WoCC'24

Women in CS Days Cameroon

UMR IRISA

Towards Linux scheduling bottleneck detection with Machine Learning

Audrey FONGUE

Supervised by:

Pr David BROMBERG
Pr Julia LAWALL

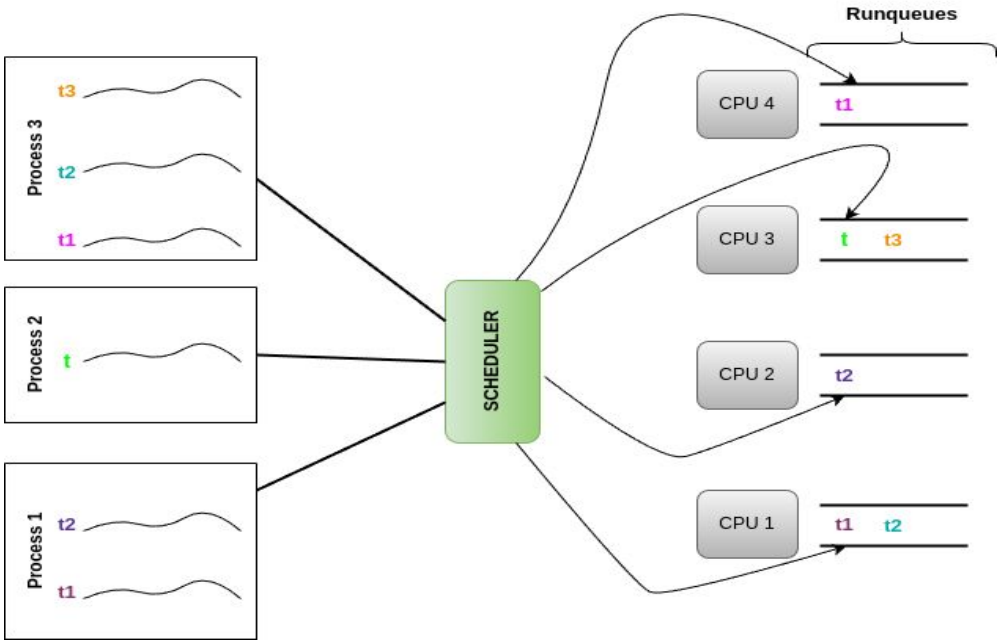
Inria INVENTEURS DU MONDE NUMÉRIQUE

WIDE

Content

- Context and Problem
- Resolution Approach
  - Machine Learning
  - Data Analysis
- Conclusion

Problem and Context



- Tools exist to trace scheduling behavior
- It’s difficult to detect what causes bad performance by looking a trace

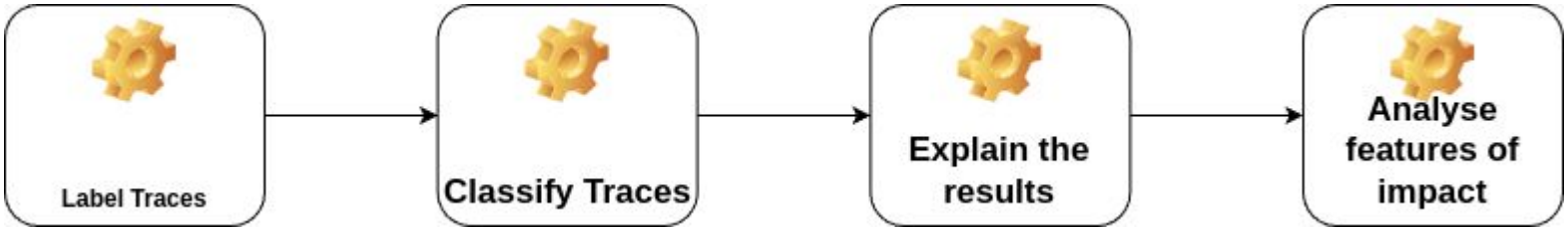
What really causes some runs to take more time than others ? What is the anomaly?

Example of trace

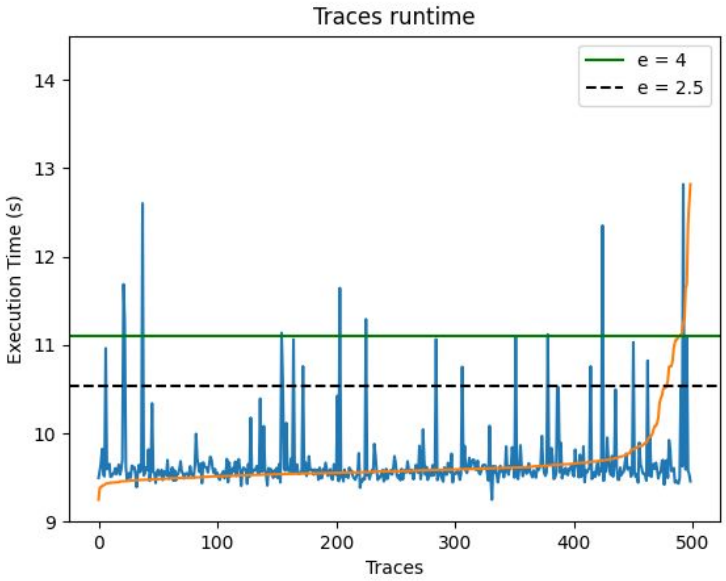
```
cpus=64
          <...>-76814 [050] .....  9187.068845634: sched_process_exec:   filename=/tmp/tmpd27ce3.sh pid=76814 old_pid=76814
          <...>-76814 [050] d..5.  9187.069655764: sched_waking:         comm=nscd pid=1444 prio=120 target_cpu=052
          <...>-76814 [050] d..5.  9187.069657185: sched_wake_idle_without_ipi: cpu=52
          <...>-76814 [050] d..2.  9187.069667215: sched_switch:         prev_comm=tmpd27ce3.sh prev_pid=76814 prev_prio=120 prev_state=S ==> next_comm=swapper/50 next_pid=0 next_prio=120
         <idle>-0     [052] dN.2.  9187.069703369: sched_wakeup:         comm=nscd pid=1444 prio=120 target_cpu=052
         <idle>-0     [052] d..2.  9187.069706114: sched_switch:         prev_comm=swapper/52 prev_pid=0 prev_prio=120 prev_state=R ==> next_comm=nscd next_pid=1444 next_prio=120 Latency: 2.745
 usecs
          <...>-1444  [052] d..2.  9187.069750135: sched_waking:         comm=nscd pid=1452 prio=120 target_cpu=010
          <...>-1444  [052] d..2.  9187.069751221: sched_wake_idle_without_ipi: cpu=10
         <idle>-0     [010] dN.2.  9187.069754777: sched_wakeup:         comm=nscd pid=1452 prio=120 target_cpu=010
          <...>-1444  [052] d..2.  9187.069754947: sched_switch:         prev_comm=nscd prev_pid=1444 prev_prio=120 prev_state=S ==> next_comm=swapper/52 next_pid=0 next_prio=120
         <idle>-0     [010] d..2.  9187.069756268: sched_switch:         prev_comm=swapper/10 prev_pid=0 prev_prio=120 prev_state=R ==> next_comm=nscd next_pid=1452 next_prio=120 Latency: 1.491
 usecs
```

Task	PID	CPU	Interupt$_s$	resched	Context	Preempt	Migrate_d	Times	Event
bt.B.x	13168	4	0	0	3	1	1	1492.812428	wake_without_ip
bt.B.x	13168	4	0	1	3	3	1	1492.812434	switch
bt.B.x	13170	1	0	1	3	1	1	1492.812440	migrate_task
bt.B.x	13168	2	1	0	2	11	1	1492.812500	wakeup
bt.B.x	13173	3	1	0	3	1	1	1492.812513	waking
bt.B.x	13173	3	1	1	2	1	1	1492.812515	wakeup$_new$
bt.B.x	13170	4	0	0	4	1	1	1492.812518	switch
bt.B.x	13170	4	1	0	7	0	1	1492.812522	switch

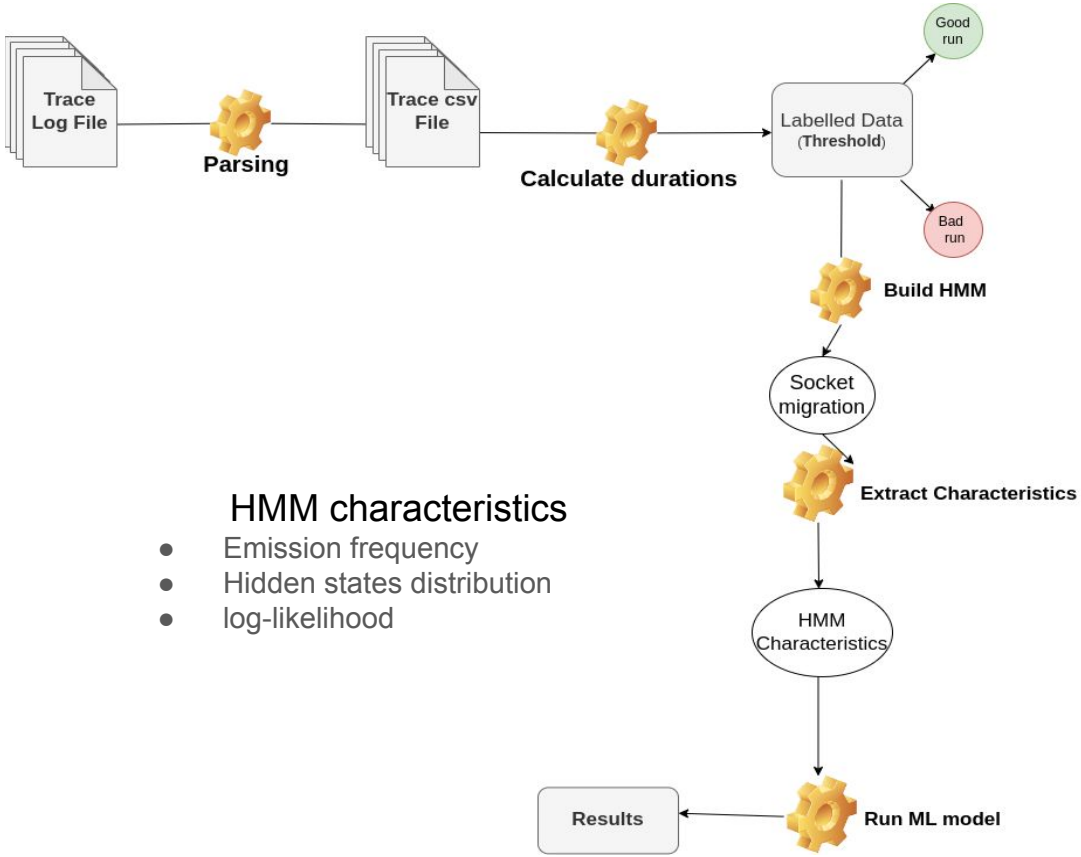
Resolution Approach: **Machine Learning**



Traces processing



Trace as a Markov Chain



HMM characteristics

- Emission frequency
- Hidden states distribution
- log-likelihood

Threshold: change point detection

Little overview of Markov chains

Markov Chain of weather forecasting

A stochastic model describing a sequence of possible events where the probability of each event depends only on the state attained in the previous event.

$$P(X_{n+1} = x_{n+1} \mid X_0 = x_0 , X_1 = x_1 , \ldots X_n = x_n) = P(X_{n+1} = x_{n+1} \mid X_n = x_n)$$

Components

- States: The different possible states.
- Transition Probabilities: The probabilities of moving from one state to another.

Hidden Markov Model

- Hidden States: The true states which are not observed.
- Observable States: The observations dependent on the hidden states.
- Transition Probabilities, Emission Probabilities, Initial Probabilities.

Little overview of Markov chains: Example

Markov Chain of weather forecasting

States: Sunny, Cloudy, Rainy

Example of forecast for five days: Sunny -> Sunny -> Rainy -> Sunny -> Cloudy

Transition Matrix			
	Sunny	Cloudy	Rainy
Sunny	70%	20%	10%
Cloudy	30%	50%	20%
Rainy	20%	30%	50%

Dataset Description

Applications	Linux Kernel versions/Clusters
BT: Block Tridiagonal Server	From 6.1.0 to 6.7.0 (October to November 2022) 500 runs per version
LU: LU Solver	
FT: 3-D FF -PDE	
SP: Pentadiagonal Server	yeti -> Intel Xeon Gold 6130, 16 cores/CPU, x86_64, 768 GiB / dahu -> Intel Xeon Gold 6130,16 cores/CPU, x86_64, 192 GiB
CG: Conjugate Gradient	
UA: Unstructured Adaptative	

Mean proportion of good and bad traces:

- Normal traces: 88%
- Abnormal traces: 12%

Dataset features and metrics of evaluation

- ***Emission frequency***: frequency of migration within the same socket, frequency of migration to another socket
- ***Hidden States distribution***: the frequency of occurrence of each thread in the chain
- The length of the generated Markov Chain

Metrics

Accuracy: proportion of well-predicted (normal and abnormal) traces among all traces.

Precision: proportion of correct normal traces among normal predicted traces.

Recall: proportion of normal traces correctly predicted among normal traces.

F1-score: harmonic mean between recall and precision.

Results: Single applications - Merged application with undersampling(Random undersampling(**Rus**) et Condensed Nearest neighbour(**CNN**))

SVM

Table: BT

Dahu	Accuracy	Precision	Recall	f1-score
v_6.1.0	0.95	0.75	0.3	0.43
v_6.2.0	0.95	1	0.11	0.2
v_6.3.0	0.88	0	0	0
v_6.4.0	0.91	0.83	0.27	0.41
v_6.5.0	0.90	0	0	0

Table: LU

Dahu	Accuracy	Precision	Recall	f1-score
v_6.1.0	0.97	0	0	0
v_6.2.0	0.97	1	0.2	0.33
v_6.3.0	0.95	0	0	0
v_6.4.0	0.98	1	0.5	0.66
v_6.5.0	0.98	1	0.5	0.66

Table: SP

Dahu	Accuracy	Precision	Recall	f1-score
v_6.1.0	0.91	1	0.13	0.16
v_6.5.0	0.93	0.66	0.57	0.62

Table: RUS: (BT,SP,CG,FT,LU)

Dahu	Accuracy	Precision	Recall	f1-score
v_6.1.0	0.71	0.75	0.75	0.75
v_6.2.0	0.78	0.77	0.7	0.74
v_6.3.0	0.63	0.52	0.81	0.63

Table: CNN: (BT,SP,CG,FT,LU)

Dahu	Accuracy	Precision	Recall	f1-score
v_6.1.0	0.75	0.55	0.62	0.58
v_6.2.0	0.75	0.33	0.08	0.13
v_6.3.0	0.81	0.5	0.15	0.24

Results: Single applications - Merged application with undersampling(Random undersampling(**Rus**) et Condensed Nearest neighbour(**CNN**))

OCSVM

Table: BT

Dahu	Accuracy	Precision	Recall	f1-score
v_6.1.0	0.93	0.5	0.33	0.38
v_6.2.0	0.93	0.43	0.33	0.38
v_6.3.0	0.79	0.31	0.65	0.42
v_6.4.0	0.89	0.53	0.44	0.48
v_6.5.0	0.81	0.31	0.73	0.43

Table: LU

Dahu	Accuracy	Precision	Recall	f1-score
v_6.1.0	0.95	0.29	0.5	0.36
v_6.2.0	0.98	1	0.4	0.57
v_6.3.0	0.91	0.27	0.57	0.36
v_6.4.0	0.99	1	0.5	0.66
v_6.5.0	0.97	0.5	0.5	0.5

Table: SP

Dahu	Accuracy	Precision	Recall	f1-score
v_6.1.0	0.81	0.21	0.33	0.26
v_6.5.0	0.92	0.55	0.71	0.63

Table: RUS: (BT,SP,CG,FT,LU)

Dahu	Accuracy	Precision	Recall	f1-score
v_6.1.0	0.51	0.03	0.39	0.07
v_6.2.0	0.53	0.1	0.57	0.02
v_6.3.0	0.51	0.01	0.16	0.02

Table: CNN: (BT,SP,CG,FT,LU)

Dahu	Accuracy	Precision	Recall	f1-score
v_6.1.0	0.52	0.60	0.65	0.62
v_6.2.0	0.54	0.01	0.57	0.02
v_6.3.0	0.51	0.01	0.22	0.02

Results: Single applications - Merged application with undersampling(Random undersampling(**Rus**) et Condensed Nearest neighbour(**CNN**))

OCSVM

Table: BT

Dahu	Accuracy	Precision	Recall	f1-score
v_6.1.0	0.93	0.5	0.33	0.38
v_6.2.0	0.93	0.43	0.33	0.38
v_6.3.0	0.79	0.31	0.65	0.42
v_6.4.0	0.89	0.53	0.44	0.48
v_6.5.0	0.81	0.31	0.73	0.43

Table: LU

Dahu	Accuracy	Precision	Recall	f1-score
v_6.1.0	0.95	0.29	0.5	0.36
v_6.2.0	0.98	1	0.4	0.57
v_6.3.0	0.91	0.27	0.57	0.36
v_6.4.0	0.99	1	0.5	0.66
v_6.5.0	0.97	0.5	0.5	0.5

Table: SP

Dahu	Accuracy	Precision	Recall	f1-score
v_6.1.0	0.81	0.21	0.33	0.26
v_6.5.0	0.92	0.55	0.71	0.63

Table: RUS: (BT,SP,CG,FT,LU)

Dahu	Accuracy	Precision	Recall	f1-score
v_6.1.0	0.51	0.03	0.39	0.07
v_6.2.0	0.53	0.1	0.57	0.02
v_6.3.0	0.51	0.01	0.16	0.02

Table: CNN: (BT,SP,CG,FT,LU)

Dahu	Accuracy	Precision	Recall	f1-score
v_6.1.0	0.52	0.60	0.65	0.62
v_6.2.0	0.54	0.01	0.57	0.02
v_6.3.0	0.51	0.01	0.22	0.02

Results: Single applications - Merged application with undersampling(Random undersampling(**Rus**) et Condensed Nearest neighbour(**CNN**))

Logistic Regression

Table: BT

Dahu	Accuracy	Precision	Recall	f1-score
v_6.1.0	0.94	0.6	0.3	0.4
v_6.2.0	0.95	0.66	0.22	0.33
v_6.3.0	0.85	0.2	0.12	0.15
v_6.4.0	0.86	0.38	0.28	0.32
v_6.5.0	0.88	0.38	0.2	0.26

Table: LU

Dahu	Accuracy	Precision	Recall	f1-score
v_6.1.0	0.97	0	0	0
v_6.2.0	0.98	1	0.6	0.75
v_6.3.0	0.96	0.6	0.43	0.5
v_6.4.0	0.99	0.75	0.75	0.75
v_6.5.0	0.98	1	0.5	0.66

Table: SP

Dahu	Accuracy	Precision	Recall	f1-score
v_6.1.0	0.89	0.43	0.2	0.27
v_6.5.0	0.92	0.63	0.36	0.45

Table: RUS: (BT,SP,CG,FT,LU)

Dahu	Accuracy	Precision	Recall	f1-score
v_6.1.0	0.71	0.83	0.63	0.71
v_6.2.0	0.69	0.63	0.70	0.67
v_6.3.0	0.60	0.5	0.63	0.55

Table: CNN: (BT,SP,CG,FT,LU)

Dahu	Accuracy	Precision	Recall	f1-score
v_6.1.0	0.65	0.34	0.39	0.37
v_6.2.0	0.76	0.55	0.38	0.45
v_6.3.0	0.74	0.53	0.6	0.56

Merged data (BT, LU, SP, Ft, CG)

Models Explained: SVM, Random Forest, Logistic Regression. Normal and Abnormal traces of test dataset for version 6.1.0 and 6.2.0

Version 6.1.0

Normal traces :

- bt.B.x_dahu-3_6.1.0_performance_50.txt
- cg.B.x_dahu-6_6.1.0_performance_141.txt
- bt.B.x_dahu-3_6.1.0_performance_458.txt
- bt.B.x_dahu-3_6.1.0_performance_499.txt
- lu.B.x_dahu-5_6.1.0_performance_172.txt
- cg.B.x_dahu-6_6.1.0_performance_163.txt
- sp.B.x_dahu-7_6.1.0_performance_146.txt
- lu.B.x_dahu-5_6.1.0_performance_296.txt
- bt.B.x_dahu-3_6.1.0_performance_180.txt
- bt.B.x_dahu-3_6.1.0_performance_14.txt
- bt.B.x_dahu-3_6.1.0_performance_443.txt
- lu.B.x_dahu-5_6.1.0_performance_14.txt
- ft.B.x_dahu-4_6.1.0_performance_178.txt
- lu.B.x_dahu-5_6.1.0_performance_102.txt
- ft.B.x_dahu-4_6.1.0_performance_220.txt
- ft.B.x_dahu-4_6.1.0_performance_70.txt
- sp.B.x_dahu-7_6.1.0_performance_60.txt

Abnormal traces :

- sp.B.x_dahu-7_6.1.0_performance_447.txt
- ft.B.x_dahu-4_6.1.0_performance_105.txt
- ft.B.x_dahu-4_6.1.0_performance_4.txt
- ft.B.x_dahu-4_6.1.0_performance_469.txt
- ft.B.x_dahu-4_6.1.0_performance_160.txt
- ft.B.x_dahu-4_6.1.0_performance_206.txt
- ft.B.x_dahu-4_6.1.0_performance_463.txt
- ft.B.x_dahu-4_6.1.0_performance_20.txt
- ft.B.x_dahu-4_6.1.0_performance_68.txt
- bt.B.x_dahu-3_6.1.0_performance_11.txt
- ft.B.x_dahu-4_6.1.0_performance_432.txt
- cg.B.x_dahu-6_6.1.0_performance_397.txt
- ft.B.x_dahu-4_6.1.0_performance_197.txt
- ft.B.x_dahu-4_6.1.0_performance_255.txt
- ft.B.x_dahu-4_6.1.0_performance_39.txt

Version 6.2.0

Normal traces:

- lu.B.x_dahu-5_6.2.0_performance_86.txt
- ft.B.x_dahu-4_6.2.0_performance_22.txt
- sp.B.x_dahu-7_6.2.0_performance_195.txt
- lu.B.x_dahu-5_6.2.0_performance_343.txt
- lu.B.x_dahu-5_6.2.0_performance_277.txt
- cg.B.x_dahu-6_6.2.0_performance_220.txt
- sp.B.x_dahu-7_6.2.0_performance_113.txt
- bt.B.x_dahu-3_6.2.0_performance_50.txt
- sp.B.x_dahu-7_6.2.0_performance_120.txt
- cg.B.x_dahu-6_6.2.0_performance_164.txt

Abnormal traces:

- cg.B.x_dahu-6_6.2.0_performance_319.txt
- cg.B.x_dahu-6_6.2.0_performance_483.txt
- sp.B.x_dahu-7_6.2.0_performance_464.txt
- sp.B.x_dahu-7_6.2.0_performance_266.txt
- lu.B.x_dahu-5_6.2.0_performance_260.txt
- cg.B.x_dahu-6_6.2.0_performance_90.txt
- ft.B.x_dahu-4_6.2.0_performance_40.txt
- sp.B.x_dahu-7_6.2.0_performance_381.txt
- ft.B.x_dahu-4_6.2.0_performance_62.txt
- cg.B.x_dahu-6_6.2.0_performance_235.txt
- bt.B.x_dahu-3_6.2.0_performance_467.txt

Explainability of the results

Table: SVM

Dahu	Accuracy	Precision	Recall	f1-score
v_6.1.0	0.75	0.88	0.53	0.66
v_6.2.0	0.80	1	0.64	0.77

version 6.2.0

Traces predicted by the model as Abnormal:

- 'sp.B.x_dahu-7_6.2.0_performance_464.txt'
- 'sp.B.x_dahu-7_6.2.0_performance_266.txt'
- 'lu.B.x_dahu-5_6.2.0_performance_260.txt'
- 'ft.B.x_dahu-4_6.2.0_performance_40.txt'
- 'sp.B.x_dahu-7_6.2.0_performance_381.txt'
- 'ft.B.x_dahu-4_6.2.0_performance_62.txt'
- 'bt.B.x_dahu-3_6.2.0_performance_467.txt'

7/11

: Wrong Prediction
: Number of traces predicted as Abnormal among the anormal traces of the test set

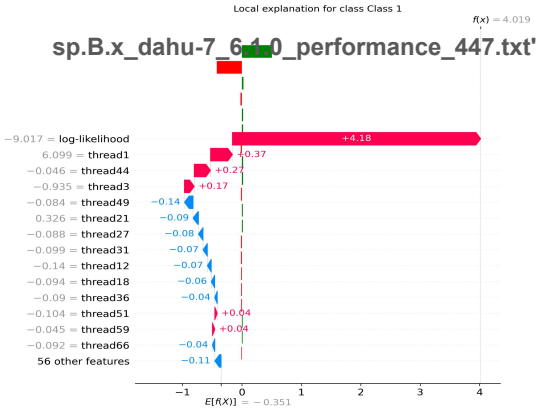
version 6.1.0

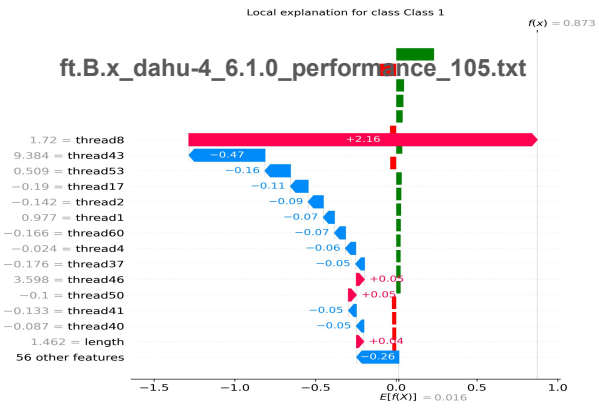
Traces predicted by the model as Abnormal:

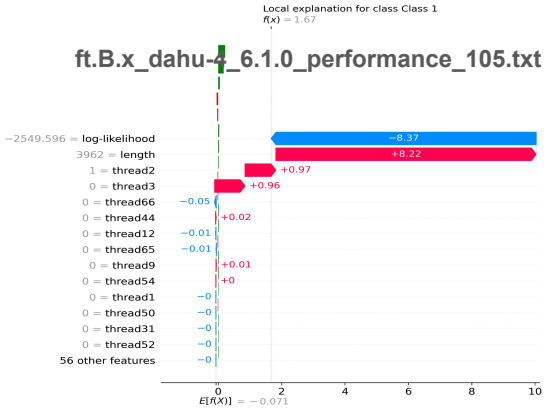
- 'sp.B.x_dahu-7_6.1.0_performance_447.txt'
- 'ft.B.x_dahu-4_6.1.0_performance_105.txt'
- 'ft.B.x_dahu-4_6.1.0_performance_469.txt'
- 'ft.B.x_dahu-4_6.1.0_performance_160.txt'
- 'lu.B.x_dahu-5_6.1.0_performance_172.txt'
- 'ft.B.x_dahu-4_6.1.0_performance_432.txt'
- 'ft.B.x_dahu-4_6.1.0_performance_197.txt'
- 'ft.B.x_dahu-4_6.1.0_performance_255.txt'
- 'ft.B.x_dahu-4_6.1.0_performance_39.txt'

9/15

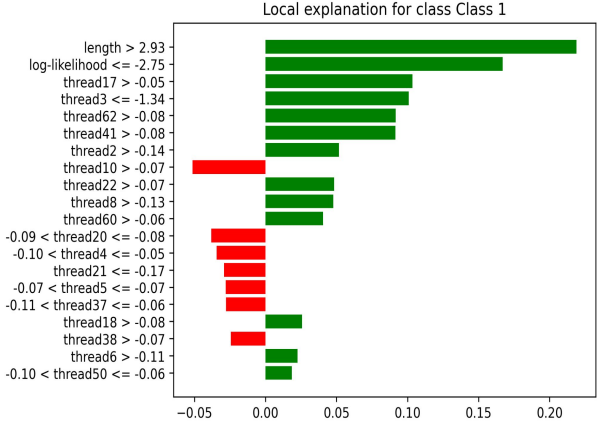
Explanations with SHAP the first three traces predicted as Abnormal for version 1('sp.B.x_dahu-7_6.1.0_performance_447.txt', 'ft.B.x_dahu 4_6.1.0_performance_105.txt', 'ft.B.x_dahu 4_6.1.0_performance_469.txt').

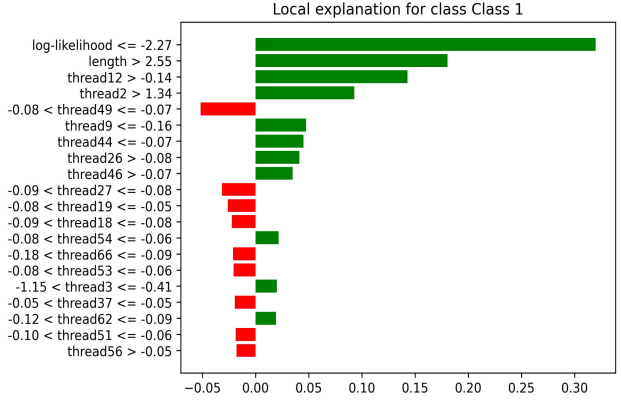


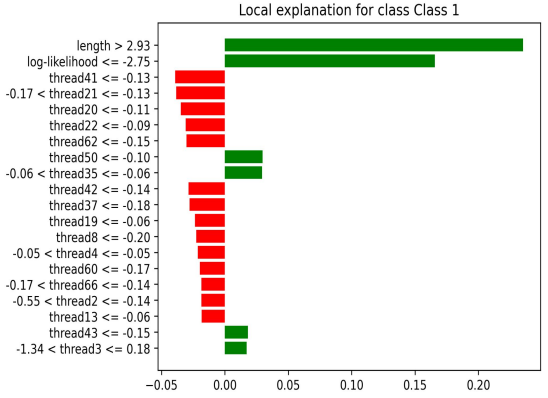




Explanations with LIME the first three traces predicted as Abnormal for version 1('sp.B.x_dahu-7_6.1.0_performance_447.txt', 'ft.B.x_dahu 4_6.1.0_performance_105.txt', 'ft.B.x_dahu 4_6.1.0_performance_469.txt').







Explainability of the results

Table: Random Forest

Dahu	Accuracy	Precision	Recall	f1-score
v_6.1.0	0.91	1	0.8	0.88
v_6.2.0	0.90	0.90	0.90	0.90

version 6.2.0

Traces predicted by the model as Abnormal:

- 'cg.B.x_dahu-6_6.2.0_performance_483.txt'
- 'sp.B.x_dahu-7_6.2.0_performance_464.txt'
- 'sp.B.x_dahu-7_6.2.0_performance_266.txt'
- 'cg.B.x_dahu-6_6.2.0_performance_220.txt'
- 'lu.B.x_dahu-5_6.2.0_performance_260.txt'
- 'cg.B.x_dahu-6_6.2.0_performance_90.txt'
- 'ft.B.x_dahu-4_6.2.0_performance_40.txt'
- 'sp.B.x_dahu-7_6.2.0_performance_381.txt'
- 'ft.B.x_dahu-4_6.2.0_performance_62.txt'
- 'cg.B.x_dahu-6_6.2.0_performance_235.txt'
- 'bt.B.x_dahu-3_6.2.0_performance_467.txt'

11/11

version 6.1.0

Traces predicted by the model as Abnormal:

- 'sp.B.x_dahu-7_6.1.0_performance_447.txt'
- 'ft.B.x_dahu-4_6.1.0_performance_105.txt'
- 'ft.B.x_dahu-4_6.1.0_performance_4.txt'
- 'ft.B.x_dahu-4_6.1.0_performance_469.txt'
- 'ft.B.x_dahu-4_6.1.0_performance_160.txt'
- 'ft.B.x_dahu-4_6.1.0_performance_206.txt'
- 'ft.B.x_dahu-4_6.1.0_performance_20.txt'
- 'ft.B.x_dahu-4_6.1.0_performance_68.txt'
- 'ft.B.x_dahu-4_6.1.0_performance_432.txt'
- 'ft.B.x_dahu-4_6.1.0_performance_197.txt'
- 'ft.B.x_dahu-4_6.1.0_performance_255.txt'
- 'ft.B.x_dahu-4_6.1.0_performance_39.txt'

12/15

Explainability of the results

Table: Logistic Regression

Dahu	Accuracy	Precision	Recall	f1-score
v_6.1.0	0.84	0.92	0.73	0.81
v_6.2.0	0.81	0.88	0.72	0.8

version 6.2.0

Traces predicted by the model as Abnormal:

- 'sp.B.x_dahu-7_6.2.0_performance_464.txt'
- 'sp.B.x_dahu-7_6.2.0_performance_266.txt'
- 'lu.B.x_dahu-5_6.2.0_performance_260.txt'
- 'bt.B.x_dahu-3_6.2.0_performance_50.txt'
- 'ft.B.x_dahu-4_6.2.0_performance_40.txt'
- 'sp.B.x_dahu-7_6.2.0_performance_381.txt'
- 'ft.B.x_dahu-4_6.2.0_performance_62.txt'
- 'cg.B.x_dahu-6_6.2.0_performance_235.txt'
- 'bt.B.x_dahu-3_6.2.0_performance_467.txt'

9/11

version 6.1.0

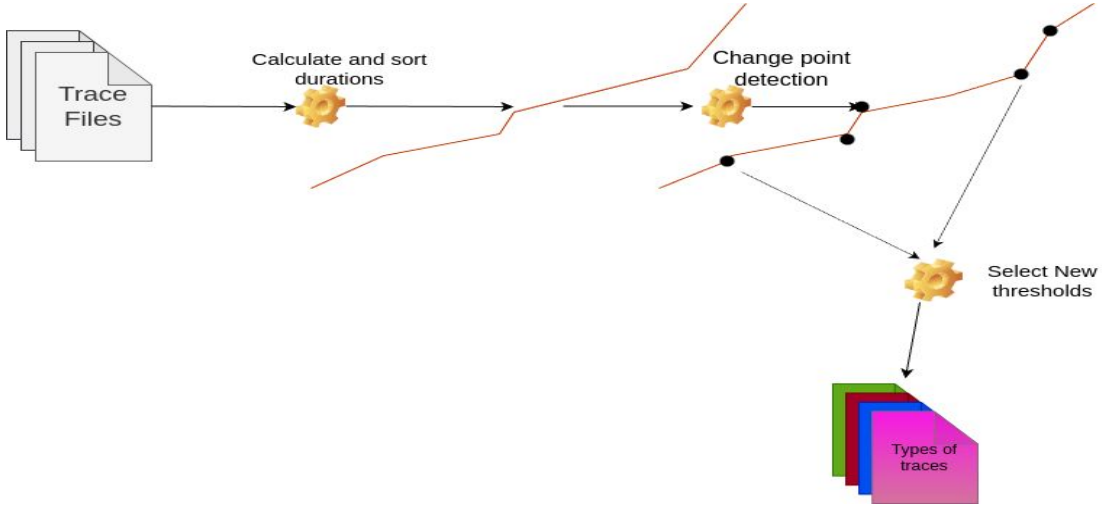
Traces predicted by the model as Abnormal:

- 'sp.B.x_dahu-7_6.1.0_performance_447.txt'
- 'ft.B.x_dahu-4_6.1.0_performance_105.txt'
- 'ft.B.x_dahu-4_6.1.0_performance_469.txt'
- 'ft.B.x_dahu-4_6.1.0_performance_160.txt'
- 'lu.B.x_dahu-5_6.1.0_performance_172.txt'
- 'ft.B.x_dahu-4_6.1.0_performance_206.txt'
- 'ft.B.x_dahu-4_6.1.0_performance_20.txt'
- 'ft.B.x_dahu-4_6.1.0_performance_68.txt'
- 'ft.B.x_dahu-4_6.1.0_performance_432.txt'
- 'ft.B.x_dahu-4_6.1.0_performance_197.txt'
- 'ft.B.x_dahu-4_6.1.0_performance_255.txt'
- 'ft.B.x_dahu-4_6.1.0_performance_39.txt'

12/15

These models do a better job of classification, but the attributes that contribute most to these predictions are not the expected ones (e.g:the percentage of migration)

Resolution Approach: **Data Analysis**



Types of bottleneck

- Bottleneck CPU
  - Context-switch frequency
  - Latency wakeup-switch, waking-wakeup
- Bottleneck I/O
  - Latency sleeping- wakeup/waking
- Bottleneck Memory
  - Wait for memory pages to be exchanged with disk
  - Latency sleeping- wakeup/waking
- Bottleneck Lock
  - waiting for shared lock (Lock contention)
  - Latency wakeup-switch
- Bottleneck Scheduder
  - Inefficient Usage of CPU
- Bottleneck Affinity CPU
  - Cash overflow (several migrations)

- **Task/PID**
- **TimeStamp**
- **CPU**
- **Event**: waking, wakeup, switch
- **Context**

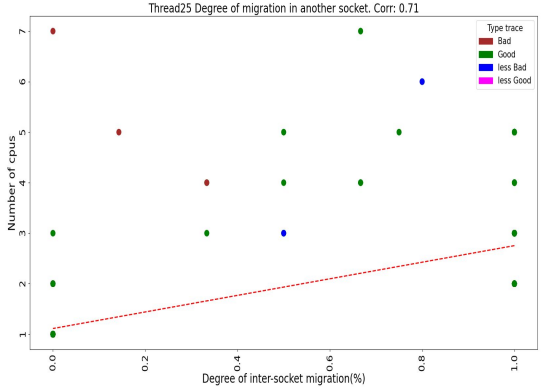
Brown: Abnormal traces (highest execution time)

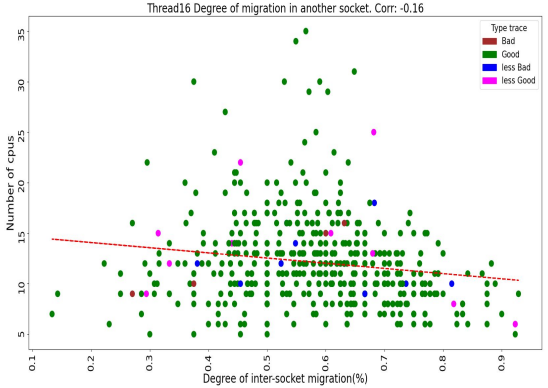
Green: Normal (Lowest execution time)

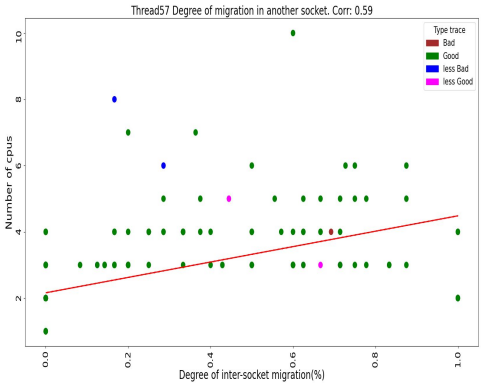
Blue: less bad traces

Magenta: less good traces

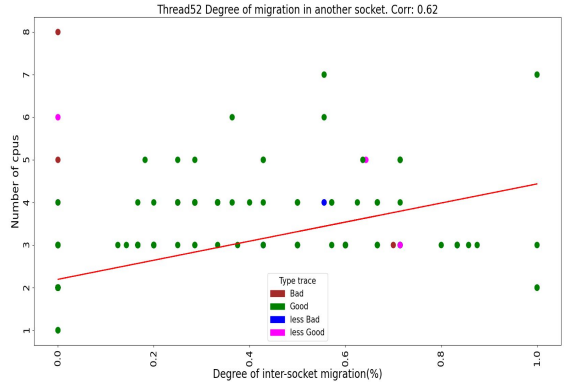
When processes/threads migrate, what percentage migrates to another socket? Do Abnormal traces have a higher percentage?

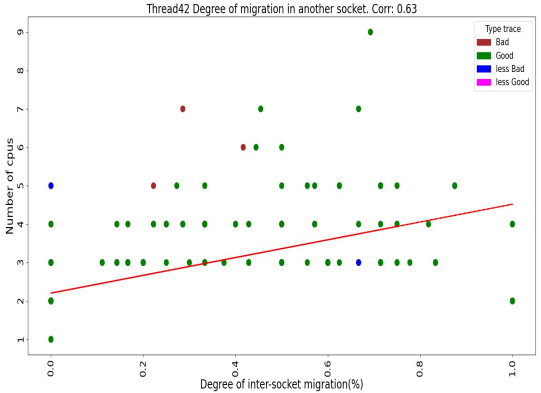


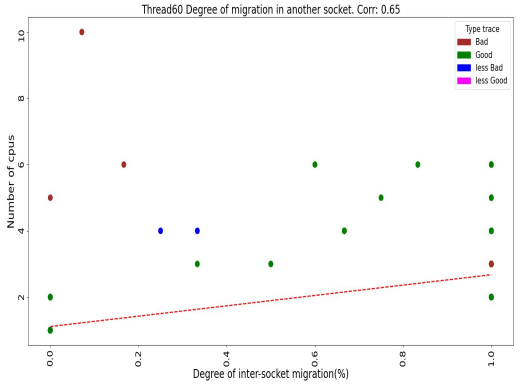




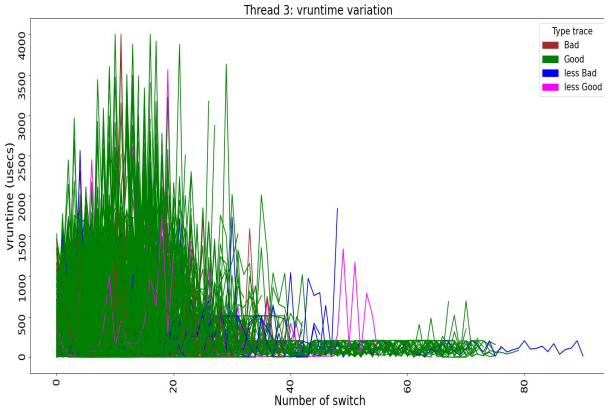
We can see that for most threads, Abnormal traces make more migrations, but these migrations are rarely to a different socket. It's normal traces that are most likely to migrate to another socket.

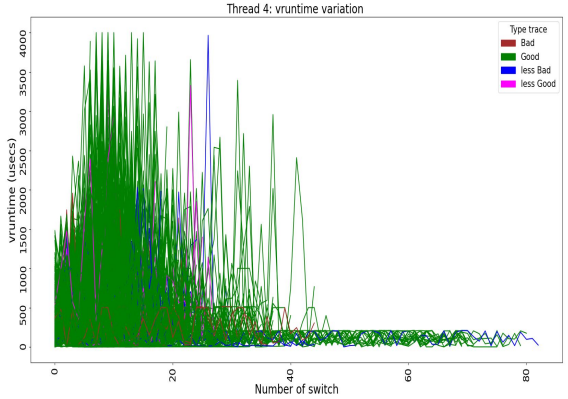


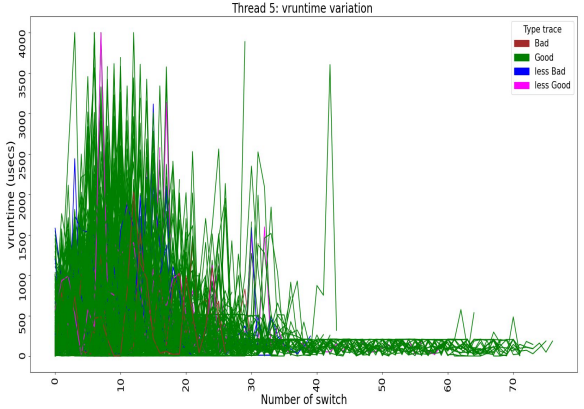




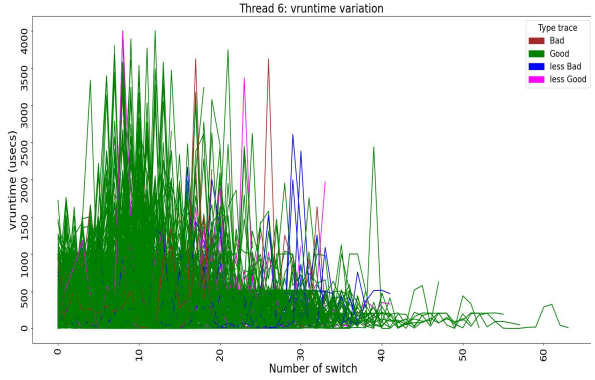
Do normal traces take less time to execute because the scheduler gives them a longer execution time (vruntime) at each context switch?

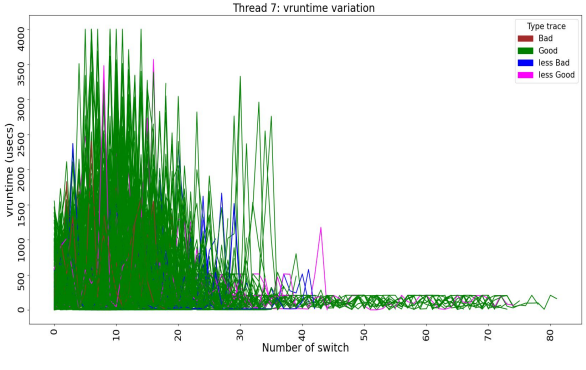


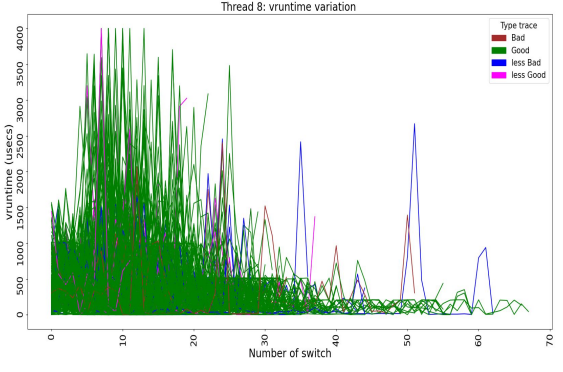




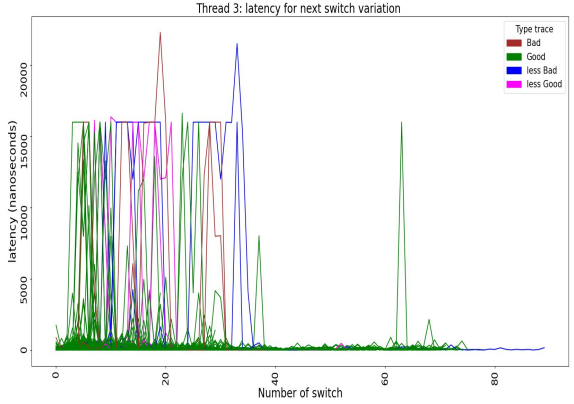
Wesee that anormal traces generally have a lower vruntime than the others, but when they start with a high vruntime, towards the end it drops again (case of the blue traces in figure 2).

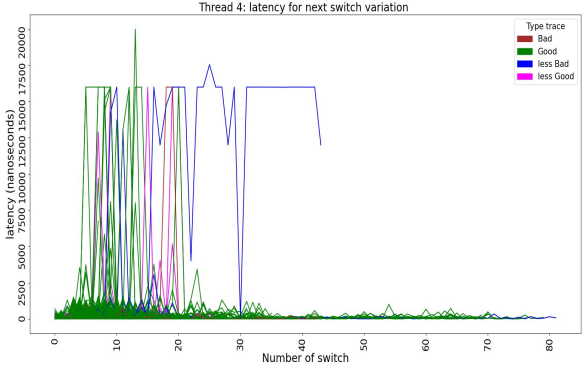


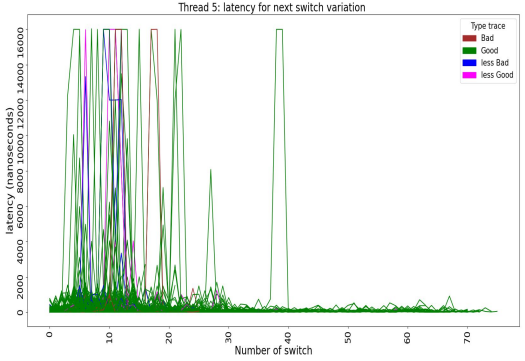




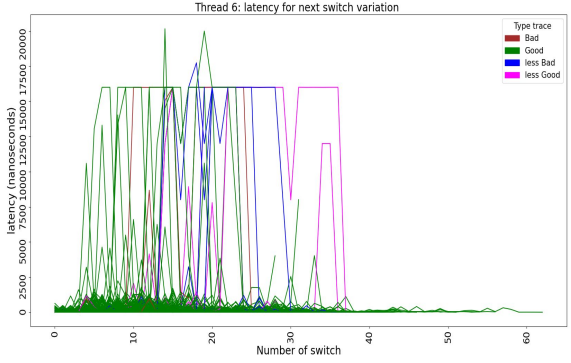
On abnormal traces, when a process runs and is removed from the core, does it take a long time to run again?

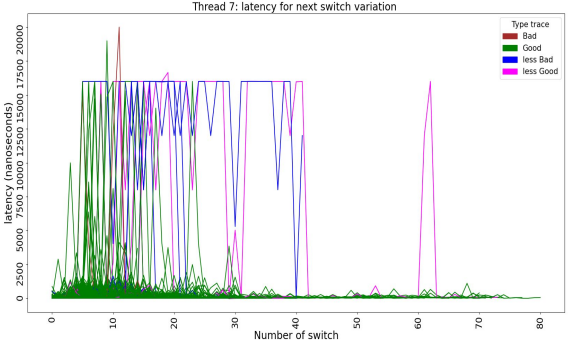


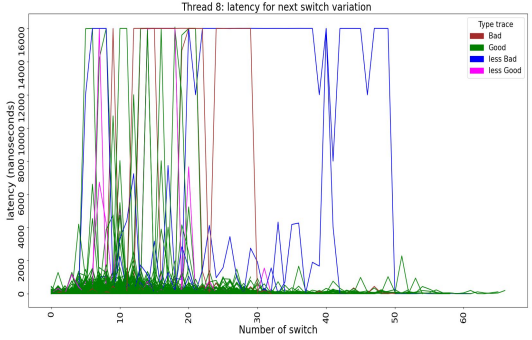




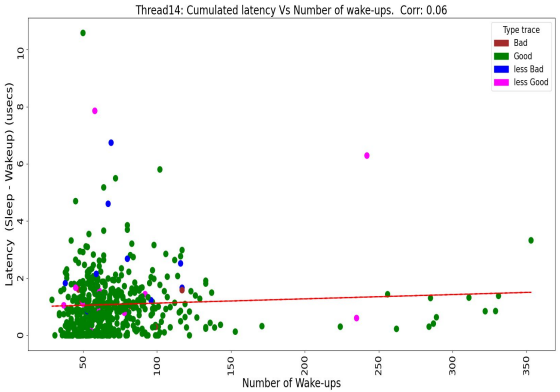
We observe that normal traces rarely take long before the next execution, but it's more common for abnormal traces to take longer.

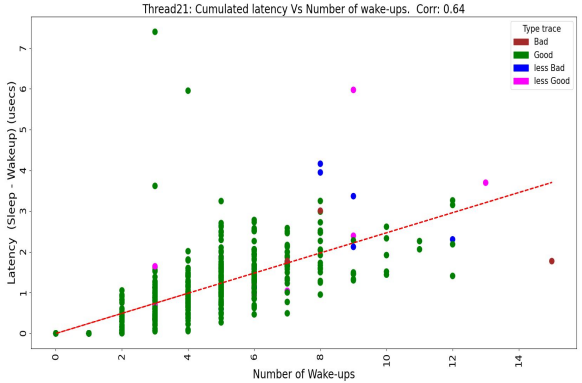


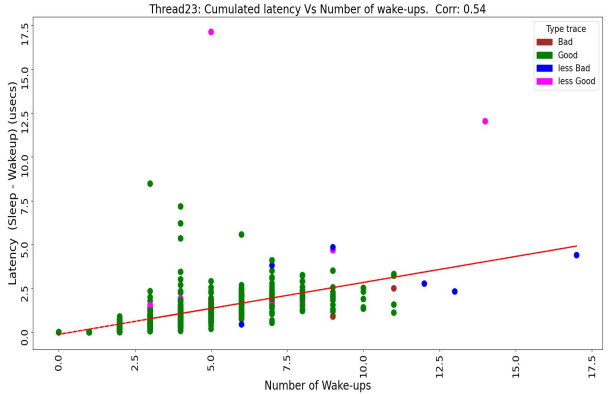




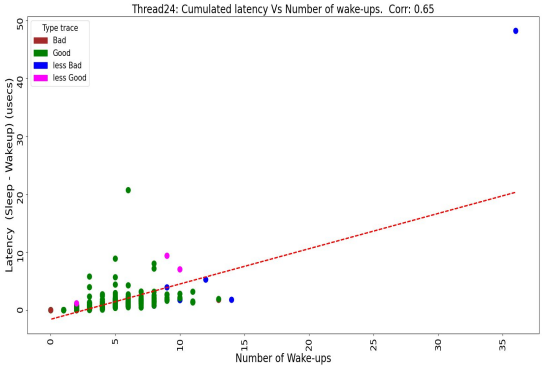
Cumulative latency between sleep and wakeup. Do bad traces remain inactive longer than normal traces throughout program execution? Est ce que le nombre de wakeup a un impact sur ce temps?

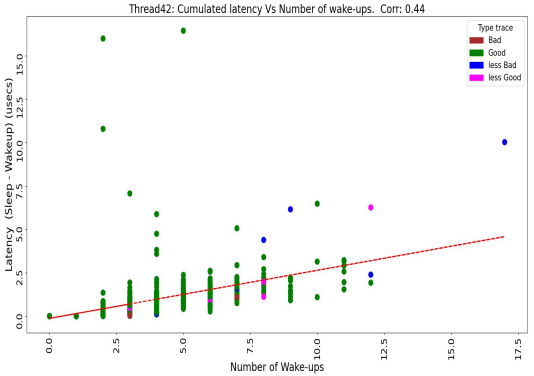


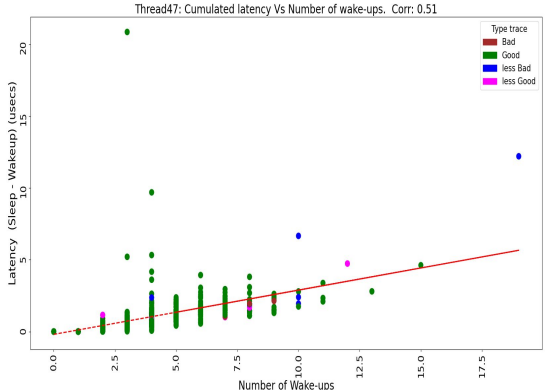




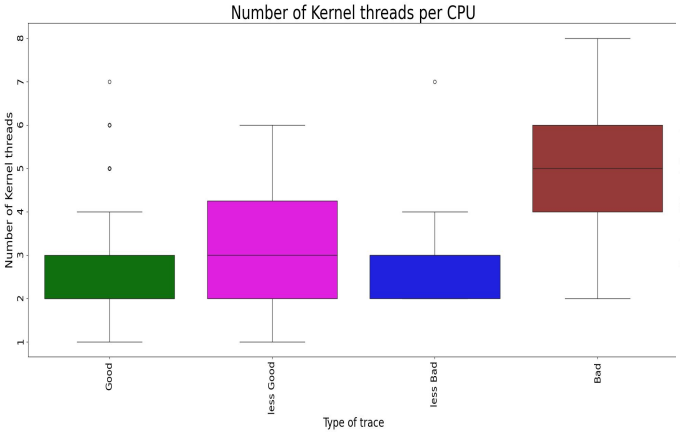
Bad traces have a longer wake-up time, but not always longer. In terms of number of awakenings, it's not always the bad traces that wake up the most. On several threads, the correlation is over 60%.

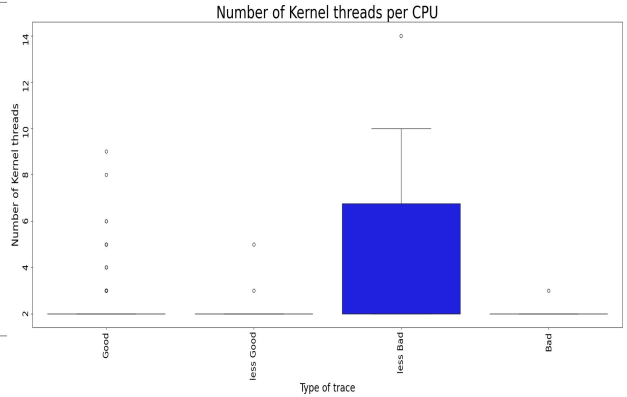


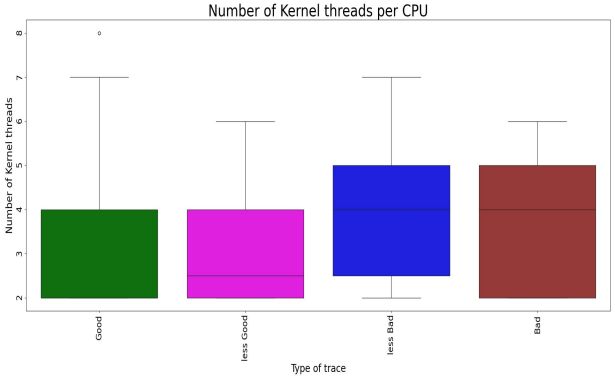




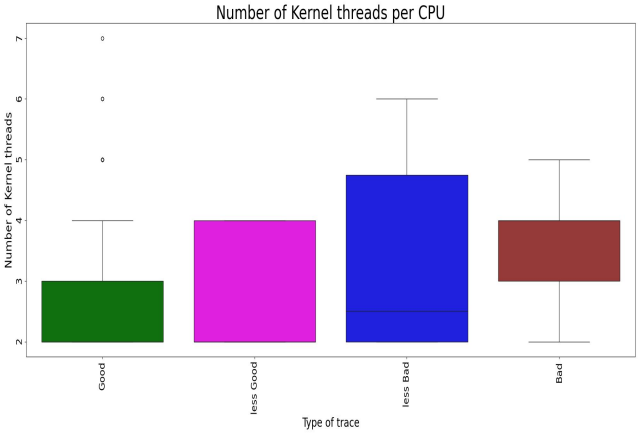
On the different CPUs, were there more kernel threads on the Abnormal traces?

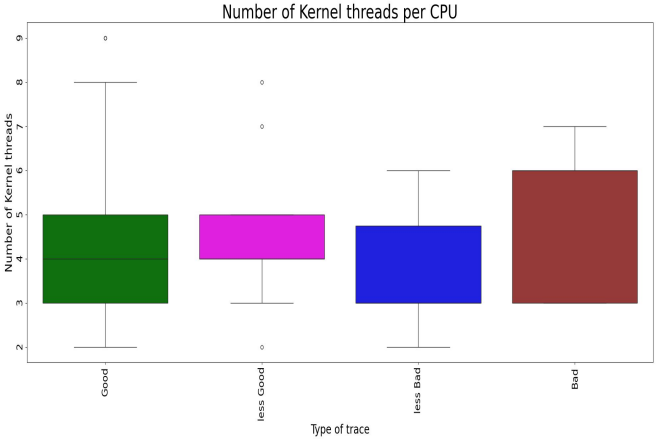


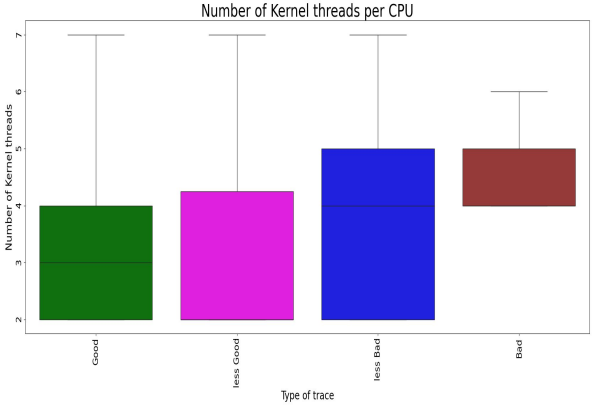




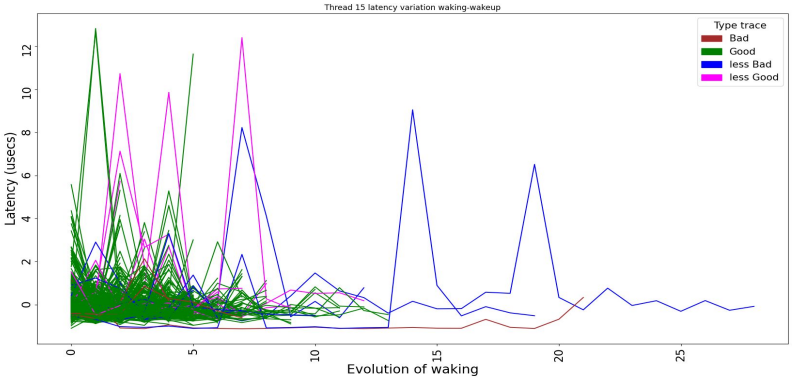
We observe that bad traces have more kernel threads on these cores(1,19,35,33,37,39). This observation is also made on cores: 20, 25,41,43,45,47,49,53,57,59,61,63.

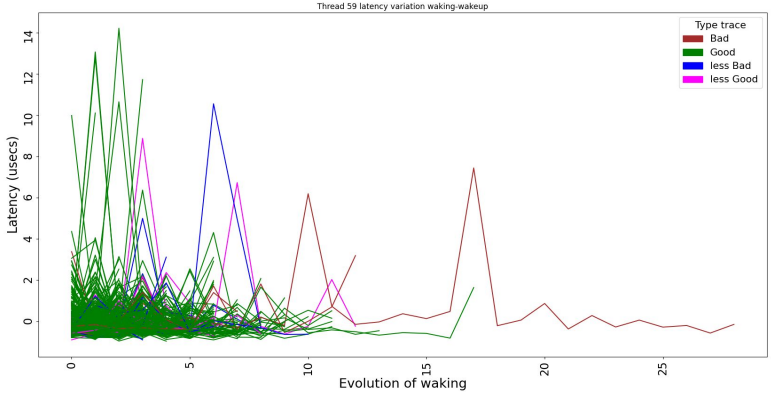




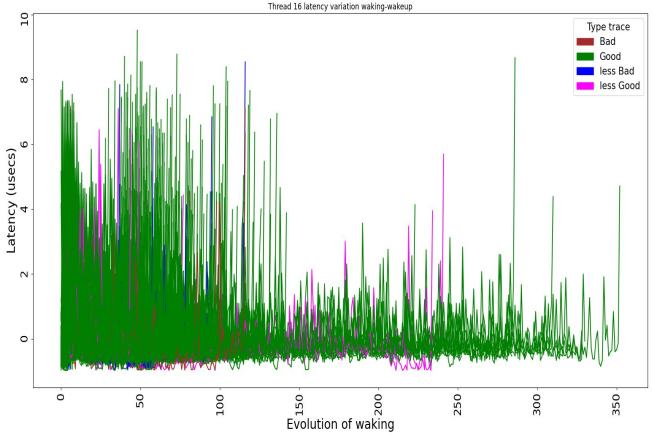


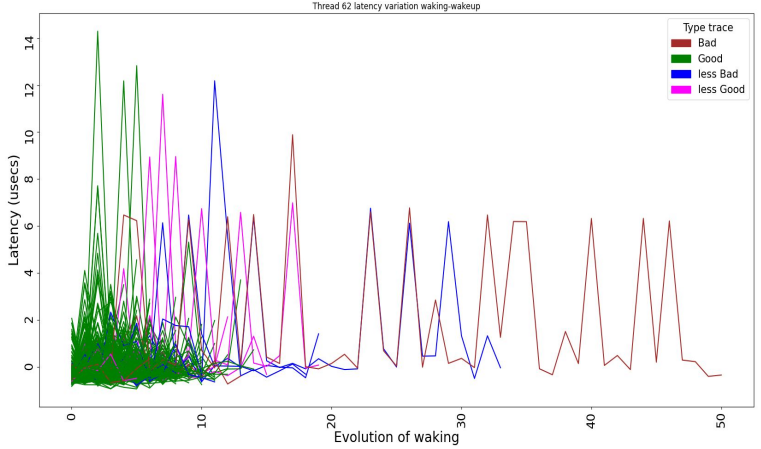
Do anormal traces take longer to load on the runqueue once they have received the response to the event they were waiting for?



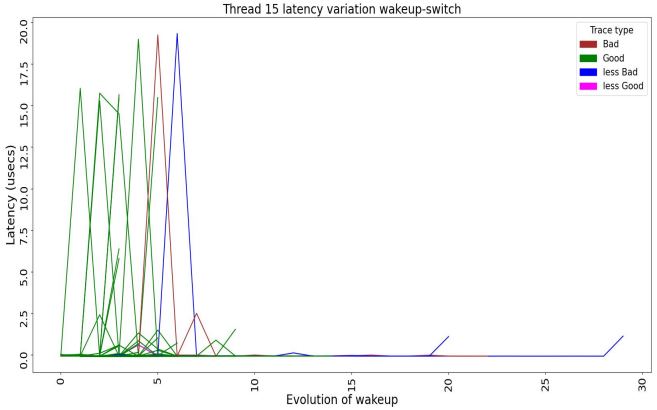


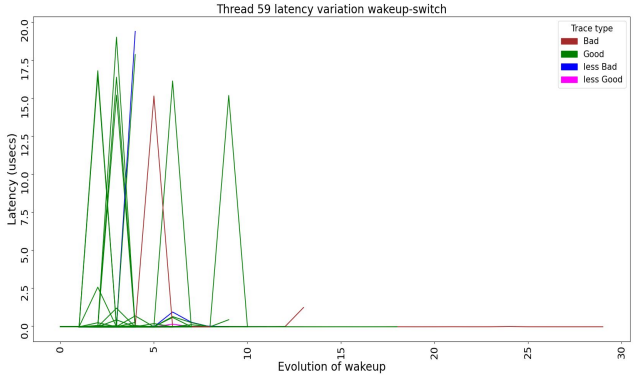
We can see from these graphs that abnormal traces tend to wake up many more times than good ones, and often have a higher latency.



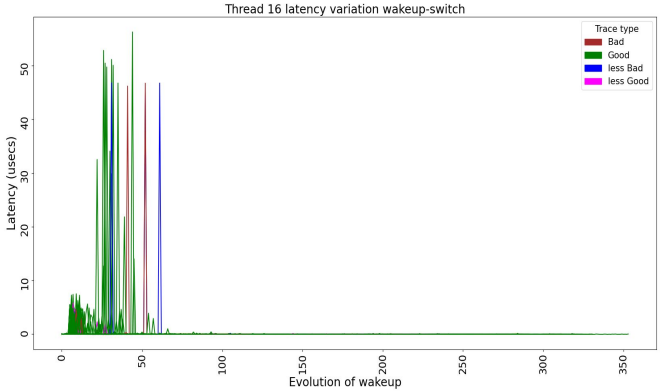


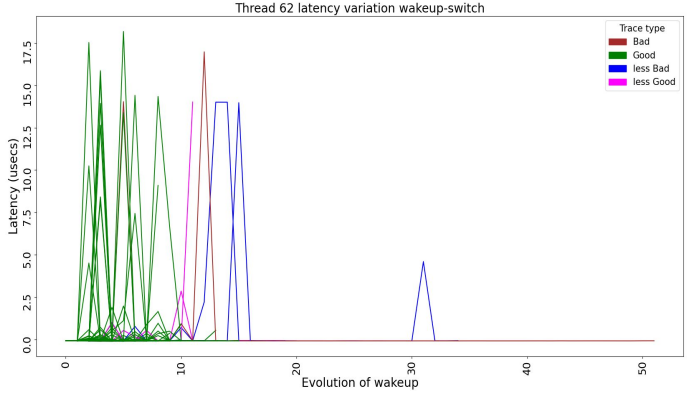
Do abnormal traces take longer to execute once they are on the runqueue?



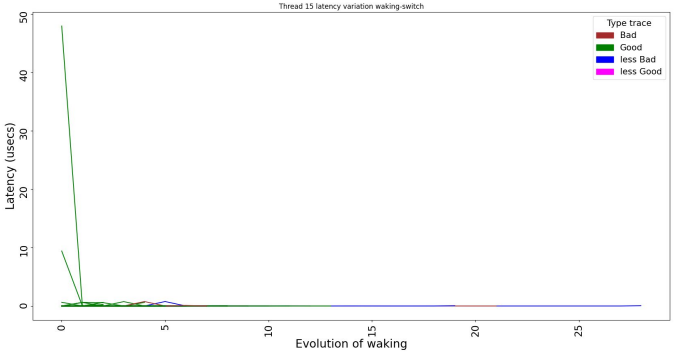


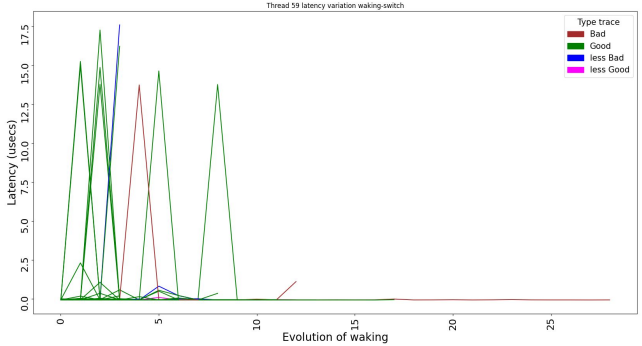
We can see that, at the start of program execution, bad traces don't take too long once they're on the runqueue, but after a while, this time climbs.



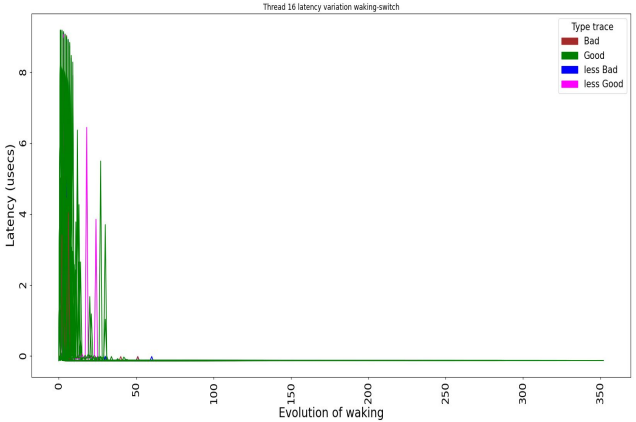


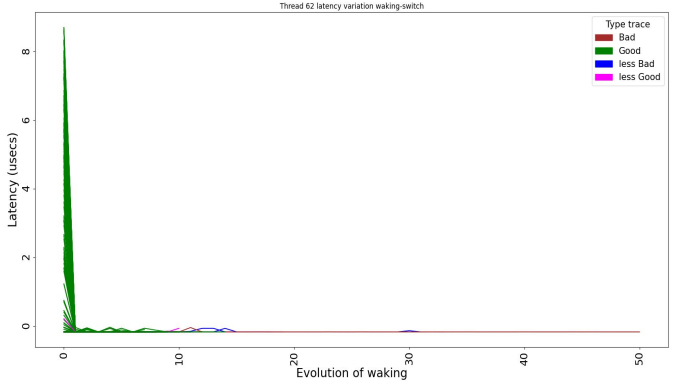
Combination of the two previous questions: Do bad traces take longer from event reception to context switch (waking-switch)?



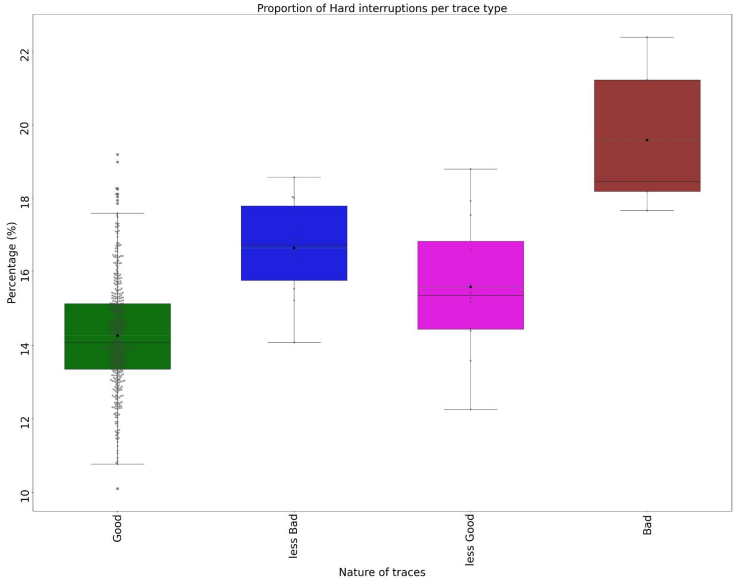


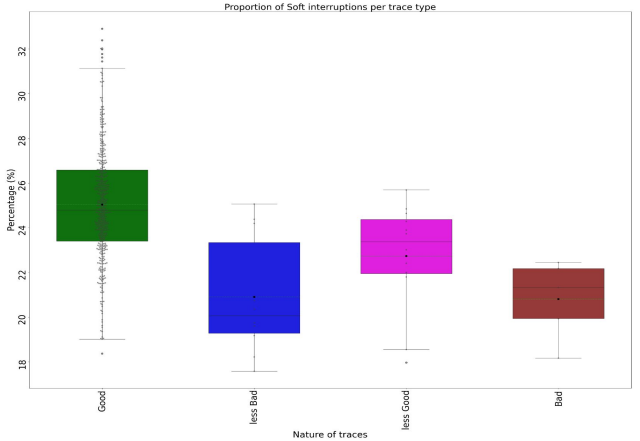
Normal traces have a higher latency and Abnormal traces wake-up the most

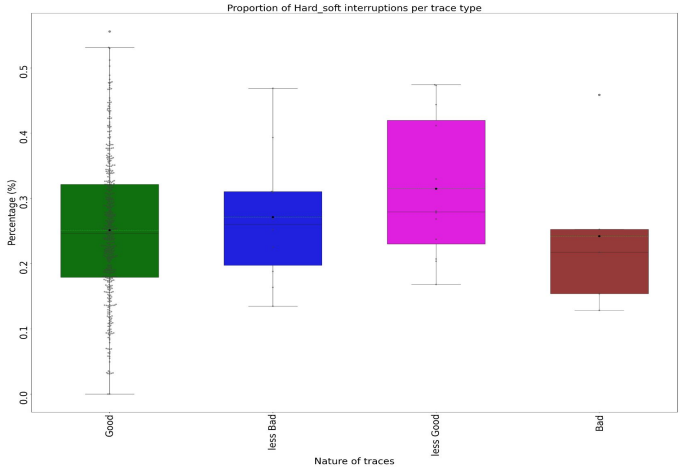




Are there more interruptions on Abnormal traces than on good ones?







- Hardware interruptions have a higher proportion on Abnormal traces.
- Software interruptions are lower on Abnormal traces.

Conclusion

- The problem was to understand what sometimes causes an application to take more execution time

- several approaches and methods in the search for a solution have been explored including classification, explicability, data analysis and data visualization.

- The data analysis method provided relevant hypotheses about what might actually cause an anomaly.

- For Future work, we are going to turn data analysis results into new future for an ML model

THANKS FOR YOUR ATTENTION !

WoCC'24

Women in CS Days Cameroon