Adaptive Tracing

By Masoumeh Nourollahi

21st January 2022

Michel DAGENAIS, Research supervisor
Naser EZZATI-JIVAN, Research co-supervisor

Introduction

Large scale tracing challenges

- Huge number of requests results in enourmous traces
- Puts overhead in trace collection, storage, and analysis
- Not much intelligence in collecting traces

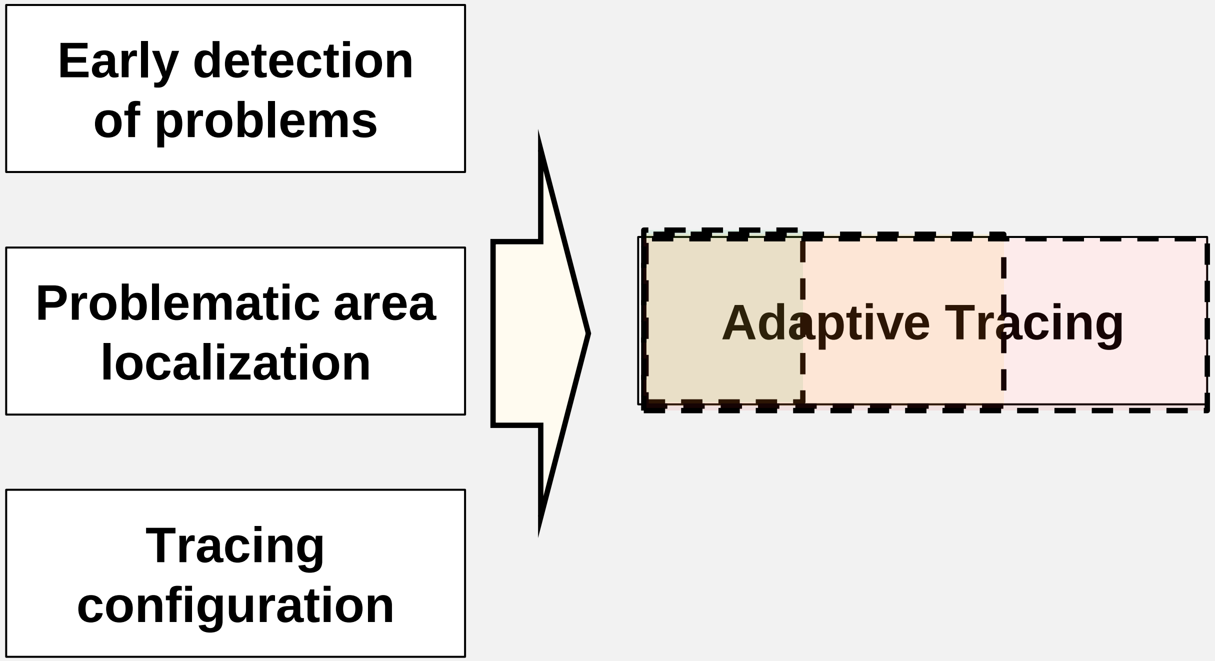
Thesis Statement

To improve tracing effectiveness, tracing focus should adjust and adapt to collecting relevant events around the issues.

Research question

1. Can we increase the tracing effectiveness using trace adjustment methods at runtime, so that tracing is more focused on collecting events around the issues?

2. Can we identify the possible problematic areas by analyzing workload and resource metrics?



Where to look for issues?

1. Workload changes
2. Application behavior changes
3. Code changes
4. Configuration changes
5. ...

Where to put tracepoints in complex large systems like Chrome or Trace compass?

- Too large to trace in detail
- Need a clever way to adapt and decide which events to collect

Case1:

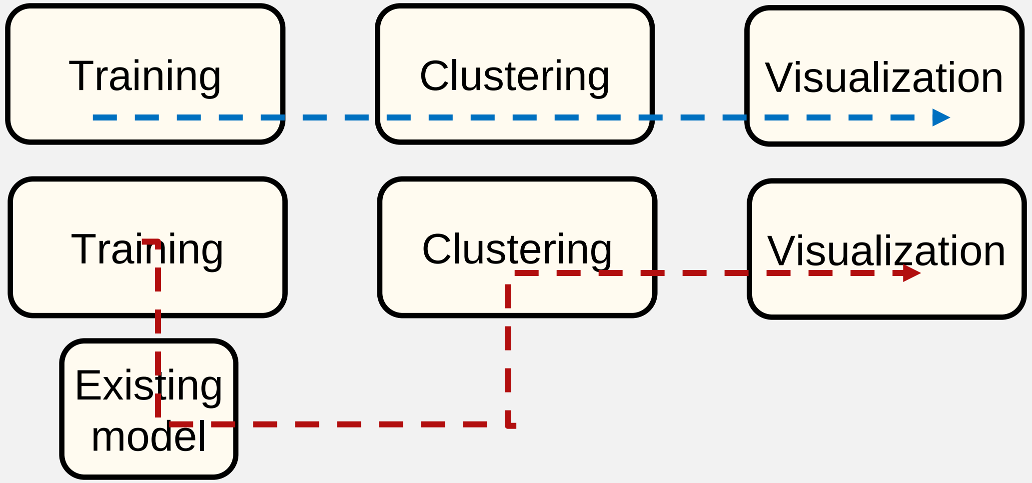
✔ All modules work well individually

X Visualization module slows down If the number of clusters > 100

Case2:

✔ scenario1: first time building the clustering model

X scenario2: updating an existing clustering model



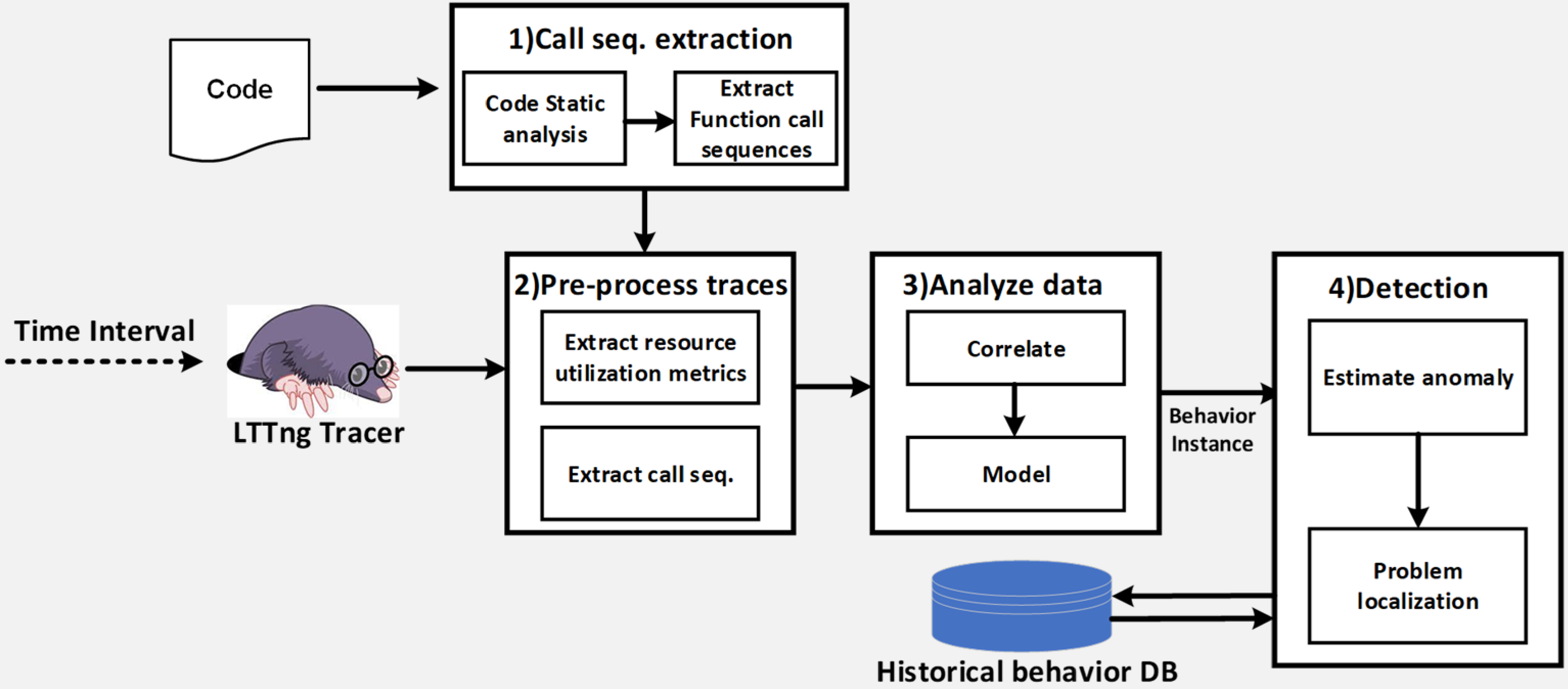
Problematic area localization

Goals

- Check if source of performance problem is internal or external to the system
- Know more precisely which module or scenario are possibly the source of an issue

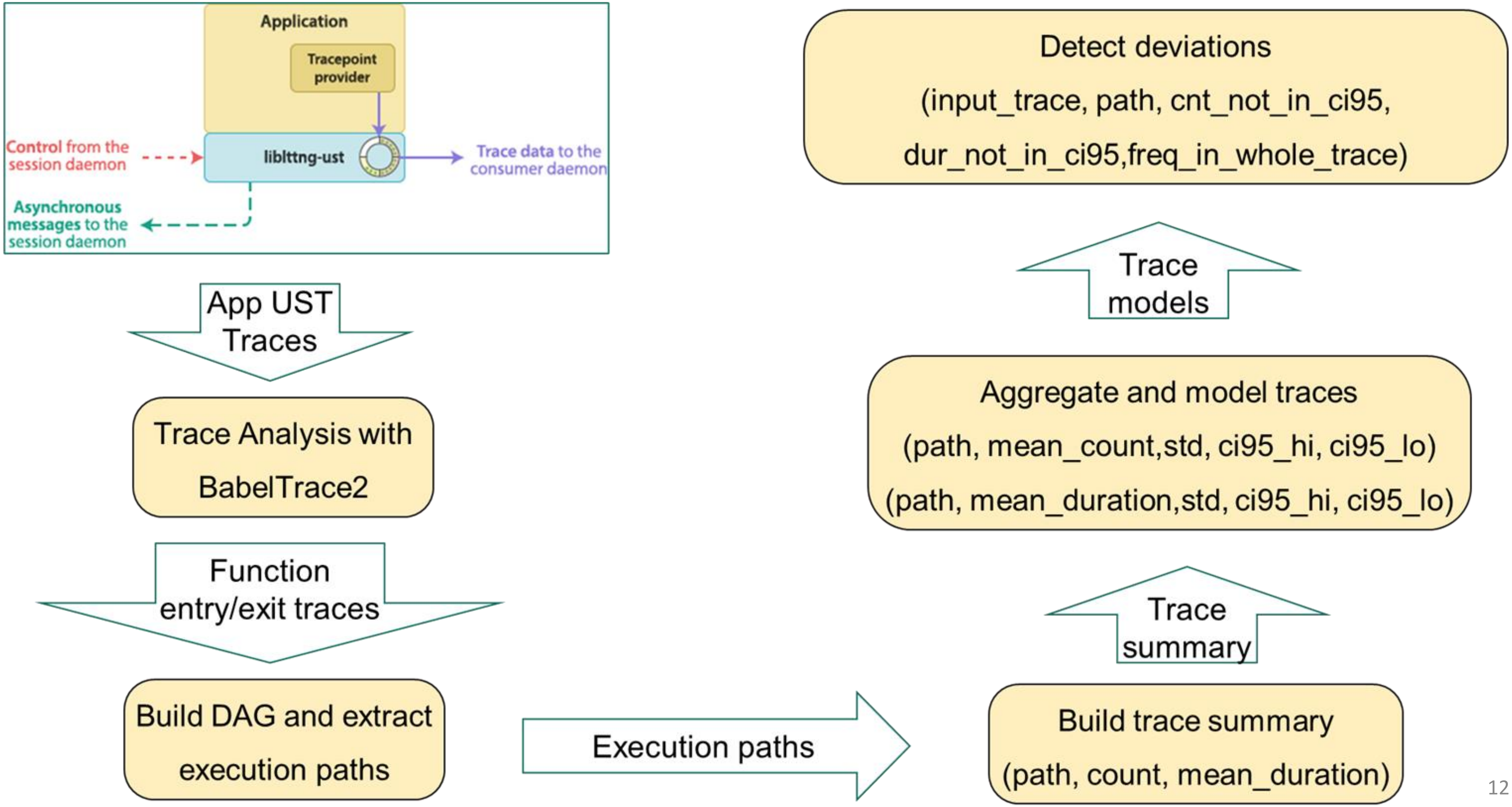
Similar workflows should perform similarly!

Problematic area localization process



Preliminary results

Adaptive tracing workflow



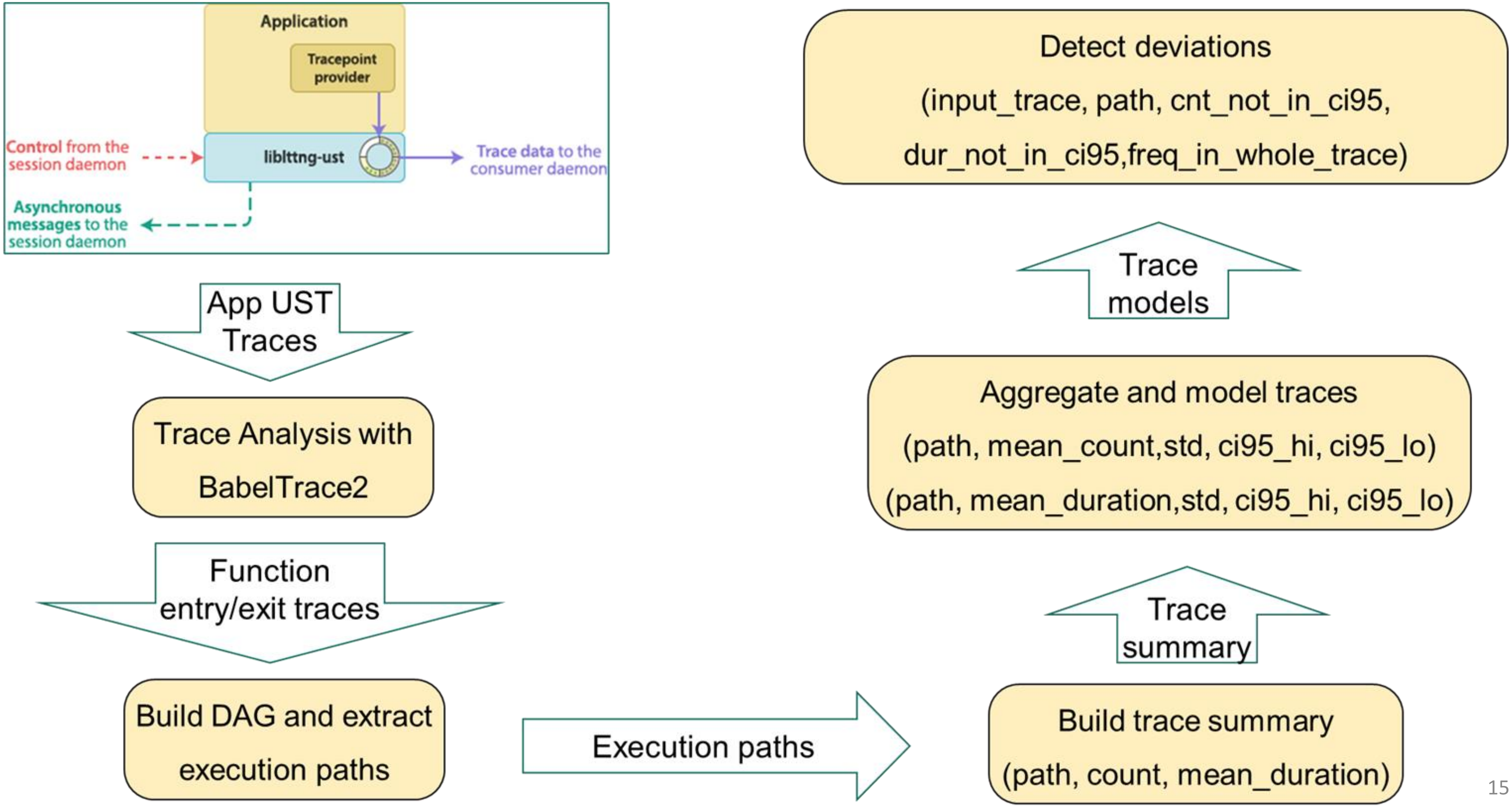
Tracepoint event definition

```
LTTNG_UST_TRACEPOINT_EVENT(
    masoum_ust_func,
    Prime_Interval,
    LTTNG_UST_TP_ARGS(
        const char* , event_type,
        const char* , file_name,
        const char* , func_name,
        long , loc
    ),
    LTTNG_UST_TP_FIELDS(
        lttng_ust_field_string(event_type_field, event_type)
         lttng_ust_field_string(file_name_field, file_name)
         lttng_ust_field_string(func_name_field, func_name)
         lttng_ust_field_integer(long, loc_field, loc)
    )
)
```

Tracepoint placement in code

```
void    Prime_Interval::write_to_file(void)
{
        lttng_ust_tracepoint(masoum_ust_func, write_to_file, "en", __FILE__, __FUNCTION__, __LINE__);
        output.open("result.txt");
        output << "<root>\n\t";
        output << "<primes>";
        for (int i = 0; i < primes.size(); i++)
        {
                output << " " << primes[i];
        }
        output << " </primes>\n";
        output << "</root>";
        output.close();
        lttng_ust_tracepoint(masoum_ust_func, write_to_file, "ex", __FILE__, __FUNCTION__, __LINE__);
}
```

Adaptive tracing workflow

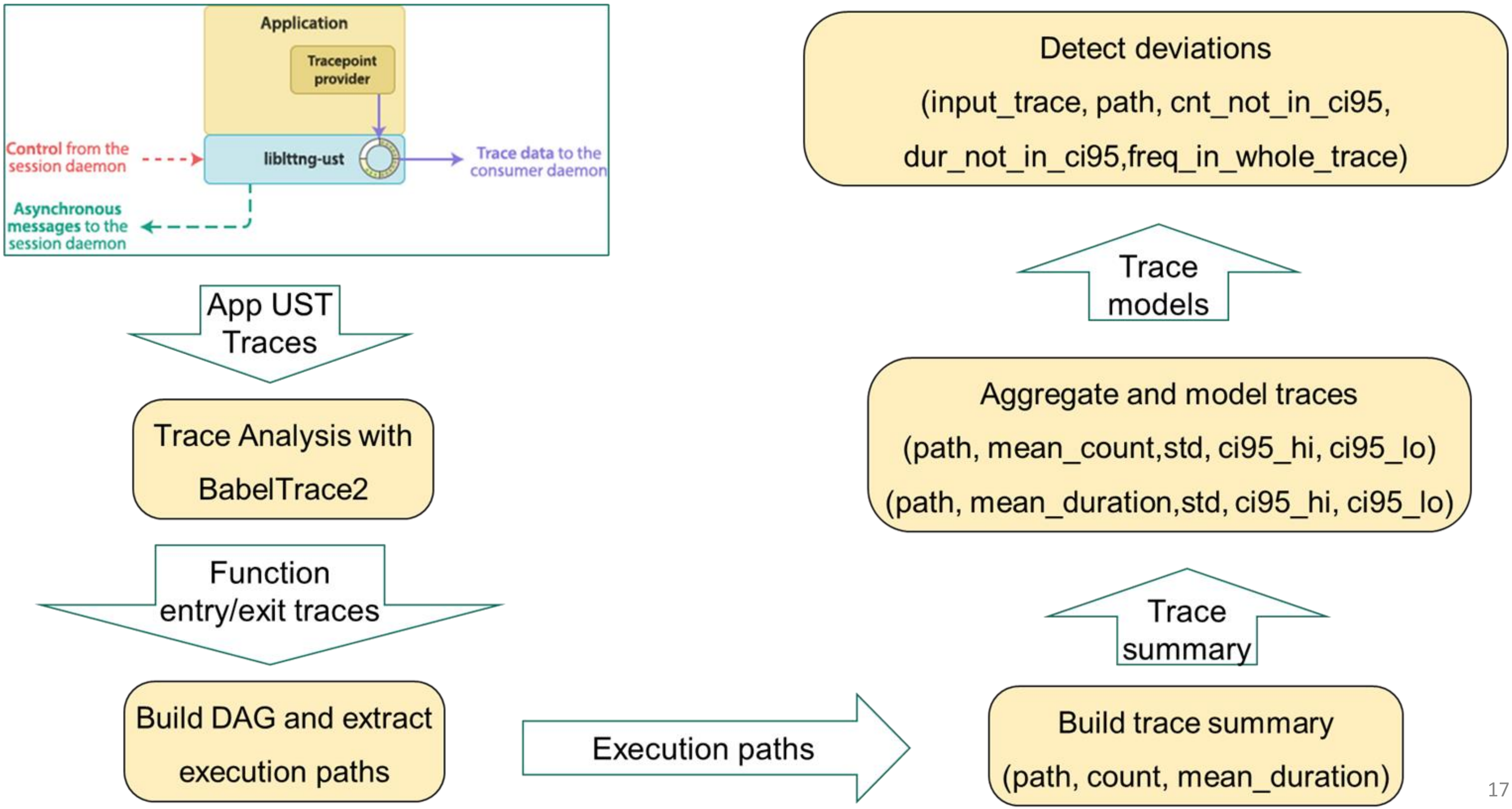


Trace analysis

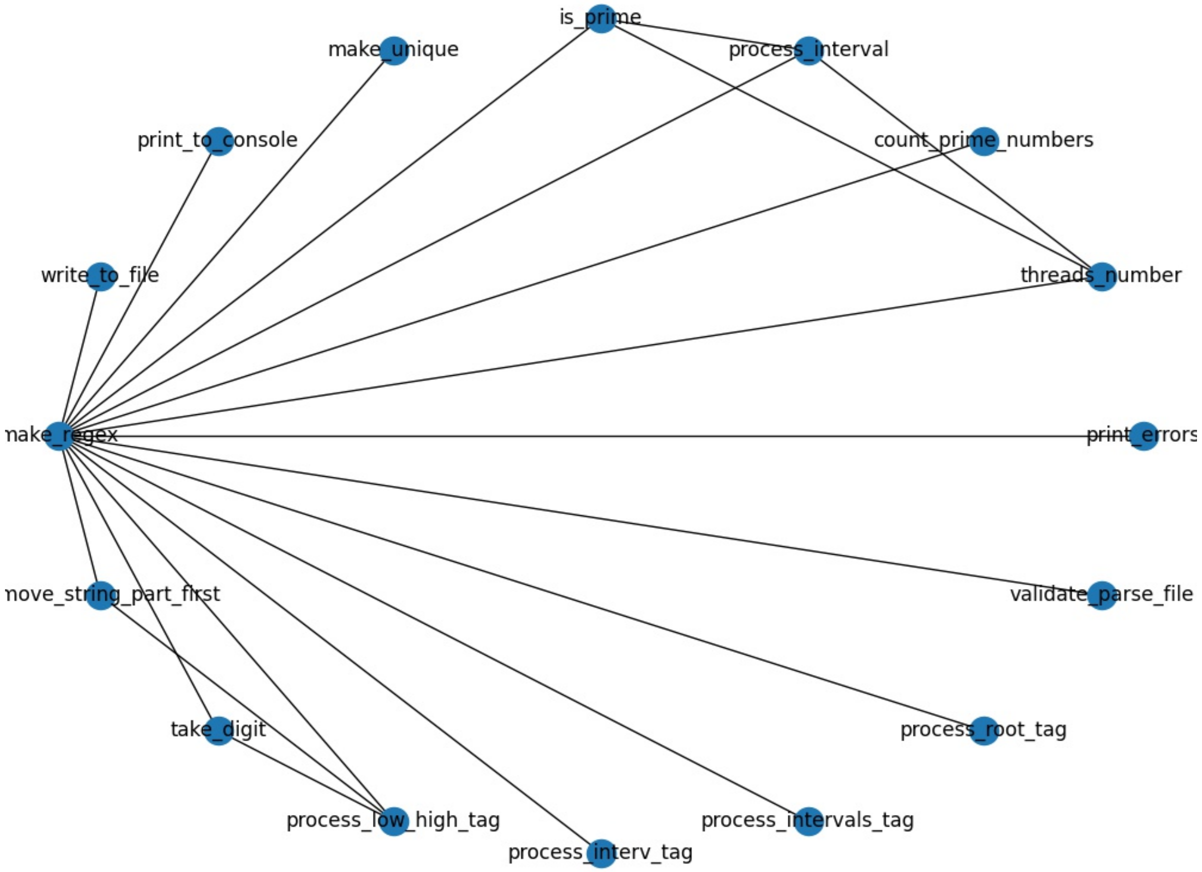
e_name,e_type,entry_ts,exit_ts,file_func,func_func,loc_func,caller_row,caller_file,caller_func

masoum_ust_func:Prime_Interval,en,1642298709193087671,1642298709193089768,Prime_Interval.cpp,Prime_Interval,11,0,prime_numbers_and_threads.cpp,main

Adaptive tracing workflow



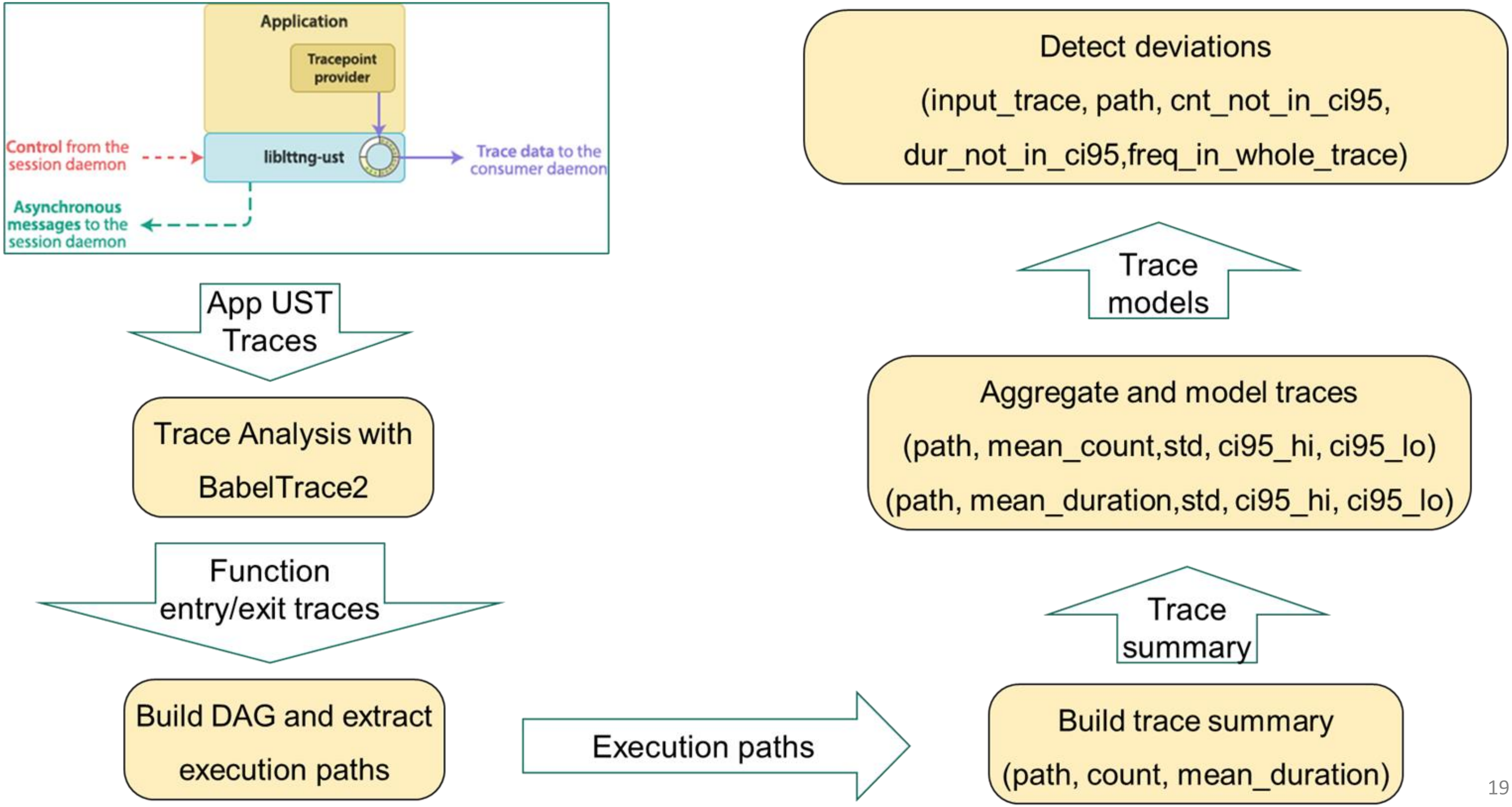
Extracting execution paths from the DAG



DAG for sample program

```
graph = nx.DiGraph()
paths_df= dag_to_branching(graph)
nx.dag_longest_path(graph)
```

Adaptive tracing workflow

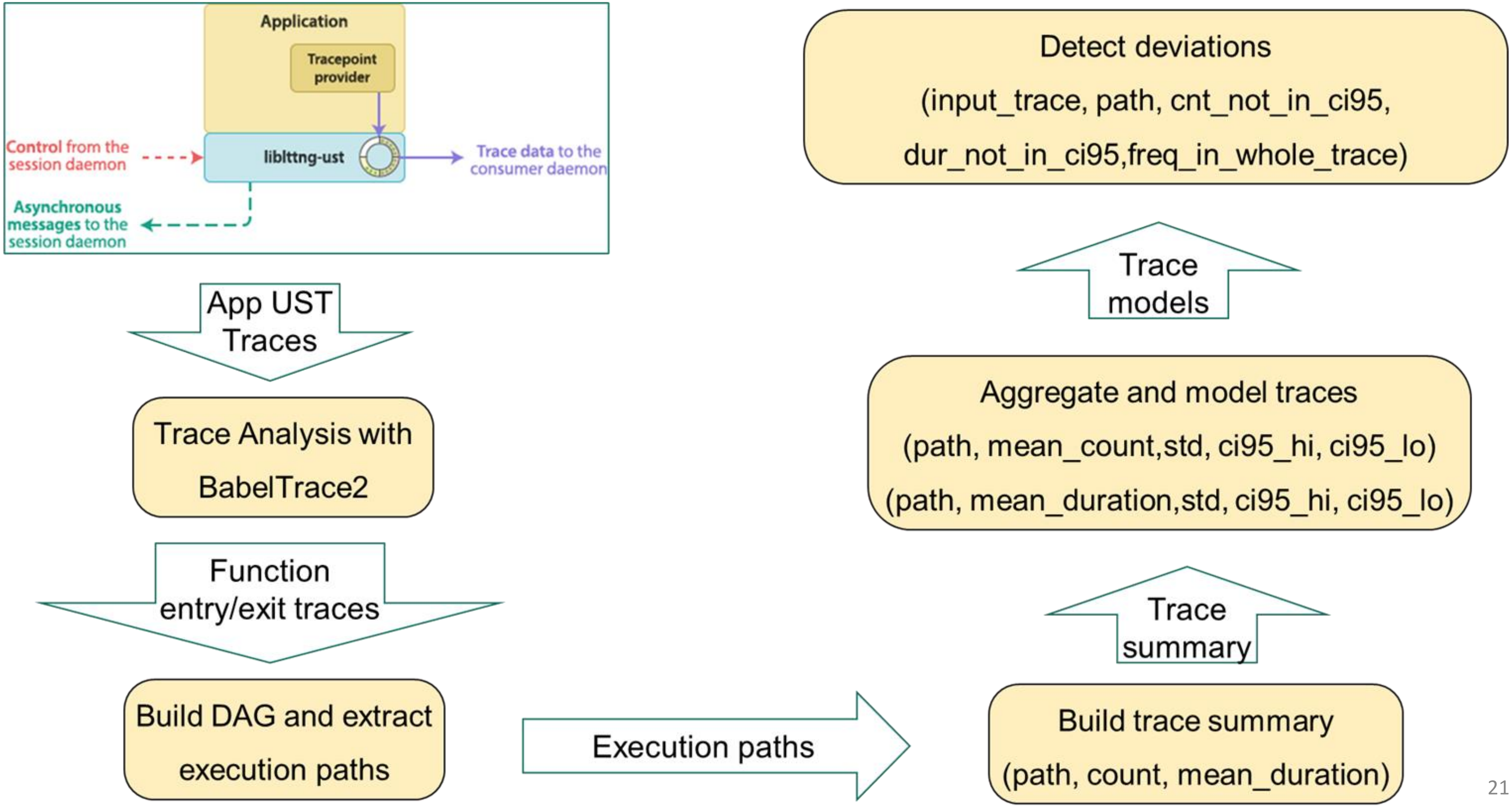


Trace summary generated for each edge

caller,callee,count,mean_duration

masoum_ust_func:count_prime_numbers:Prime_Interval.cpp100count_prime_numbers,masoum_ust_func:count_prime_numbers:Prime_Interval.cpp100count_prime_numbers,6,419.5

Adaptive tracing workflow



Trace analysis- Metrics and Actions

Metrics:

- Frequency
  - Condition1: check if occurrence frequency of the path is within confidence interval 95 high and low of its previous execution
- Duration
  - Condition2: check if response time (duration) of the path is within confidence interval 95 high and low of its previous executions
- Ratio of frequency to whole frequency
  - Condition3: Check if frequency of the path is more than 60% of the whole frequencies

Adjustment actions:

- ★ If Condition1 | condition2:
  - enable more tracepoints to observe in detail
- ★ If Condition3:
  - disable tracepoint

Path frequncy analysis:

edge,mean,count,std,ci95_hi,ci95_lo

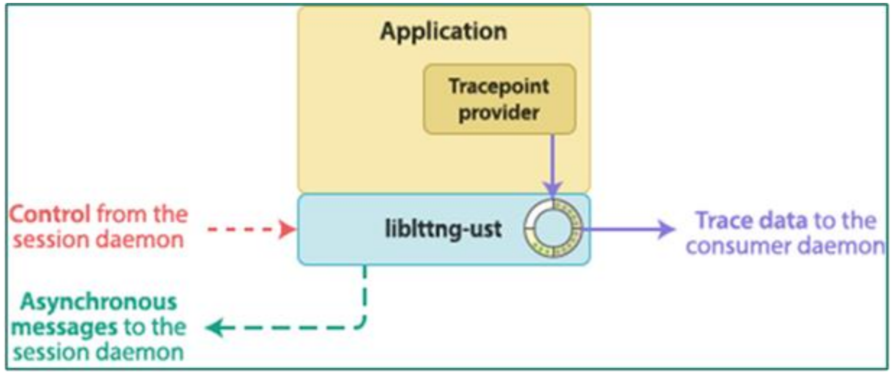
masoum_ust_func:count_prime_numbers:Prime_Interval.cpp100count_prime_numbersmasoum_ust_func:count_prime_numbers:Prime_Interval.cpp100count_prime_numbers,6.166666666666667,12,0.9374368665610919,6.697071505378846 5,5.636261827954487

Path duration analysis:

edge,mean,count,std,ci95_hi,ci95_lo

masoum_ust_func:count_prime_numbers:Prime_Interval.cpp100count_prime_numbersmasoum_ust_func:count_prime_numbers:Prime_Interval.cpp100count_prime_numbers,491.615873015873,12,144.4812515462643 3,573.3638366995009,409.867909332245

Adaptive tracing workflow



App UST Traces

Trace Analysis with BabelTrace2

Function entry/exit traces

Build DAG and extract execution paths

Execution paths

Build trace summary
(path, count, mean_duration)

Trace summary

Aggregate and model traces
(path, mean_count,std, ci95_hi, ci95_lo)
(path, mean_duration,std, ci95_hi, ci95_lo)

Trace models

Detect deviations
(input_trace, path, cnt_not_in_ci95, dur_not_in_ci95,freq_in_whole_trace)

Trace analysis in comparison to trace history

input_trace,edge,count,mean_duration,cnt_not_in_than_ci95,dur_not_in_than_ci95,freq_in_whole_trace

070611.csv,masoum_ust_func:count_prime_numbers:Prime_Interval.cpp100count_prime_numbersmasoum_ust_func:count_prime_numbers:Prime_Interval.cpp100count_prime_numbers,7,414.7142857142857,0,1,0

Adapting the tracing

lttng enable-event --userspace masoum_ust_func:'*' --filter='masoum_ust_func:is_prime"

Frequency stats:
masoum_ust_func:is_prime:Prime_Interval.cpp38is_primemasoum_ust_func:is_prime:Prime_Interval.cpp38is_prime,10538.7,10,27541.96480544633,27609.386644195707,-6531.986644195706
Duration stats:
masoum_ust_func:is_prime:Prime_Interval.cpp38is_primemasoum_ust_func:is_prime:Prime_Interval.cpp38is_prime,16863.981222495084,10,46178.043086506725,45485.428854874684,-11757.46640988452

Aggregated record:
masoum_ust_func:is_prime:Prime_Interval.cpp38is_primemasoum_ust_func:is_prime:Prime_Interval.cpp38is_prime,2225,3951.713707865169,0,0,1

- “is_prime” tracepoint takes most of the execution and causes the trace to become very large
- It shows consistent response time and its frequency of execution is dependant on the input data and can cause exponential growth in trace for some input data
- Best way here is to disable tracing this function or reduce its sampling rate

The way forward

- Provide automated pipeline for the presented method
- Test and extend the method for a more complex application with longer execution paths
- Check other possible methods to model frequency and duration metric
- Implement the same for other metrics like resource-related metrics modeling in combination with UST trace metrics
- Improve performance of the code

References

[1] Tânia Esteves, Francisco Neves, Rui Oliveira, and João Paulo. 2021. CAT: content-aware tracing and analysis for distributed systems. In Proceedings of the 22nd International Middleware Conference (Middleware '21). Association for Computing Machinery, New York, NY, USA, 223–235.

[2] S. Zhang, D. Liu, L. Zhou, Z. Ren, and Z. Wang, “Diagnostic framework for distributed application performance anomaly based on adaptive instrumentation,” in 2020 2nd International Conference on Computer Communication and the Internet (ICCCI). IEEE, 2020, pp. 164–169.

[3]E. Ates, L. Sturmann, M. Toslali, O. Krieger, R. Megginson, A. K. Coskun, and R. R. Sambasivan, “An automated, cross-layer instrumentation framework for diagnosing performance problems in distributed applications,” in Proceedings of the ACM Symposium on Cloud Computing, ser. SoCC ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 165–170.

[4] P. Las-Casas, J. Mace, D. Guedes, and R. Fonseca, “Weighted sampling of execution traces: Capturing more needles and less hay,” in Proceedings of the ACM Symposium on Cloud Computing, 2018, pp. 326–332.

[5] P. Las-Casas, G. Papakerashvili, V. Anand, and J. Mace, “Sifter: Scalable sampling for distributed traces, without feature engineering,” in Proceedings of the ACM Symposium on Cloud Computing, 2019, pp. 312–324.

[6] A. Bento, J. Correia, R. Filipe, F. Araujo, and J. Cardoso, “Automated analysis of distributed tracing: Challenges and research directions,” Journal of Grid Computing, vol. 19, no. 1, pp. 1–15, 2021.

[7] Q. Fournier, N. Ezzati-jivan, D. Aloise, and M. R. Dagenais, “Automatic cause detection of performance problems in web applications,” in 2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). IEEE, 2019, pp. 398–405.

[8] M. Dymshits, B. Myara, and D. Tolpin, “Process monitoring on sequences of system call count vectors,” in 2017 International Carnahan Conference on Security Technology (ICCST). IEEE, 2017, pp. 1–5.

[9] F. Doray and M. Dagenais, “Diagnosing performance variations by comparing multilevel execution traces,” IEEE Transactions on Parallel and Distributed Systems, vol. 28, no. 2, pp. 462–474, 2016.

[10] 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, Inc., Tech. Rep., 2010.

References

[11] J. Kaldor, J. Mace, M. Bejda, E. Gao, W. Kuropatwa, J. O’Neill, K. W. Ong, B. Schaller, P. Shan, B. Viscomi, V. Venkataraman, K. Veeraraghavan, and Y. J. Song, “Canopy: An end-to-end performance tracing and analysis system,” in Proceedings of the 26th Symposium on Operating Systems Principles, ser. SOSP ’17. New York, NY, USA: Association for Computing Machinery, 2017, p. 34–50.

[12] Wert, Alexander, Jens Happe, and Lucia Happe. "Supporting swift reaction: Automatically uncovering performance problems by systematic experiments." 2013 35th International Conference on Software Engineering (ICSE). IEEE, 2013.

[13] M. Gebai et M. R. Dagenais, “Survey and analysis of kernel and userspace tracers on linux : Design, implementation, and overhead,” ACM Comput. Surv., vol. 51, no. 2, mars 2018.

[14] S. Tjandra, “Performance model extraction using kernel event tracing,” Thèse de doctorat, Carleton University, 2019.

[15] R. R. Sambasivan, A. X. Zheng, M. De Rosa, E. Krevat, S. Whitman, M. Stroucken, W. Wang, L. Xu et G. R. Ganger, “Diagnosing performance changes by comparing request flows.” dans NSDI, vol. 5, 2011, p. 1–1.

[16] Du, Min, et al. "Deeplog: Anomaly detection and diagnosis from system logs through deep learning." Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017.

[17] V. Cortellessa et L. Traini, “Detecting latency degradation patterns in service-based systems,” dans Proceedings of the ACM/SPEC International Conference on Performance Engineering, 2020, p. 161–172.

[18] F. Doray et M. Dagenais, “Diagnosing performance variations by comparing multi-level execution traces,” IEEE Transactions on Parallel and Distributed Systems, vol. 28, no. 2, p. 462–474, 2016.