5
                 U pthread_mutex_destroy@@GLIBC_2.2.5
                 U pthread_mutex_init@@GLIBC_2.2.5
                 U pthread_mutex_lock@@GLIBC_2.2.5
                 U pthread_mutex_unlock@@GLIBC_2.2.5
                 U puts@@GLIBC_2.2.5
0000000000001200 t register_tm_clones
00000000000011a0 T _start
                 U strerror@@GLIBC_2.2.5
0000000000004070 B tid
0000000000004010 D __TMC_END__
0000000000001289 T trythis
```

InDebitO: Transactions

127.0.0.1:5000

InDebitO

Income Expense Search Profile About Log Out

Welcome Back!!!

Transactions

Transaction Type	Source	Category	Shop or Person	Remarks	Date and Time	Amount
					TOTAL INCOME	₹0.00
					TOTAL EXPENSE	₹0.00

Distribution of Expense acording to Category

No data

Distribution of Expense acording to Source

We have written a paper on this subject and submitted it to the 2022 ICPC conference.

Future Work:

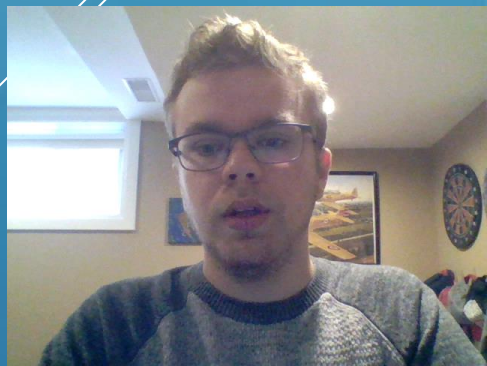
- Improvement: the metric extraction algorithm takes a significant amount of time. This is due to the number of critical paths that must be formed and examined individually. An improvement would be to form and examine the critical paths in batch jobs, reducing the number of times the trace must be iterated over.
- Addition: more performance antipatterns could be detected using system-level tracing. Antipatterns similar to the One Lane Bridge include Traffic Jam and The Ramp.

CONCLUSIONS AND FUTURE WORK

Email: rv18jq@brocku.ca

GitHub Repo: https://github.com/riley-v/n-lane-bridge-antipattern-analysis

THANK YOU



COST AWARE TRACING

AMIR HAGHSHENAS

MICHEL DAGENAIS

NASER EZZATI

PROGRESS REPORT 2022

CONTENT

Problem statement

Literature review

Methodology

Future work

References

COST AWARE TRACING

- Tracing generates large amount of data in short time
- It is not always possible to trace and save everything
  - IoT devices
  - Embedded systems
- A solution is required to choose how to collect as much as possible
  - Specially in systems with limited resources
- The solution should consider the observer effect of tracing on program
  - Execution delay
  - Memory consumption

COST AWARE TRACING

- Define a budget (how much overhead is acceptable)
- How to reduce the cost
  - Sampling
  - Based on cost
  - Keep the most important

ANALYZING TRACING OBSERVER EFFECT

Mohamad Gebai and Michel Dagenais (2018) [1]

A survey on kernel and user space tracing and overhead

Calculated execution latency of a single trace point using kernel modules

Indigo Orton and Alan Mycroft (2021) [2]

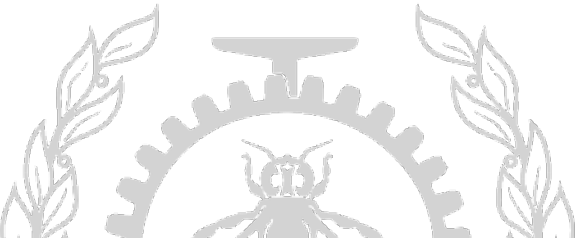
Analyzed effect of tracing on program concurrent behavior

Used configurable overhead tracer to add overhead on memory and cup usage

Applied a concurrent performance analysis

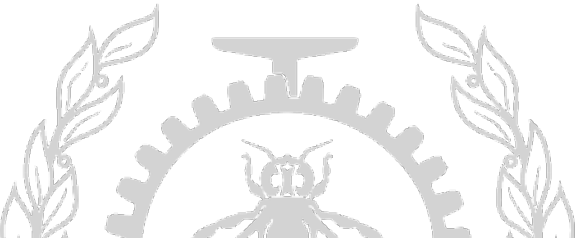
USING COST FUNCTION TO MODIFY TRACING

- Fischmeister and Lam (2010) [3]
- Introduced time-aware instrumentation
- Cost functions are execution time and instrumentation coverage
- Starts with source code analysis and naïve instrumentation
- Check execution time
  - If violated, use integer linear programming to reduce coverage to meet budget.



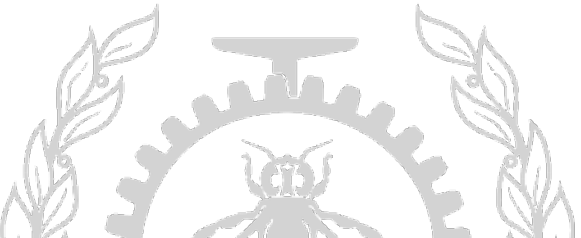
USING COST FUNCTION TO MODIFY TRACING

- Kashif and Arafa (2013) [4]
- Static instrumentation model called INSTEP
- Support up to four extra-functional properties
  - Instrumentation intent values
  - Code size, execution time and detection latency
- Each cost model is created using an automata
- provided insight into pruning the search space of instrumentation alternatives to find a feasible instrumentation solution.



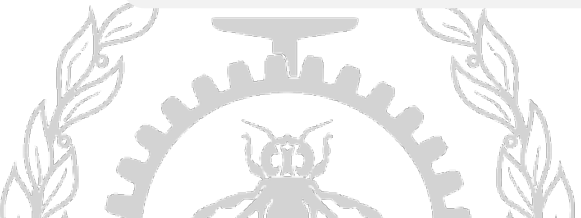
USING COST FUNCTION TO MODIFY TRACING

- Arafa and Kashif (2013) [5]
- DIME: Time-aware dynamic binary instrumentation
- Keep two version of the program.
  - One is instrumented and the other one is not instrumented
- User should provide two elements
  - P: Instrumentation period
  - B: Instrumentation budget (B<P)
- During each period, program is instrumented only for B time unit and then switch to the not instrumented version.



PROJECT OBJECTIVE

1. Define a budget for tracing overhead and monitor the system in real time.
2. Define a model for cost and effectiveness of each enabled trace point
3. Optimize the cost and effectiveness of the trace using an optimization problem



CURRENT STATE OF PROJECT

- Define a cost function to analysis
  - Execution delay
- Measurement
  - Developing kernel modules to record trace points execution time
  - Recording kernel counters using Perf_events
- Visualization
  - Working on trace compass development environment
  - Working on Histogram section to show the execution delay instead of event count

NEXT STEP

Change Histogram section to Cost analysis

→

Define a model for tracing effectiveness

↓

Define a model for time budget (static or dynamic)

→

Develop an optimization model to minimize the cost and maximize the effectiveness

REFERENCES

- Gebai, M., & Dagenais, M. R. (2018). Survey and analysis of kernel and userspace tracers on linux: Design, implementation, and overhead. *ACM Computing Surveys (CSUR)*, *51*(2), 1-33.
- Orton, I., & Mycroft, A. (2021, September). Tracing and its observer effect on concurrency. In *Proceedings of the 18th ACM SIGPLAN International Conference on Managed Programming Languages and Runtimes* (pp. 88-96).
- S. Fischmeister and P. Lam. Time-AwareInstrumentation of Embedded Software.IEEETransactions on Industrial Informatics, 6(4):652–663,Nov 2010.
- H. Kashif, P. Arafa, and S. Fischmeister. INSTEP: AStatic Instrumentation Framework for PreservingExtra-Functional Properties. InIEEE 19thInternational Conference on Embedded and Real-TimeComputing Systems and Applications, RTCSA'13,pages 257–266, Aug 2013.
- P. Arafa, H. Kashif, and S. Fischmeister. DIME: Time-aware Dynamic Binary Instrumentation UsingRate-based Resource Allocation. InProceedings of theEleventh ACM International Conference on EmbeddedSoftware, EMSOFT'13, pages 25:1–25:10. ACM, 2013

THANK YOU

Adaptive Tracing Based On Time Series Trend Detection

Mohammed Adib Khan

Supervisor: Naser Ezzati-Jivan

Department of Computer Science

Brock University

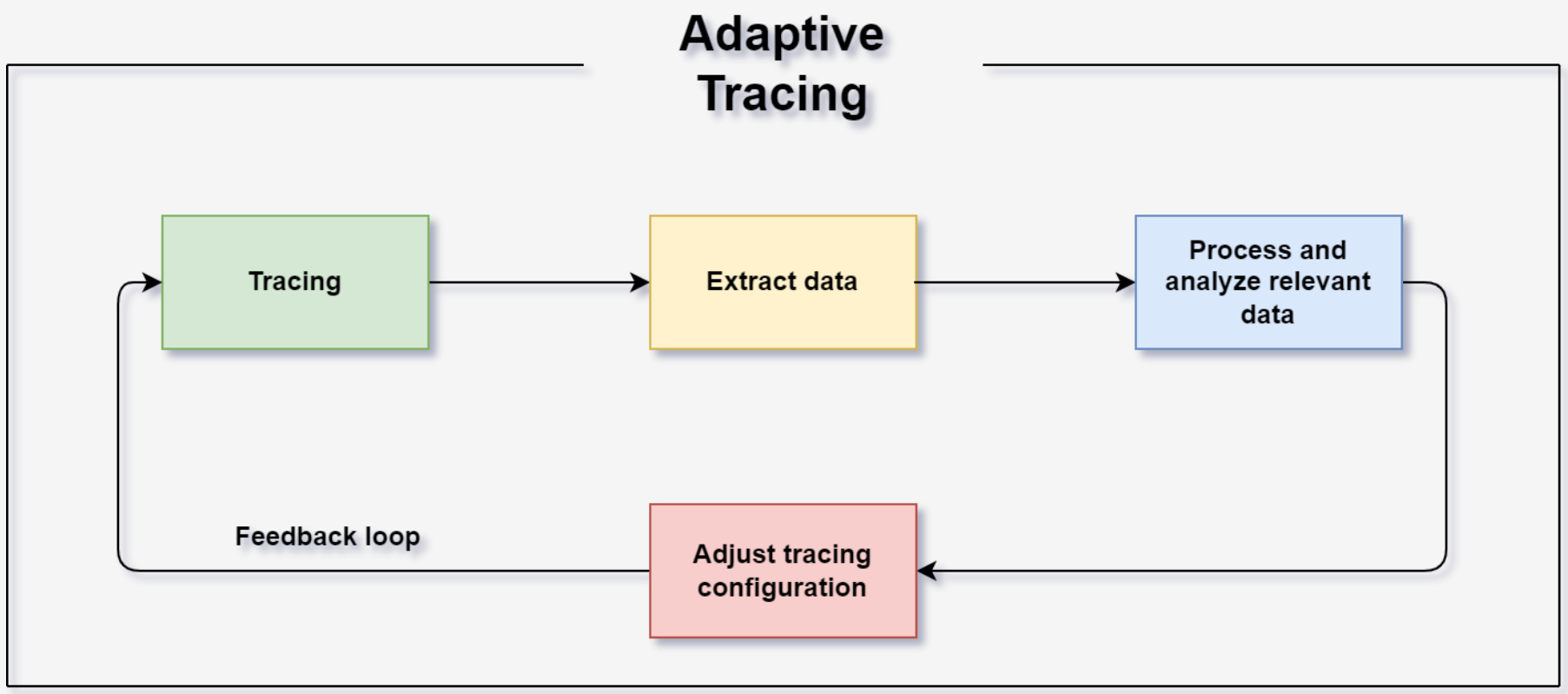
Challenges with execution tracing

- Huge size and lots of noise and irrelevant data.
  - High overhead (observer effect).
- To many instruments: a typical Linux kernel has over 300 instruments.
  - Its not easy to predict where system issues may arise and only conveniently keep those related instruments enabled.
  - Requires manual tuning of tracepoints and instruments to make adjustments, which can be tedious.
- Pre-determined set of instruments might miss out on important tracing data.

Tracing methods

- Static tracing – works by using static instrumentation.
- Dynamic tracing – tracepoints are injected into a running system.
- **Adaptive tracing** – tracing instruments/tracepoints are controlled automatically by the framework itself to figure out points of interest.

What is Adaptive Tracing?

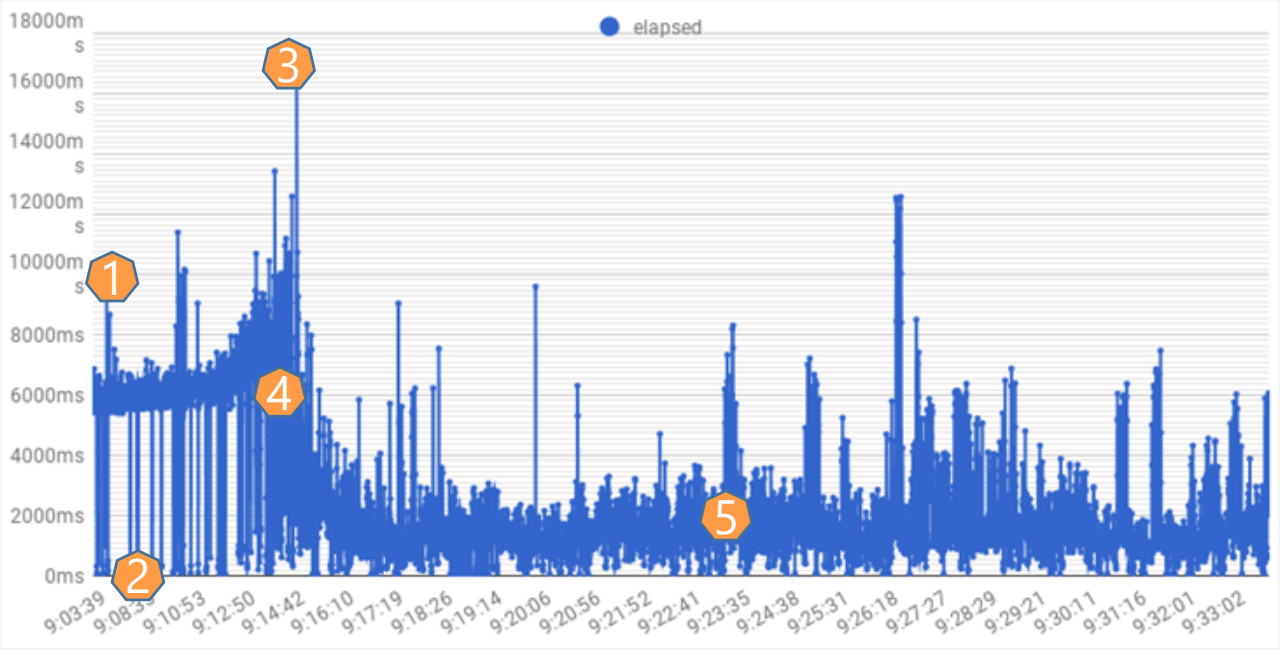


Some related works

- **System Execution Path Profiling[1], LogMine [2], Pythia[3]:**
  - Using Coefficient of Variance (CoV) or any standard deviation methods on grouped data is more biased towards shorter requests.
  - Clustering methods would always prioritize high standard deviation groups, where as there could be underlying consistently worse performing offenders going undetected.
  - Anomaly based adaptive tracing would trigger events which are not always necessary.
- **ADRL[4], DRAVM for Cloud Computing Environment[5], Workload Prediction Using ARIMA for Cloud Applications' QoS[6]:**
  - Unlike cloud computing resource allocation via reinforcement learning methods, they are not feasible in tracing due to the lack of options to verify the actions right away.
- **Process Monitoring on Sequences of System Call Count Vectors[7], Adaptive Performance Anomaly Detection in Distributed Systems Using Online SVMs[8]:**
  - Trend based anomaly detection can't adapt to increasing or decreasing of the baseline resource usage.

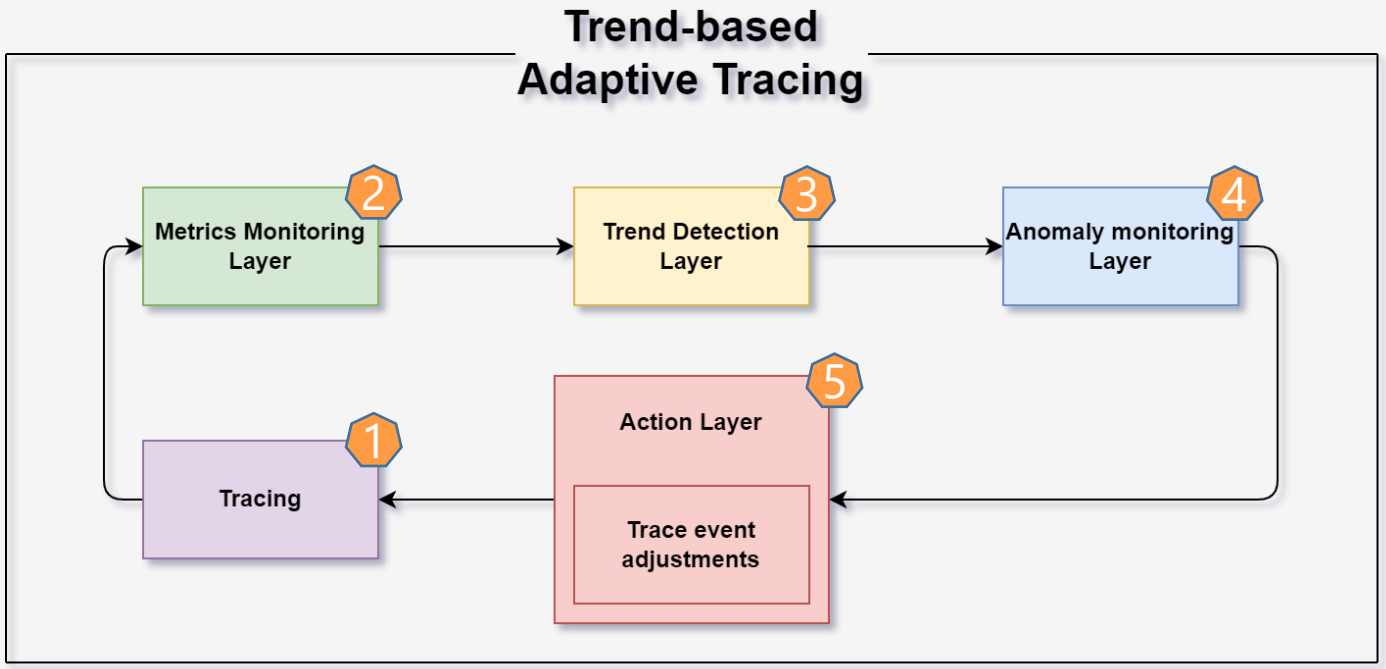
Research questions

- **How do we decide to change the tracing config?**
  - Should we adjust trace config whenever there is an anomaly/change?
- What are the possible actions? How we are going to change events?
- How do we evaluate the trace adjustments?

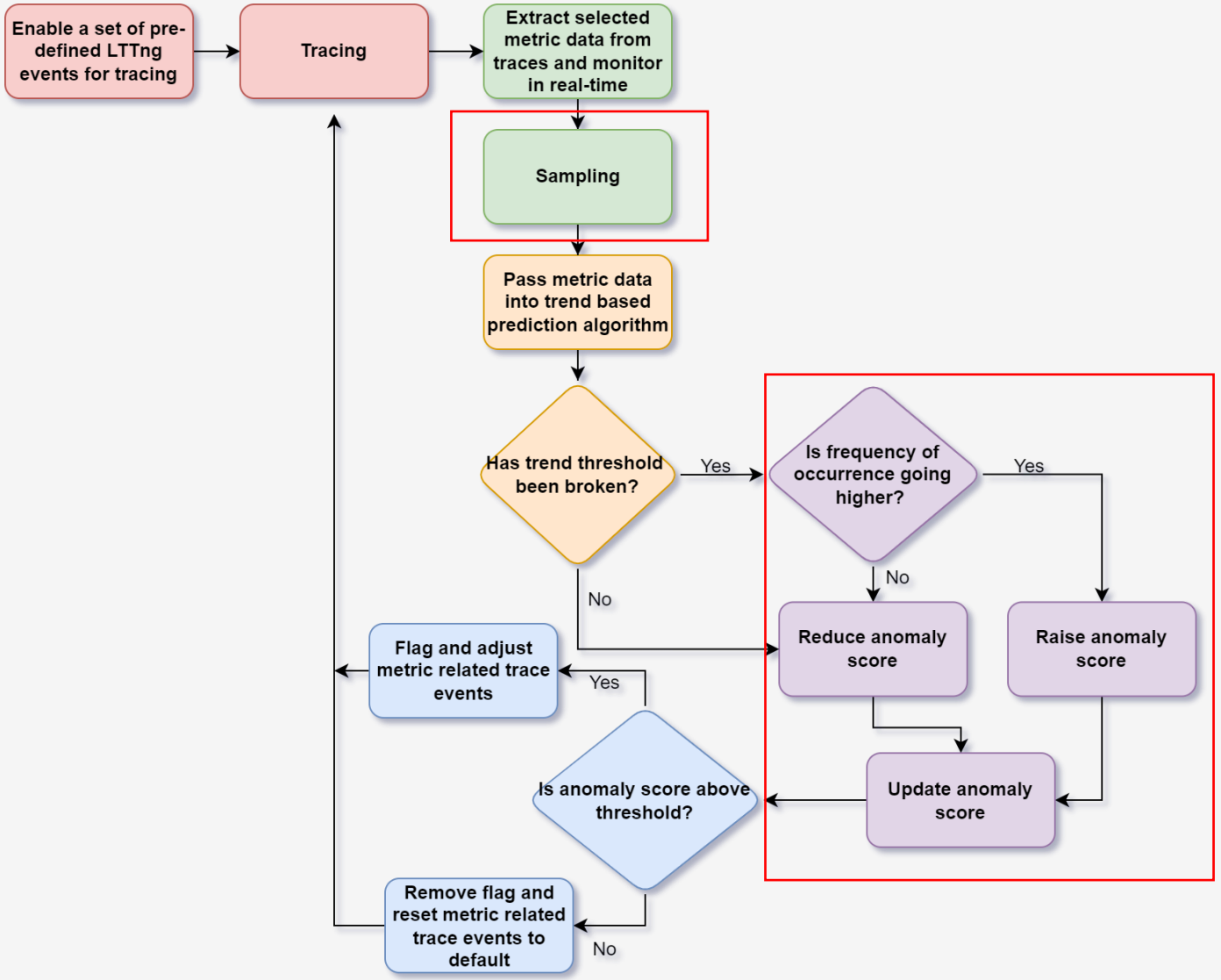


Proposed method

- **Architecture / Structure:**
  - Tracing
    - Kernel-level tracing with LTTng
  - Metrics Monitoring Layer
    - cpu_utilization, disk_utilization, etc.
    - Total of 23 metrics.
  - Trend Detection Layer
    - Time series ARIMA, EMA, DEMA, etc.
    - Financial market trend prediction
  - Anomaly Monitoring Layer
    - Maintains anomaly score
    - Adaptive threshold
  - Action layer
    - Controls tracing events



TAT method flowchart



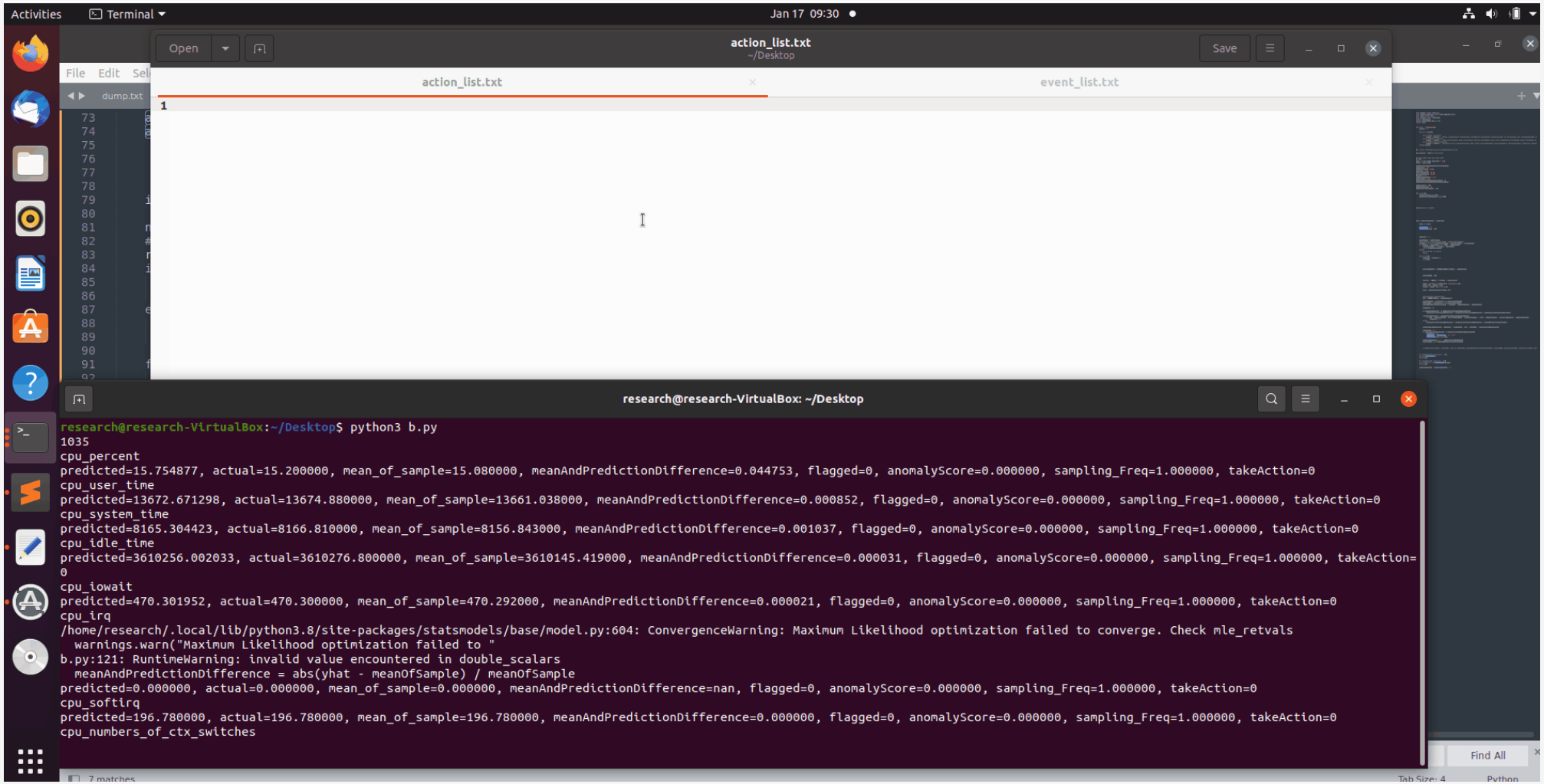
Proposed method

- Metrics monitoring layer:
  - LTTng: Trace data collection
  - Babeltrace: Metrics data extraction from trace files.
- Trend detection layer:
  - Sampling:
    - Take sample of fixed size dataset at random intervals.
    - Control frequency of intervals based on anomaly scores.
  - Use prediction model (ARIMA) to predict X+1 step.
  - If mean of sample vs prediction is above or below anomaly threshold, then flag it as anomaly.

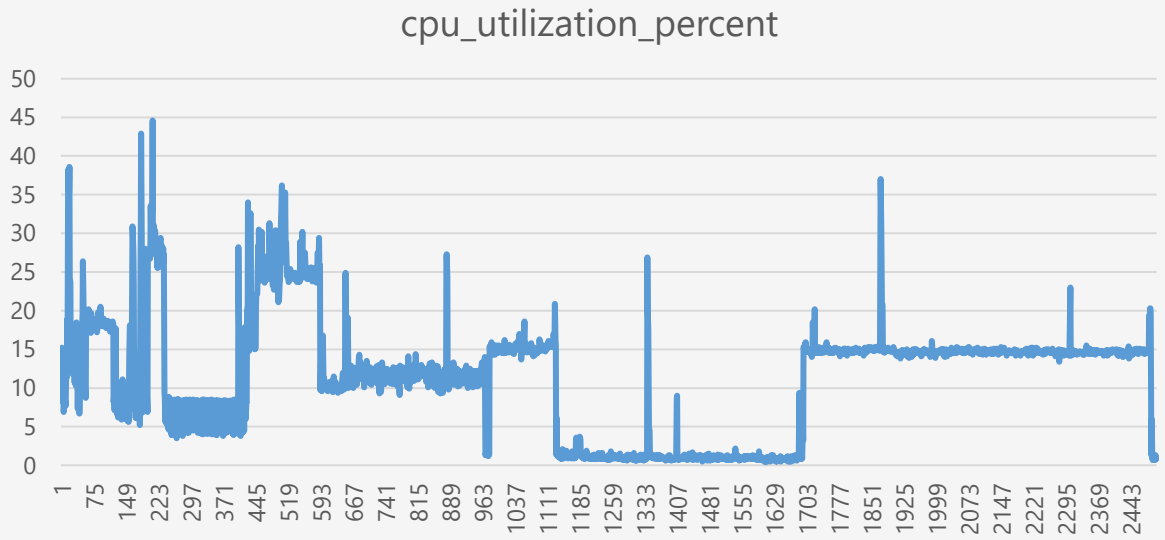
Proposed method

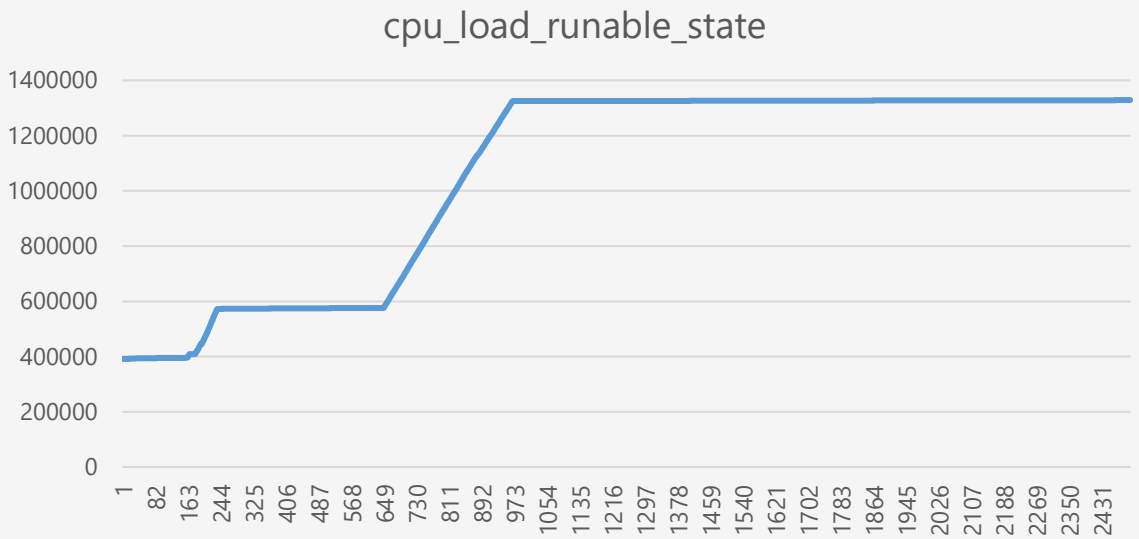
- Anomaly monitoring layer:
  - `anomalyScore = (betaVal * isAnomaly) + ((1 - betaVal) * anomalyScore)`
  - Auto adjust betaVal to increase or decrease anomalyScore based on *frequency of the anomaly occurrence*.
- Action layer:
  - Group LTTng events based on the list of relevant monitoring metrics.
  - List problematic metrics and enable related LTTng events.
  - Monitor changes in anomaly list to disable back events which are not required.

Demo


Activities
Terminal
Jan 17 09:30
Open
action_list.txt
~/Desktop
Save
File Edit Sel
action_list.txt
event_list.txt
dump.txt
research@research-VirtualBox: ~/Desktop
research@research-VirtualBox:~/Desktop$ python3 b.py
1035
cpu_percent
predicted=15.754877, actual=15.200000, mean_of_sample=15.080000, meanAndPredictionDifference=0.044753, flagged=0, anomalyScore=0.000000, sampling_Freq=1.000000, takeAction=0
cpu_user_time
predicted=13672.671298, actual=13674.880000, mean_of_sample=13661.038000, meanAndPredictionDifference=0.000852, flagged=0, anomalyScore=0.000000, sampling_Freq=1.000000, takeAction=0
cpu_system_time
predicted=8165.304423, actual=8166.810000, mean_of_sample=8156.843000, meanAndPredictionDifference=0.001037, flagged=0, anomalyScore=0.000000, sampling_Freq=1.000000, takeAction=0
cpu_idle_time
predicted=3610256.002033, actual=3610276.800000, mean_of_sample=3610145.419000, meanAndPredictionDifference=0.000031, flagged=0, anomalyScore=0.000000, sampling_Freq=1.000000, takeAction=0
cpu_iowait
predicted=470.301952, actual=470.300000, mean_of_sample=470.292000, meanAndPredictionDifference=0.000021, flagged=0, anomalyScore=0.000000, sampling_Freq=1.000000, takeAction=0
cpu_irq
/home/research/.local/lib/python3.8/site-packages/statsmodels/base/model.py:604: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check mle_retvals
warnings.warn("Maximum Likelihood optimization failed to "
b.py:121: RuntimeWarning: invalid value encountered in double_scalars
meanAndPredictionDifference = abs(yhat - meanOfSample) / meanOfSample
predicted=0.000000, actual=0.000000, mean_of_sample=0.000000, meanAndPredictionDifference=nan, flagged=0, anomalyScore=0.000000, sampling_Freq=1.000000, takeAction=0
cpu_softirq
predicted=196.780000, actual=196.780000, mean_of_sample=196.780000, meanAndPredictionDifference=0.000000, flagged=0, anomalyScore=0.000000, sampling_Freq=1.000000, takeAction=0
cpu_numbers_of_ctx_switches
Find All
7 matches
Tab Size: 4
Python

Example analysis





Conclusion & future work

- Binning of metrics data so that all metrics don’t have to be processed through the trend detection layer all the time.
- Implement better method for a dynamic betaValue for the anomaly score function.
- Needs better correlation groupings of events and metrics list.
- Implement better methods to solve conflicts/overlap between events list.
- Rigorous testing.

Selected references

1. Giraldeau, Francis, et al. “System Execution Path Profiling Using Hardware Performance Counters.” *2021 IEEE International Systems Conference (SysCon)*, 2021. *Crossref*, https://doi.org/10.1109/syscon48628.2021.9447121.

2. Hamooni, Hossein, et al. “LogMine.” *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, 2016. *Crossref*, https://doi.org/10.1145/2983323.2983358.

3. Ates, Emre, et al. “An Automated, Cross-Layer Instrumentation Framework for Diagnosing Performance Problems in Distributed Applications.” *Proceedings of the ACM Symposium on Cloud Computing*, 2019. *Crossref*, https://doi.org/10.1145/3357223.3362704.

4. Kardani-Moghaddam, Sara, et al. “ADRL: A Hybrid Anomaly-Aware Deep Reinforcement Learning-Based Resource Scaling in Clouds.” *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 3, 2021, pp. 514–26. *Crossref*, https://doi.org/10.1109/tpds.2020.3025914.

5. Xiao, Zhen, et al. “Dynamic Resource Allocation Using Virtual Machines for Cloud Computing Environment.” *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, 2013, pp. 1107–17. *Crossref*, https://doi.org/10.1109/tpds.2012.283.

Selected references

6. Calheiros, Rodrigo N., et al. “Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications’ QoS.” *IEEE Transactions on Cloud Computing*, vol. 3, no. 4, 2015, pp. 449–58. *Crossref*, https://doi.org/10.1109/tcc.2014.2350475.

7. Dymshits, Michael. “Process Monitoring on Sequences of System Call Count Vectors.” *ArXiv.Org*, 12 July 2017, arxiv.org/abs/1707.03821.

8. Alvarez Cid-Fuentes, Javier, et al. “Adaptive Performance Anomaly Detection in Distributed Systems Using Online SVMs.” *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 5, 2020, pp. 928–41. *Crossref*, https://doi.org/10.1109/tdsc.2018.2821693.

9. Alkasem, Ameen, et al. “Cloud Computing: A Model Construct of Real-Time Monitoring for Big Dataset Analytics Using Apache Spark.” *Journal of Physics: Conference Series*, vol. 933, 2018, p. 012018. *Crossref*, https://doi.org/10.1088/1742-6596/933/1/012018.

10. Ehlers, Jens, et al. “Self-Adaptive Software System Monitoring for Performance Anomaly Localization.” *Proceedings of the 8th ACM International Conference on Autonomic Computing - ICAC ’11*, 2011. *Crossref*, https://doi.org/10.1145/1998582.1998628.

The End

Questions?

Thank you!