WWU
MÜNSTER

Performance Evaluation of a Legacy Real-Time System: An Improved RAST Approach

Juri Tomak, Adrian Liermann, and Sergei Gorlatch

University of Muenster, Germany

Introduction

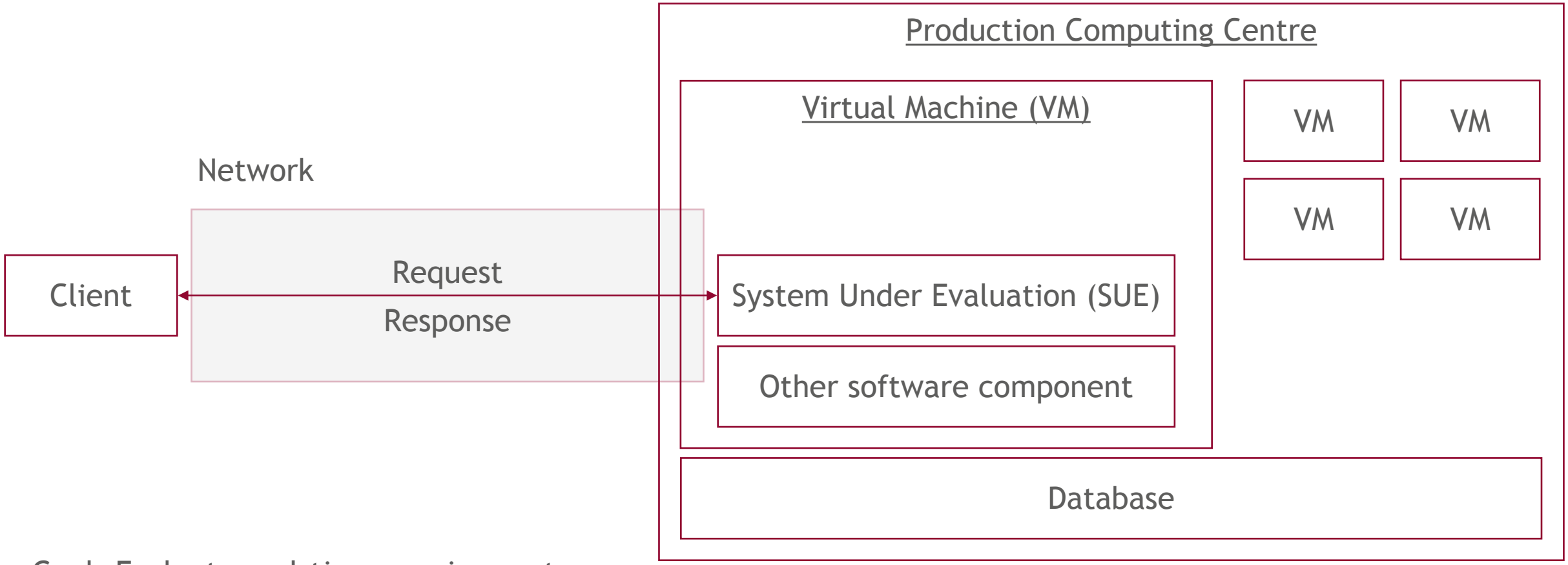
Juri Tomak

- Dual affiliation:
  - Ph.D. student | University of Muenster, Germany
  - Team leader / Software Engineer | GS electronic GmbH, Germany

- Today's agenda:
  - Why is evaluating the performance of a legacy, production system challenging?
  - What is the RAST (Regression Analysis, Simulation, and load Testing) toolset?
  - What is improved compared to our initial, proof-of-concept implementation?
  - Our experimental results using our RAST toolset for the particular use case of an industrial alarm system.

Problem: Performance Evaluation of a Legacy Production System

Distributed Software System running on a number of virtual machines in production



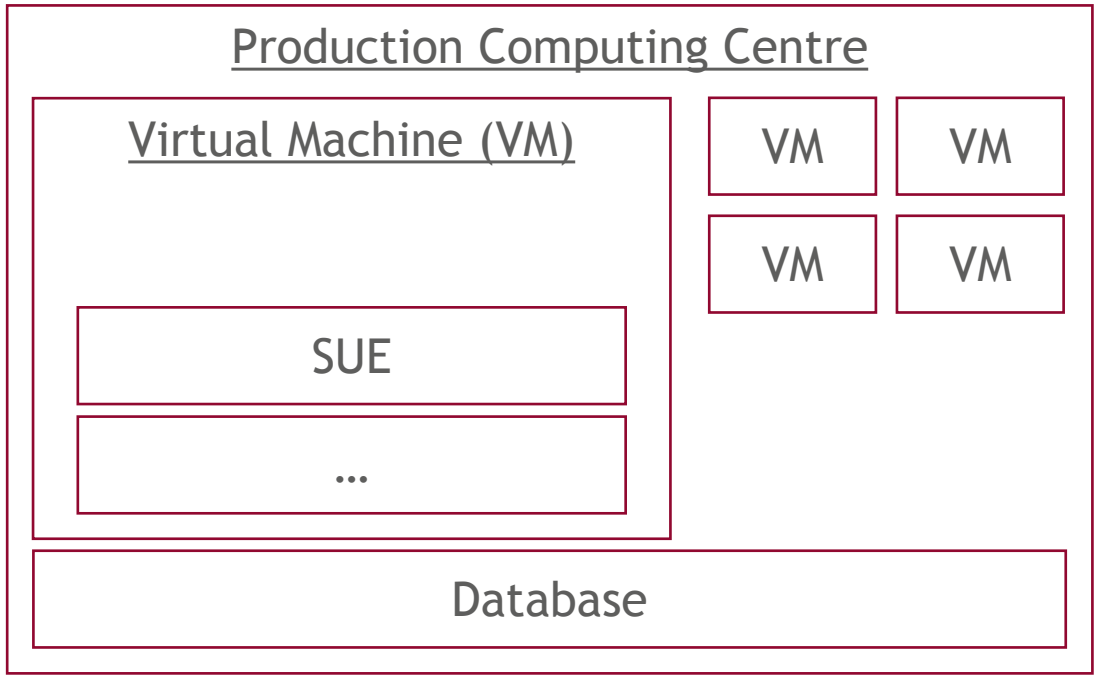
- Goal: Evaluate real-time requirements regarding the response time[1].

[1] Response time is the time it takes between sending a request and receiving its response. It is the sum of the SUE's processing time and network latency.

Properties of our Class of Systems

Our Example: Commercial Alarm System

- Distributed
- Database-centric
- Runs in virtual machines
- Large (> 1.5 Million Lines of Code)
- Legacy
- Real-Time Requirements
- Mission-critical



With our work, we address the following challenges that exist in systems with these properties.

Challenges in our Class of Systems

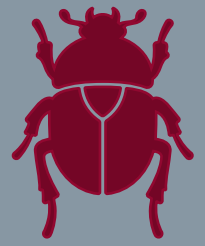
CHALLENGES

Disrupting regular operation is not allowed

Real-Time Requirements

No APM Tool

No Test Environment



Bugs

Active Development

No Documentation

Resource-Sharing Environment

Challenges that prevent traditional load testing

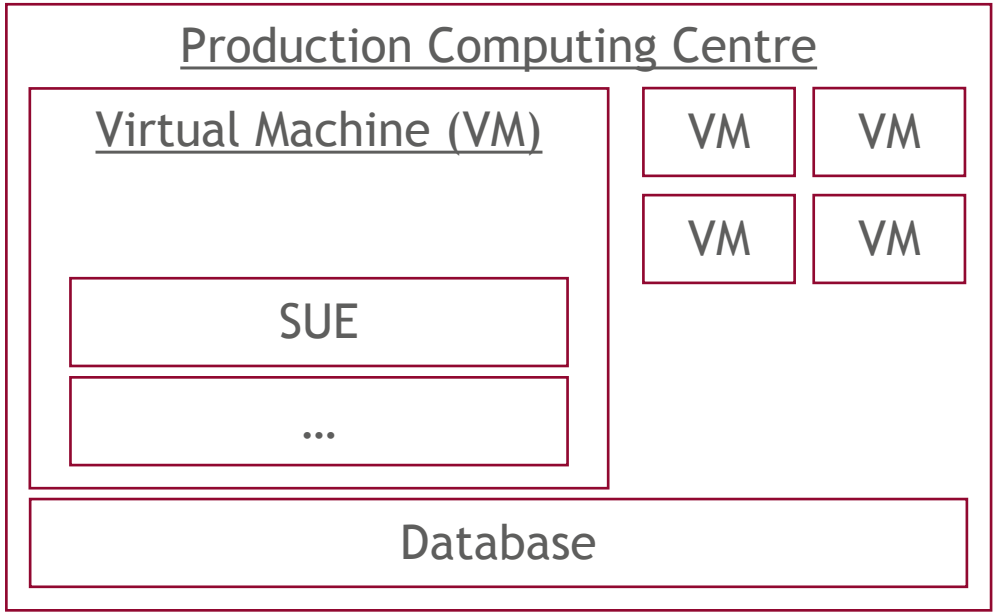
[1] Jin 2007

- Load testing is not feasible: it disrupts regular operation by consuming computing resources and possibly changing the contents of the database.
- Load testing in a test environment is not applicable: test environment has less computing power than the production environment [1].

Challenges that prevent traditional performance modeling

[1] Jin 2007
[2] Sharma 2016

- Resource usage of programs that run in parallel to the System Under Evaluation (SUE) affects performance evaluation results (*Noisy Neighbor*) [1], [2].
- Therefore, it is necessary to model the whole system, even if only part of it is subject to evaluation.



Challenges that prevent traditional performance modeling

[3] Tomak 2021

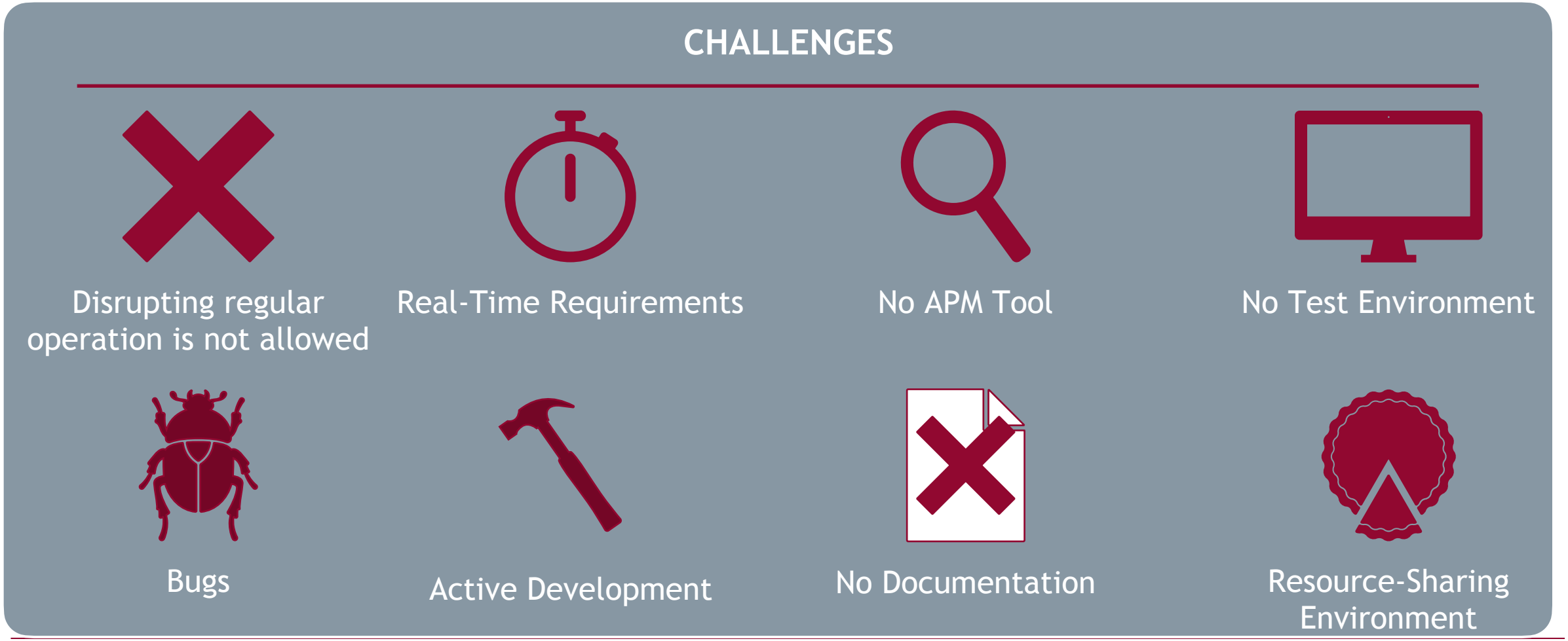
- Performance modeling is very difficult: existing bugs, continuous development, and the lack of documentation will likely make the model not represent the system well enough, especially as time passes [3].

Related Work

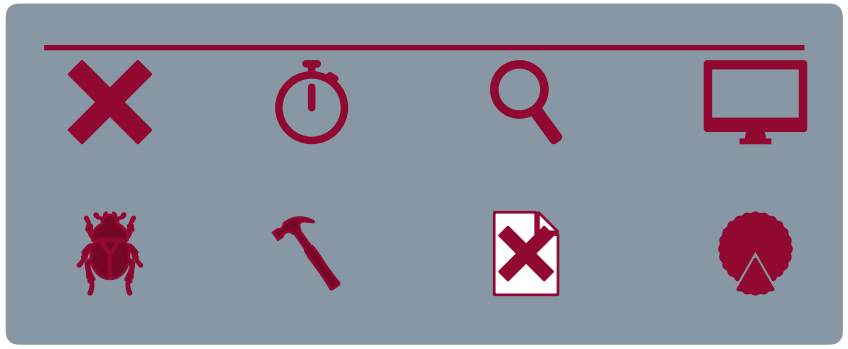
- Several similar approaches use regression analysis to build a predictive model of the target system. They collect low-level metrics, like resource consumption and service demand, using Application Performance Monitoring (APM) tools [4], [5], [6], [7].
- In our class of systems, we have no such tools and it is not possible to integrate them.

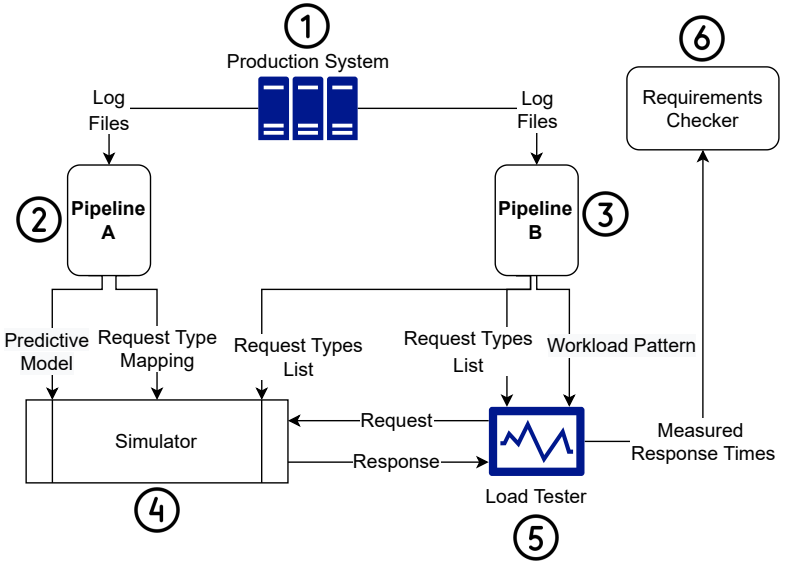
[4]: Okanovi´c 2012
[5]: Grohmann 2019
[6]: Courageux-Sudan 2021
[7]: Aichernig 2019

Challenges in our Class of Systems



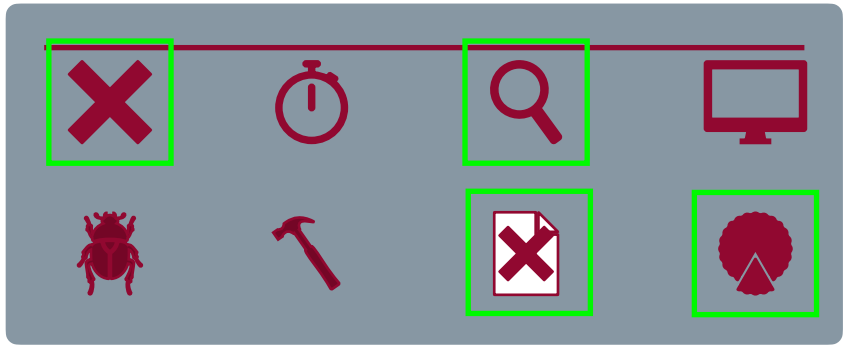
Our Approach to Performance Evaluation

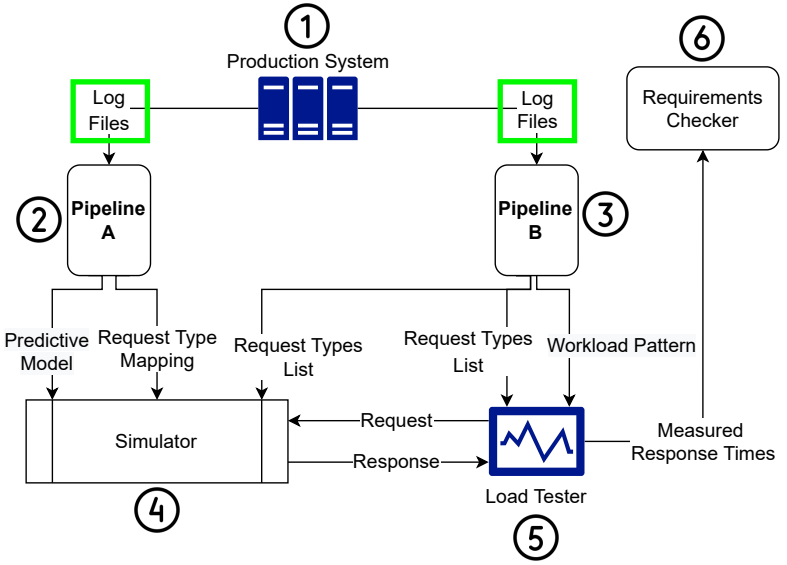




- Regression Analysis, Simulation, and load Testing (RAST).
- RAST has similarities with previous work in using regression analysis to create a predictive model for processing time.
- The combination of regression analysis, simulation and load testing results in features that help us in addressing our challenges.

Using the Available Log Files





1) RAST uses the available request / access log files of the production system as input for regression analysis.

Challenges addressed:

➡ no APM tool,

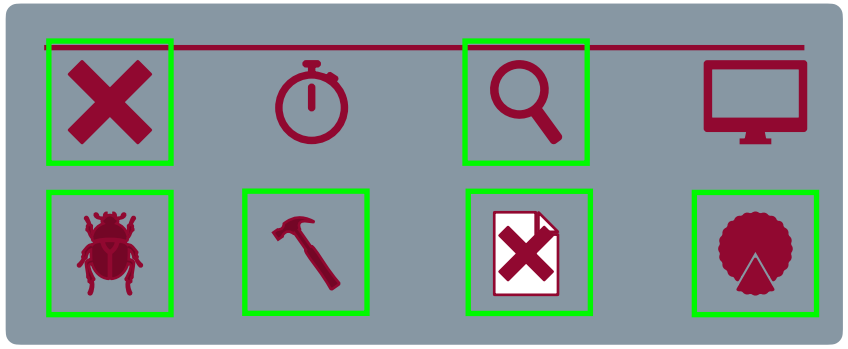
➡ no documentation,

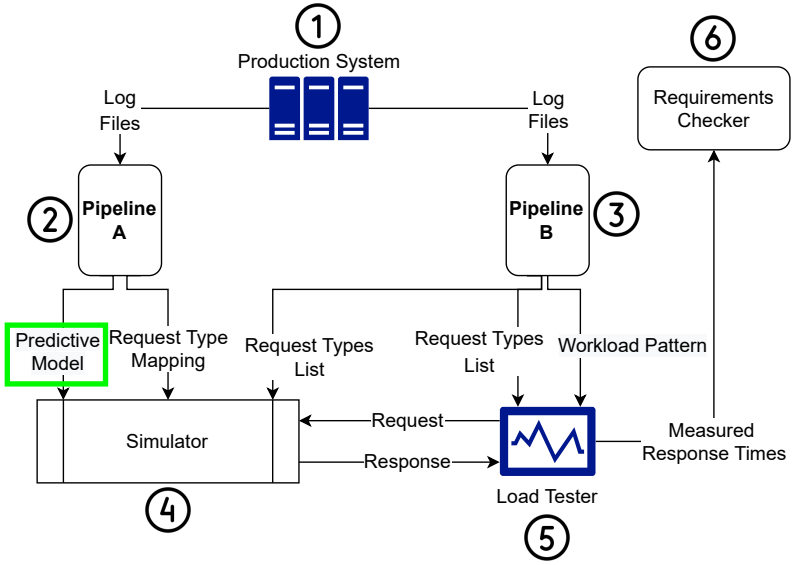
➡ resource-sharing environment,

➡ disrupting regular operation is not allowed.

Finding the Optimal Predictive Model

[8] Scikit-learn Development Team





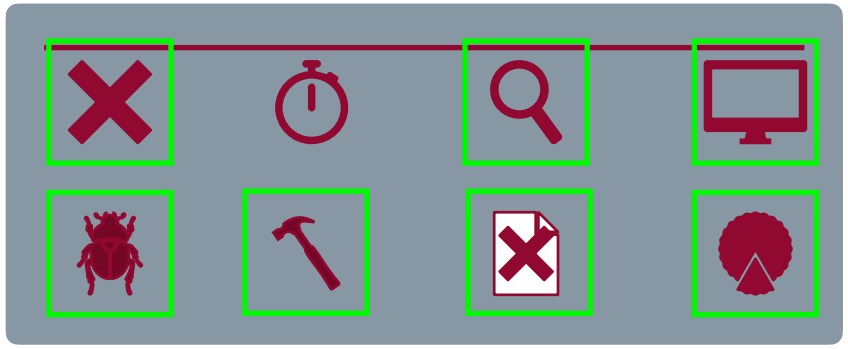
2) RAST automatically finds the optimal predictive model for the target system based on the provided log files by choosing the best-performing regression algorithm via cross-validation of common regression algorithms [8], such as: Linear, Elastic net, and Decision tree regression.

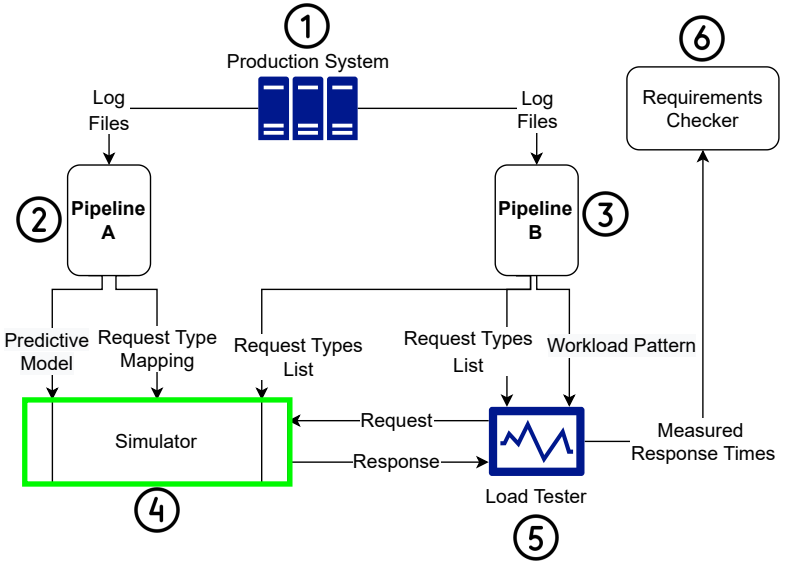
Challenges addressed:

➡ bugs,

➡ active development.

Simulating the System





3) RAST simulates the system as a server software that receives the same requests as the real legacy software and sends valid responses.

Challenges addressed:

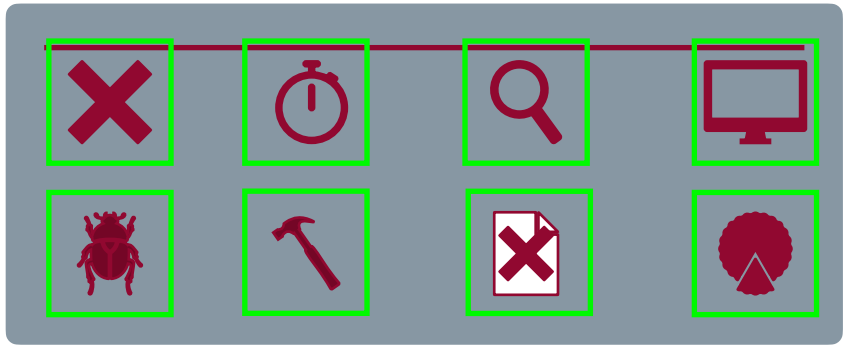
➡ no test environment,

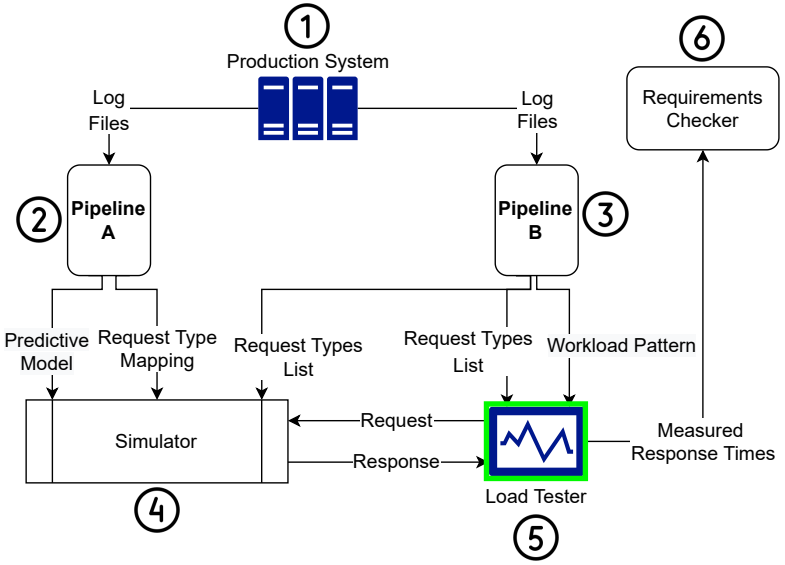
➡ disrupting regular operation is not allowed.

Extra:

➡ existing load testing tools.

Load Testing with Production Workload





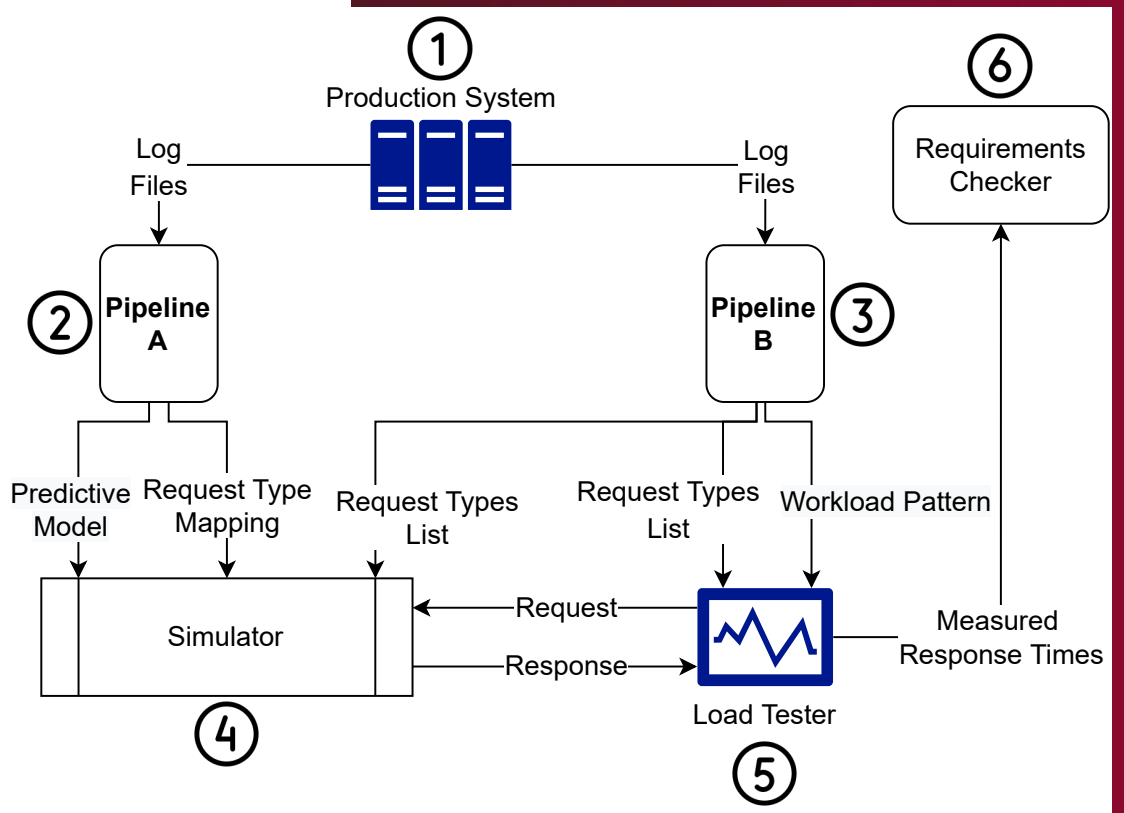
4) RAST uses load testing to submit the same workload to the simulation that the production system is processing.

Challenges addressed:

➡ real-time requirements.

RAST

Regression Analysis, Simulation and load Testing

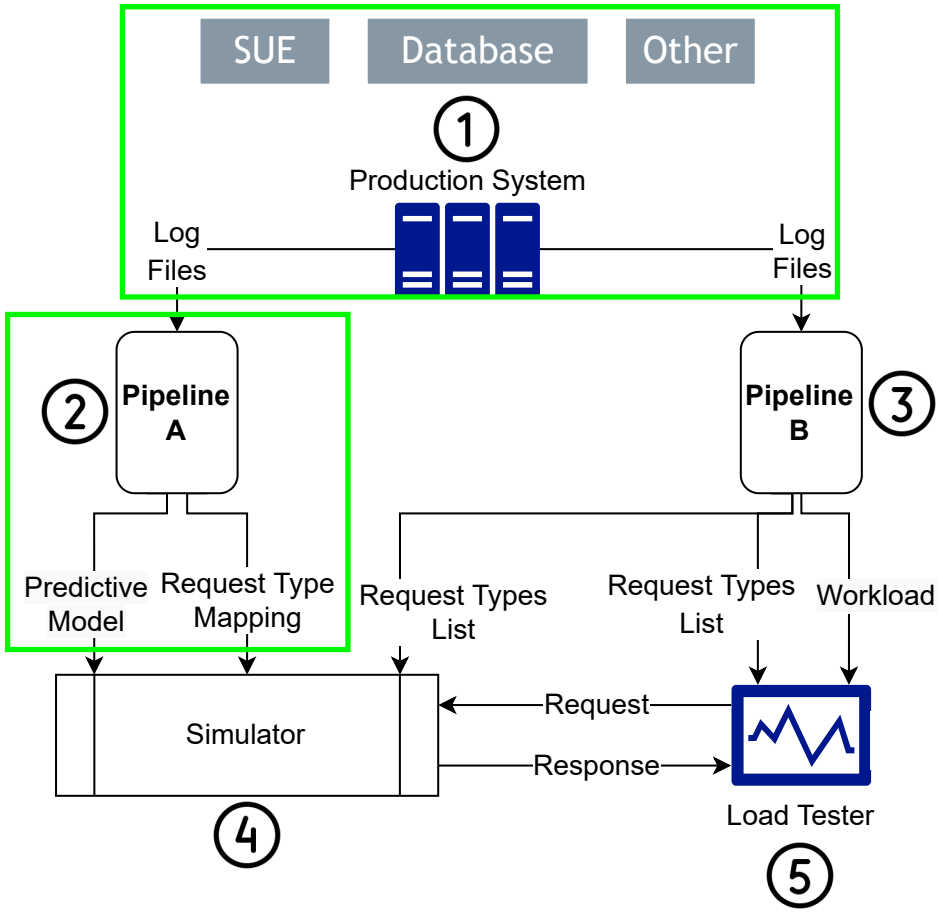


RAST Components

① The Production System provides log files to Pipelines A and B with information regarding the requests the system has received and processed.

② Pipeline A creates the predictive model and request type mapping.

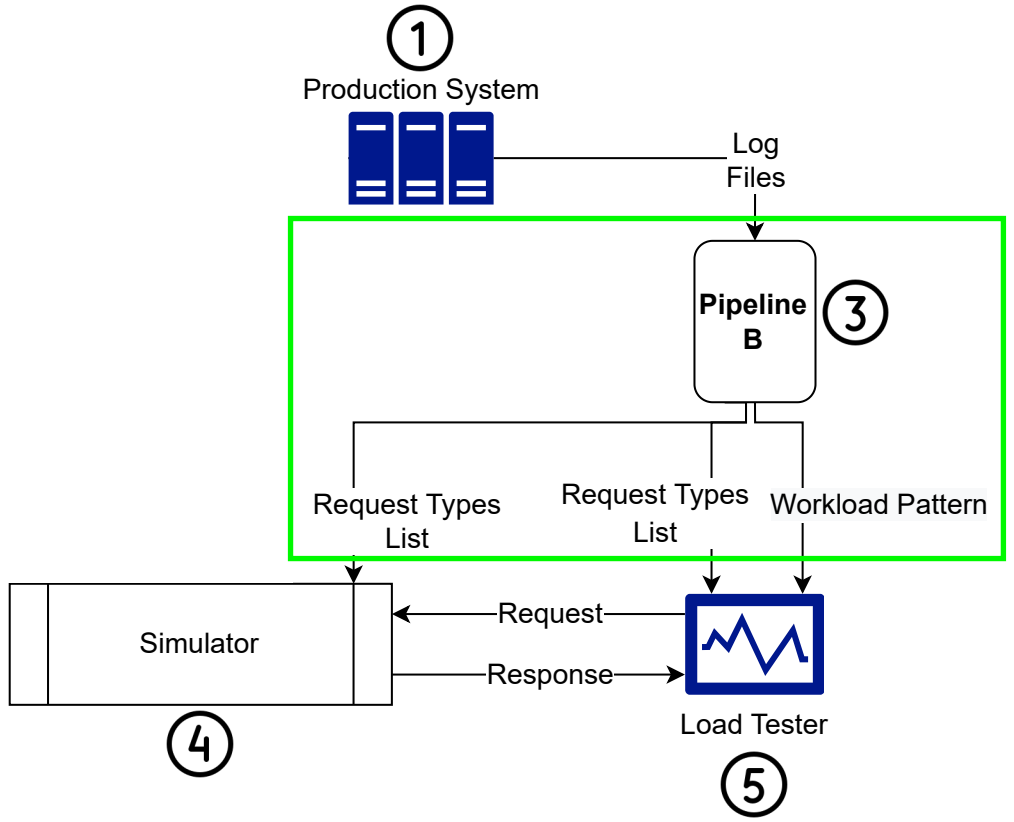
- Predictive model: a regression algorithm and its parameters.
- Request type mapping: transforms textual request types into numerical values.



RAST Components

③ Pipeline B creates the request types list and the workload pattern.

- Request types list: all requests of the whole production system.
- Workload pattern: the number and types of requests the system processed in a given time interval, e.g., the number of requests per hour.

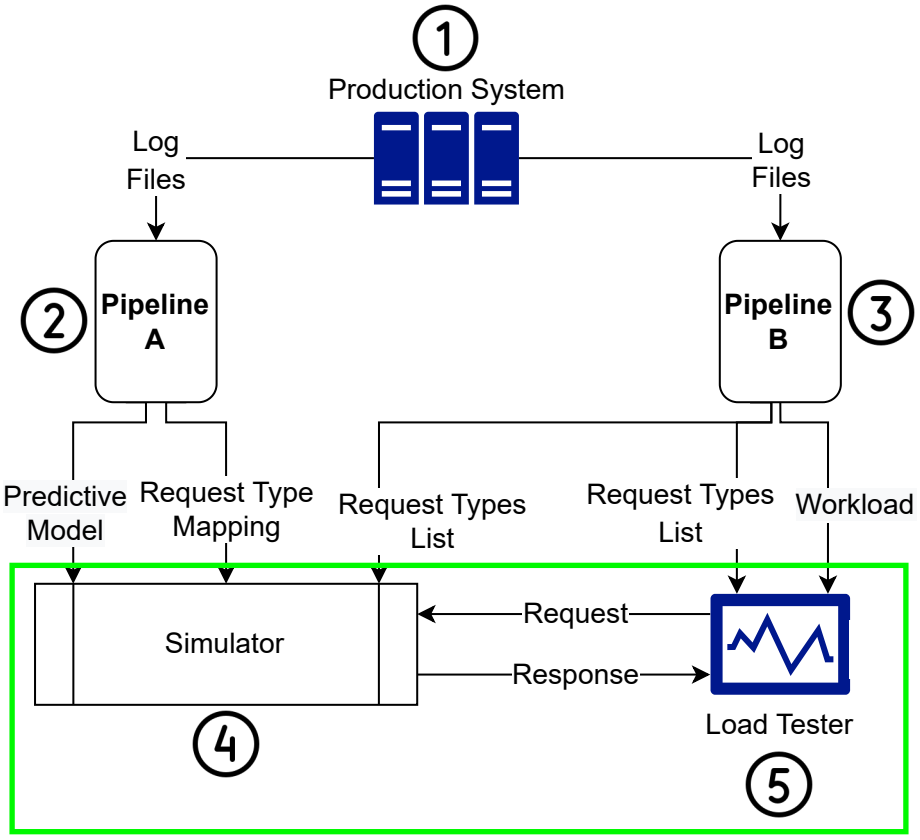


RAST Components

④ The Simulator is a server software that processes all requests that the production system accepts.

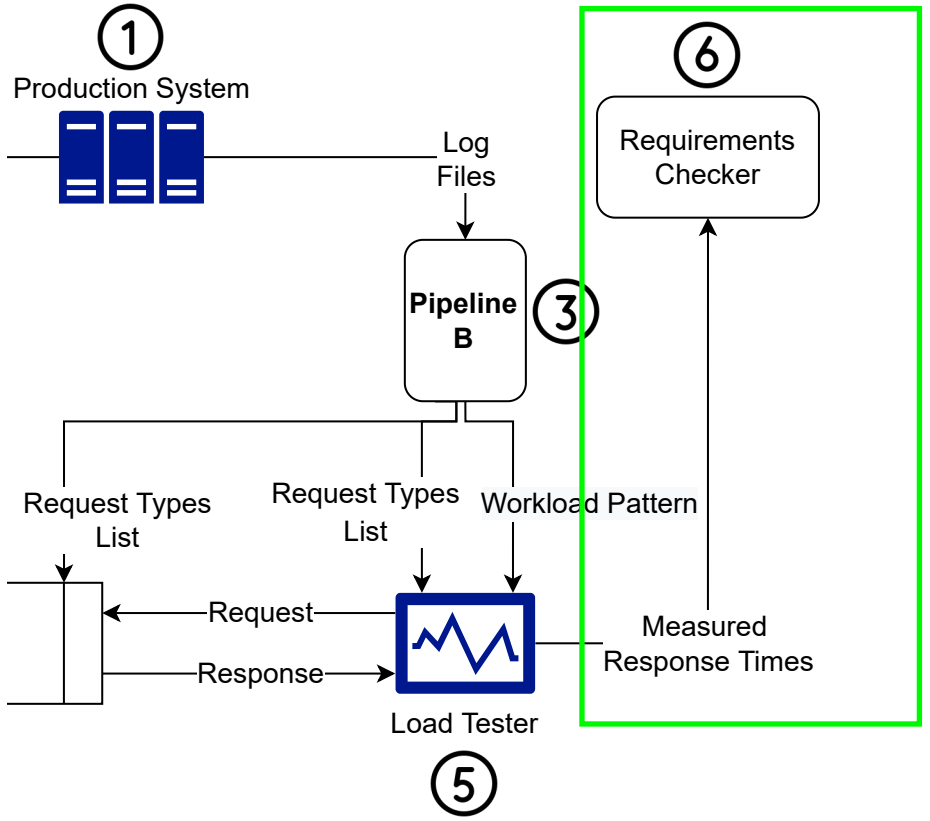
- Request type verification and transformation.
- Processing time simulation.

⑤ The Load Tester generates a synthetic workload and measures the response times of the Simulator.



RAST Components

⑥ Requirements Checker verifies if the measured response times fit within the requirements and reports the result to the Load Tester.



Details of our implementation are given in our paper.

RAST Components

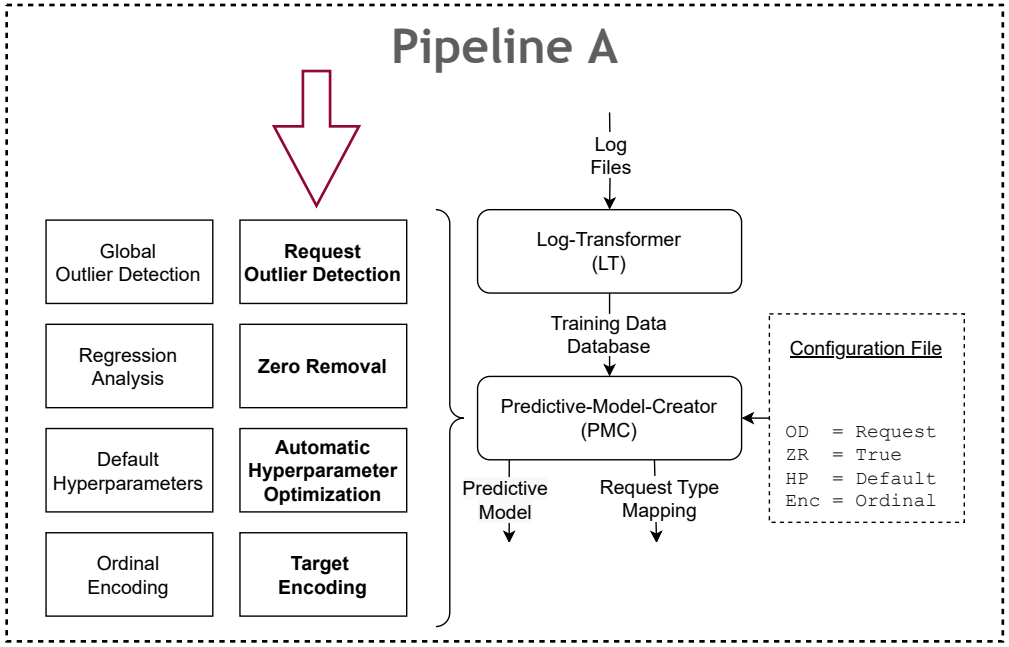
Pipeline A

- Compared to the initial version, the Predictive-Model-Creator component in Pipeline A was enhanced:
  - Request Outlier Detection,

~~Request Outlier Detection~~

Requests of Type T
x with 10 ms
y with 12 ms

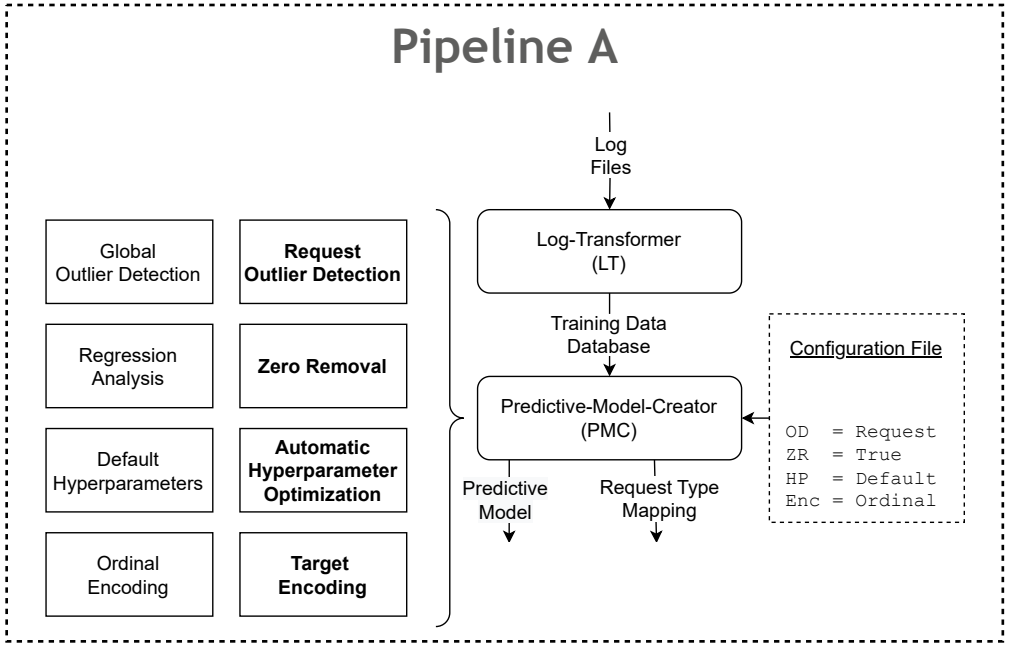
Requests of Type M
~~a with 100 ms~~
~~b with 1000 ms~~



RAST Components

Pipeline A

- Compared to the initial version, the Predictive-Model-Creator component in Pipeline A was enhanced:
  - Request Outlier Detection,
  - Zero Removal,
  - Automatic Hyperparameter Optimization using Grid search.
  - Target Encoding.



RAST Components

Pipeline A

Request Type	Processing Time
A	15
B	4
C	8
A	9
C	3

Categorical / Nominal data

Request Type	Processing Time
1	15
2	4
3	8
1	9
3	3

Ordinal Encoding

A = 1
B = 2
C = 3

A < B < C is logically **incorrect**

Request Type	Processing Time
4.8	15
0.8	4
2.2	8
4.8	9
2.2	3

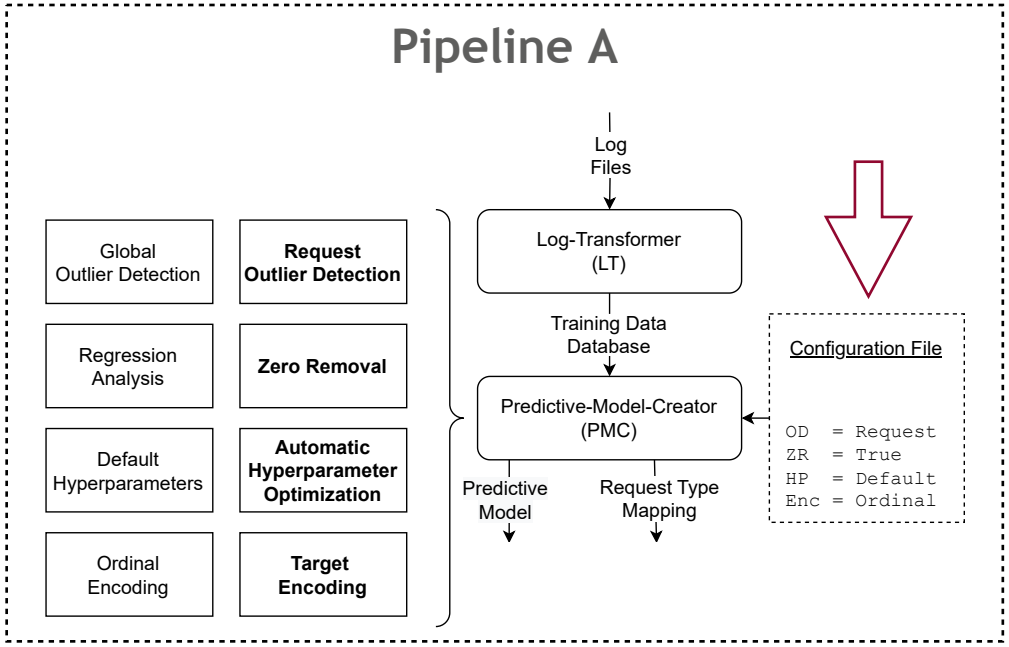
Target Encoding

Example Request Type A
(15+9)/2*0.4 = 4.8

RAST Components

Pipeline A

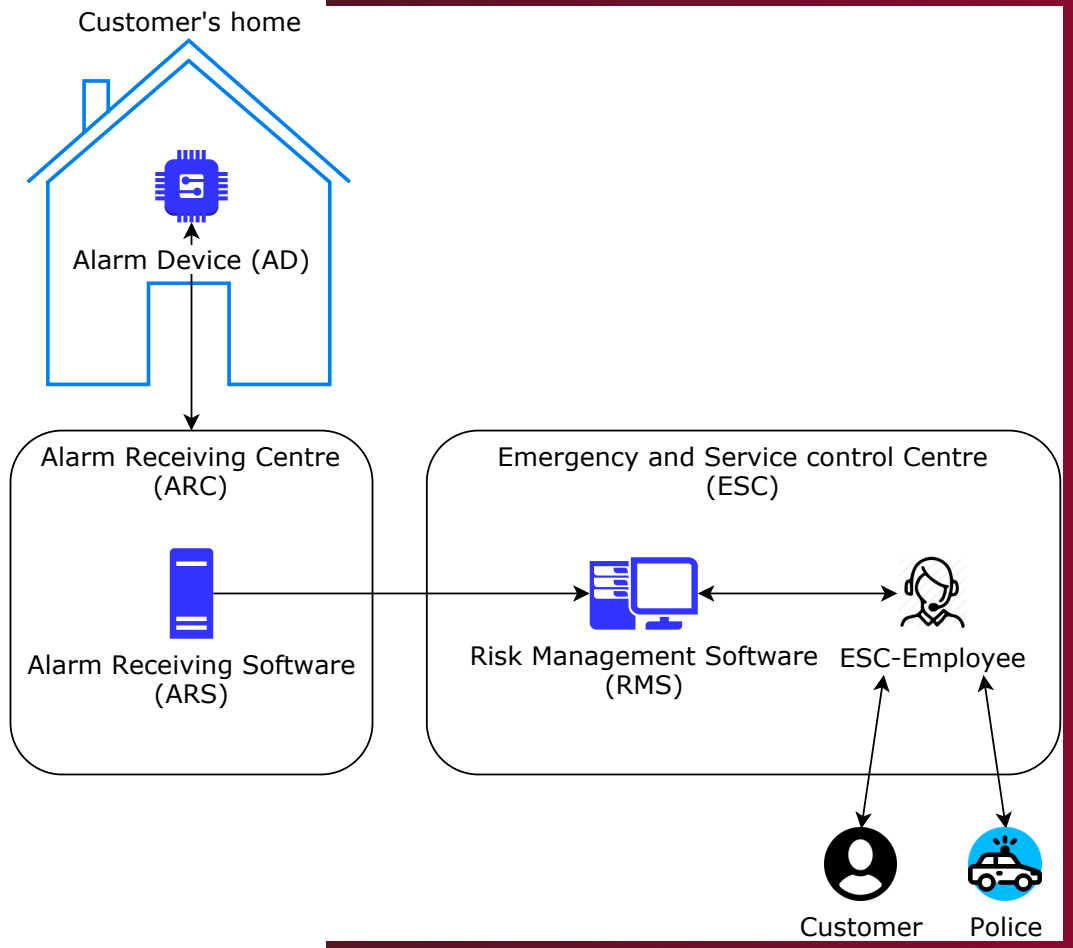
- Compared to the initial version, the Predictive-Model-Creator component in Pipeline A was enhanced:
  - Request Outlier Detection,
  - Zero Removal,
  - Automatic Hyperparameter Optimization using Grid search.
  - Target Encoding.
- Configuration File



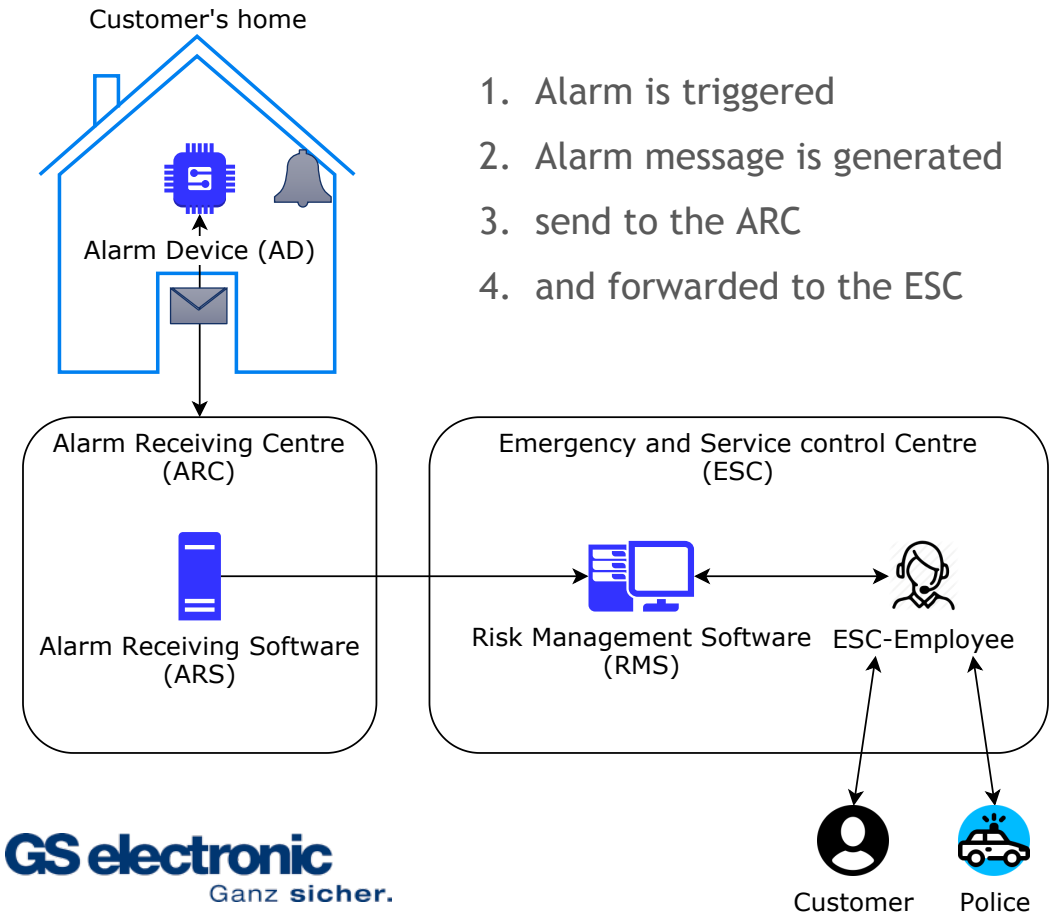
➡Different configurations will be presented in the upcoming slides

Case Study

Alarm System



Alarm System of GS electronic



1. Alarm is triggered
2. Alarm message is generated
3. send to the ARC
4. and forwarded to the ESC

Details of the system are given in our paper.

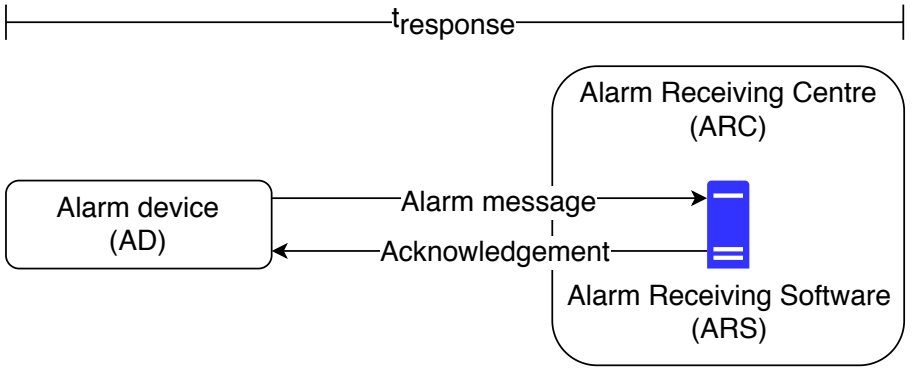
Real-Time Requirements of an Alarm Receiving Software

[9] EN 50136-1:2012-08 2012

- System Under Evaluation = Alarm Receiving Software
- Real-time requirements for the response time $t_{response}$ of an alarm receiving software are defined in the European standard EN 50136 [9]:

$$mean(t_{response}) < 10\ sec$$

$$max(t_{response}) < 30\ sec$$



Our Goal

- find the current system's saturation point, i.e., how many Alarm Devices (ADs) the system can simultaneously handle at maximum.
- company's question: can the system handle 25.000 ADs?

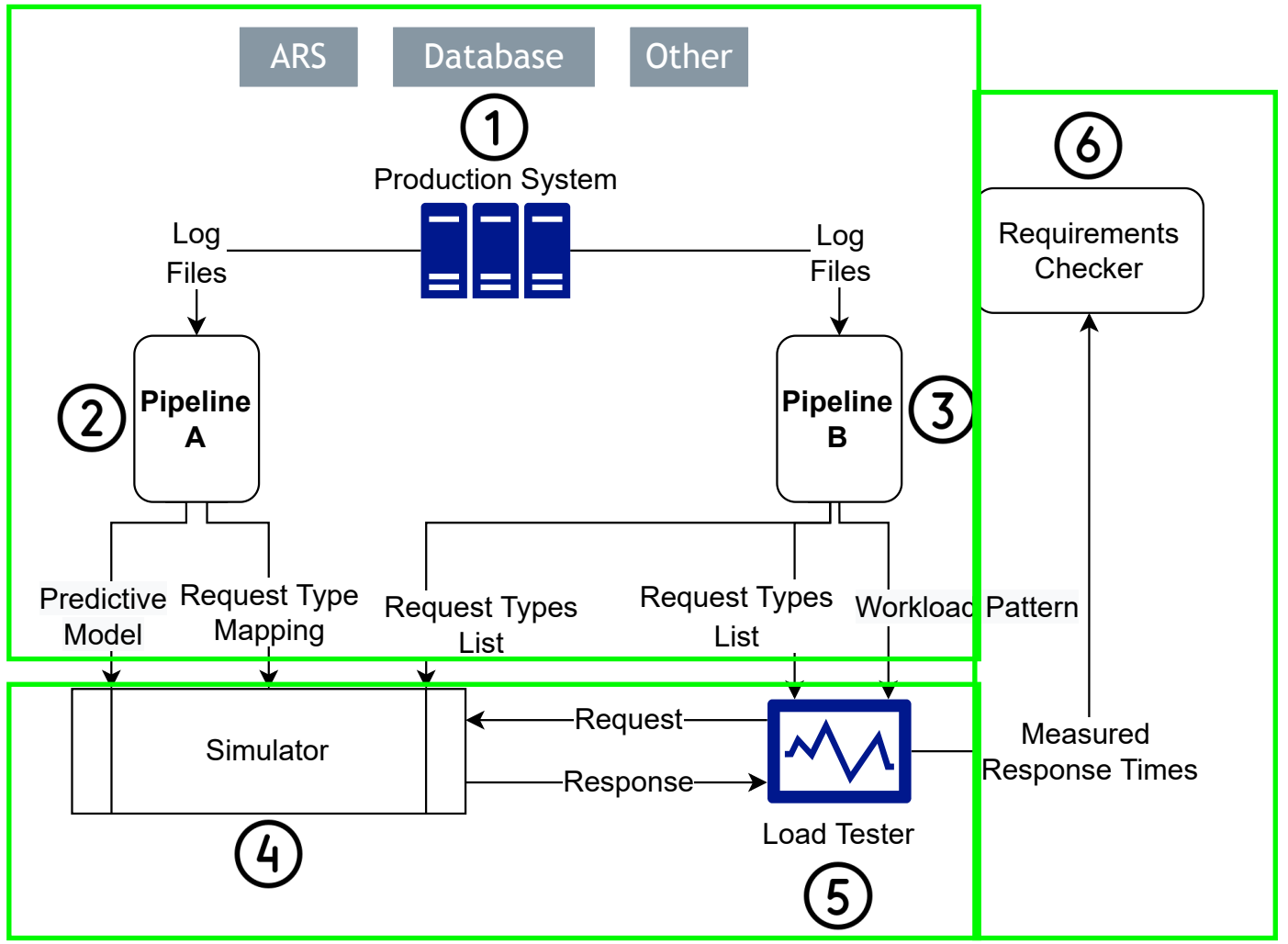
Workflow using RAST

Preparation Phase

Experimental Phase

① Collect log files and feed them into the pipelines ②, ③ to create the predictive model, the request type mapping, request type list, and the workload pattern.

Use the data created by the pipelines to run the Simulator ④ and repeatedly execute the Load Tester ⑤ with a continuously increasing workload until the Requirements Checker ⑥ reports a violation of real-time requirements.



Training the Optimal Predictive Model

- Giving RAST 180 log files of our particular Alarm System from 30 distinct days collected over 13 months yields a Training Database with around 20 million records.
- We determine the influence on the R^2 score of each individual RAST configuration:
  - Request Outlier Detection (ROD) or Global Outlier Detection (GOD)
  - Zero Removal (ZR)
  - Ordinal Encoding (CMD O) or Target Encoding (CMD T) of Request Type
  - Default or Optimized hyperparameters
- Goal: Find the RAST configuration that yields the highest R^2 score.

Training Database Record

Timestamp	No. Parallel Requests at begin (PR 1 N)	No. Parallel Requests finished (PR 2 N)	Request Type (CMD)	Processing Time

Training the Optimal Predictive Model

Influence of Outlier Detection Method and Zero Removal (ZR)

- Zero Removal has no effect when used with Global Outlier Detection (GOD)
- Request Outlier Detection (ROD) causes an increase of around 0.2 for all estimators except Lasso Regression.
- Zero Removal provides a slight improvement in combination with ROD
  - Likely due to our specific dataset

Estimator	(1) R^2 Score, No ZR, GOD	(2) R^2 Score, ZR, GOD	(3) R^2 Score, No ZR, ROD	**(4) R^2 Score, ZR, ROD**
Ridge Regression	0.537	0.537	0.756	**0.761**
Lasso Regression	-0.000	-0.000	-0.000	**-0.000**
ElasticNet Regression	-0.000	-0.000	0.263	**0.264**
DecisionTree Regression	0.626	0.626	0.811	**0.814**

Training the Optimal Predictive Model

Ordinal Encoding vs. Target Encoding

- Target Encoding
  - Improves Ridge Regression by 0.031
  - Degrade DecisionTree Regression by 0.004
  - Does not influence Lasso and ElasticNet Regression

Ordinal Encoding

R^2 Score
0.761
-0.000
0.264
0.814

Target Encoding

Estimator	R^2 Score
Ridge Regression	0.792
Lasso Regression	-0.000
ElasticNet Regression	0.264
DecisionTree Regression	0.810

Training the Optimal Predictive Model

Default vs. Automatic Hyperparameter Optimization

- Significant improvement for ElasticNet and Lasso Regression with an improvement of 0.484 for ElasticNet and 0.676 for Lasso Regression for both encodings
- No influence for Ridge Regression
- The DecisionTree Regression slightly improved by 0.006 for ordinal encoding and 0.001 for target encoding.

Default Hyperparameters

CMD O	CMD T
0.761	0.792
-0.000	-0.000
0.264	0.264
0.814	0.810

Optimized Hyperparameters

Estimator	R^2 Score CMD O	R^2 Score CMD T
Ridge Regression	0.761	0.792
Lasso Regression	0.676	0.676
ElasticNet Regression	0.748	0.748
DecisionTree Regression	0.820	0.821

Training the Optimal Predictive Model

Summary

- Best predictive model before optimization: DecisionTree Regression with an R^2 score of 0.626. => *Baseline model*
- Best predictive model after optimization: DecisionTree Regression with an R^2 score of 0.821 (31% increase). => *Optimal model*
- Observations:
  - Request Outlier Detection provides the biggest increase for *Ridge and DecisionTree Regression* => *Use Request Outlier Detection*.
  - Ordinal Encoding has a detrimental effect on *Ridge Regression*, but not on the other estimators => *Use Target Encoding*.
  - Default hyperparameters of *Lasso and ElasticNet Regression* are effectively useless => *Use Automatic hyperparameter optimization*.

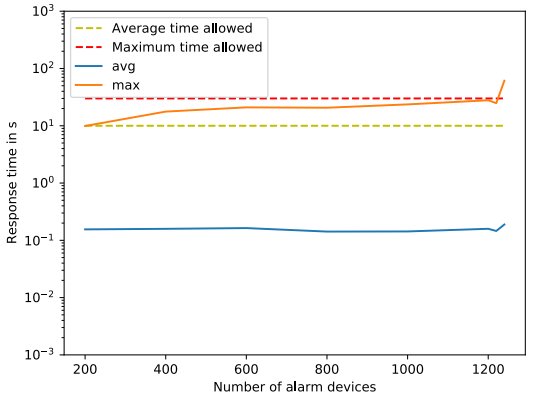
Experimental Results

[11] Keti 2015

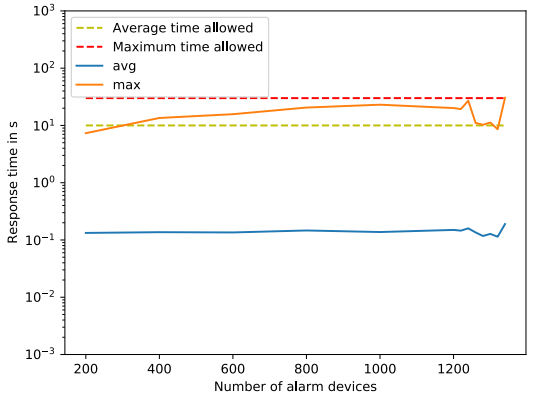
Experimental Setup

- Virtual Machine (VM) on an HP Proliant dl380 G7 server with two Intel Xeon X5690 processors with 3.46 GHz.
- Eight virtual CPU (vCPU) cores with 16 GB of RAM.
- Mininet [11] environment to simulate network latency.
- Within this environment, the Simulator, the Load Tester, and the Requirements Checker are automatically launched.
- For consistent comparison, we perform our experiment twice; with the baseline predictive model (PM1) and the optimal model (PM2).

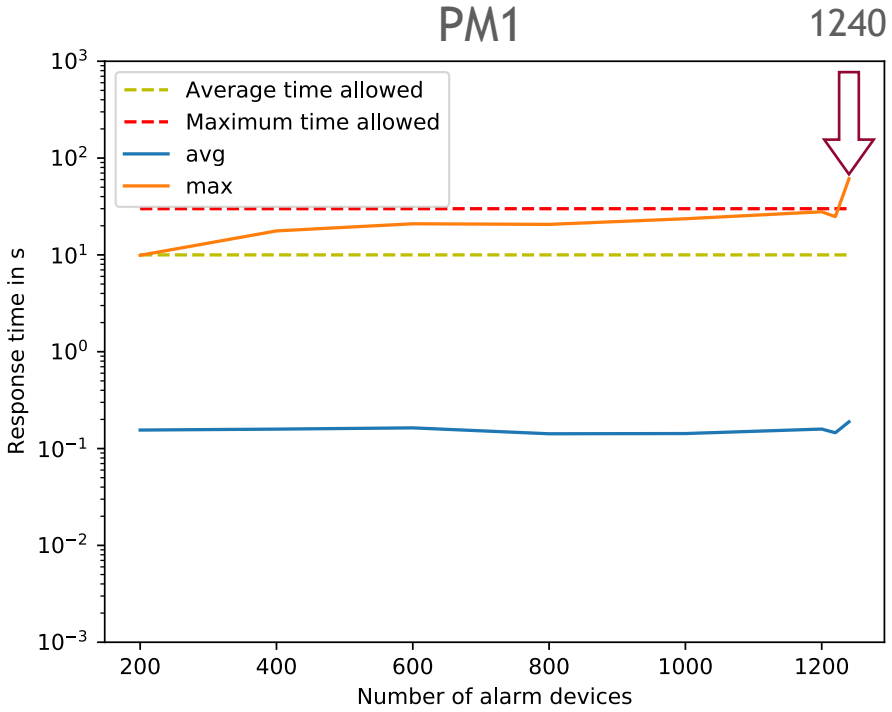
PM1

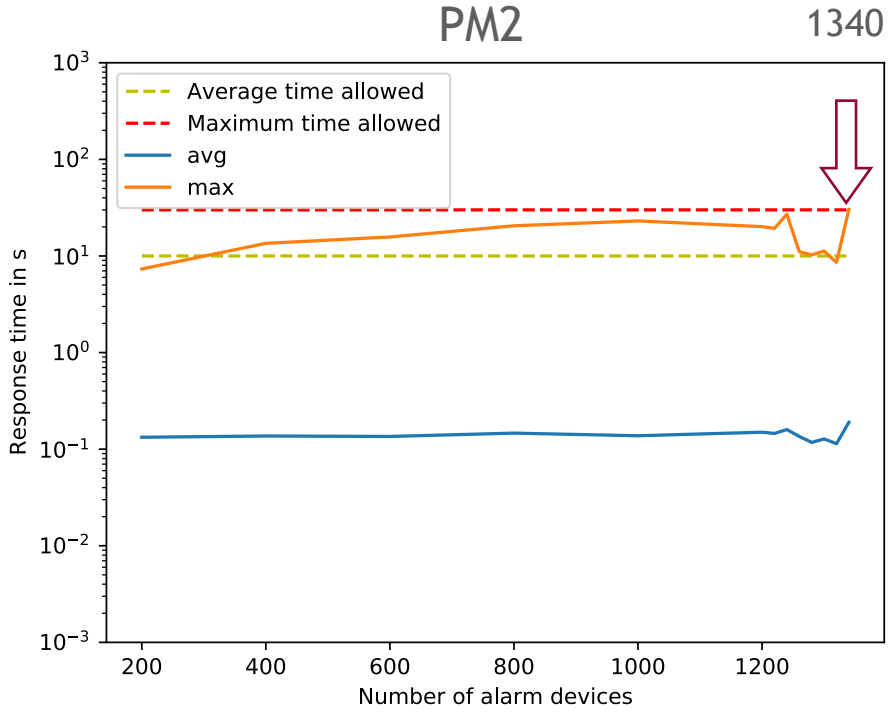


PM2

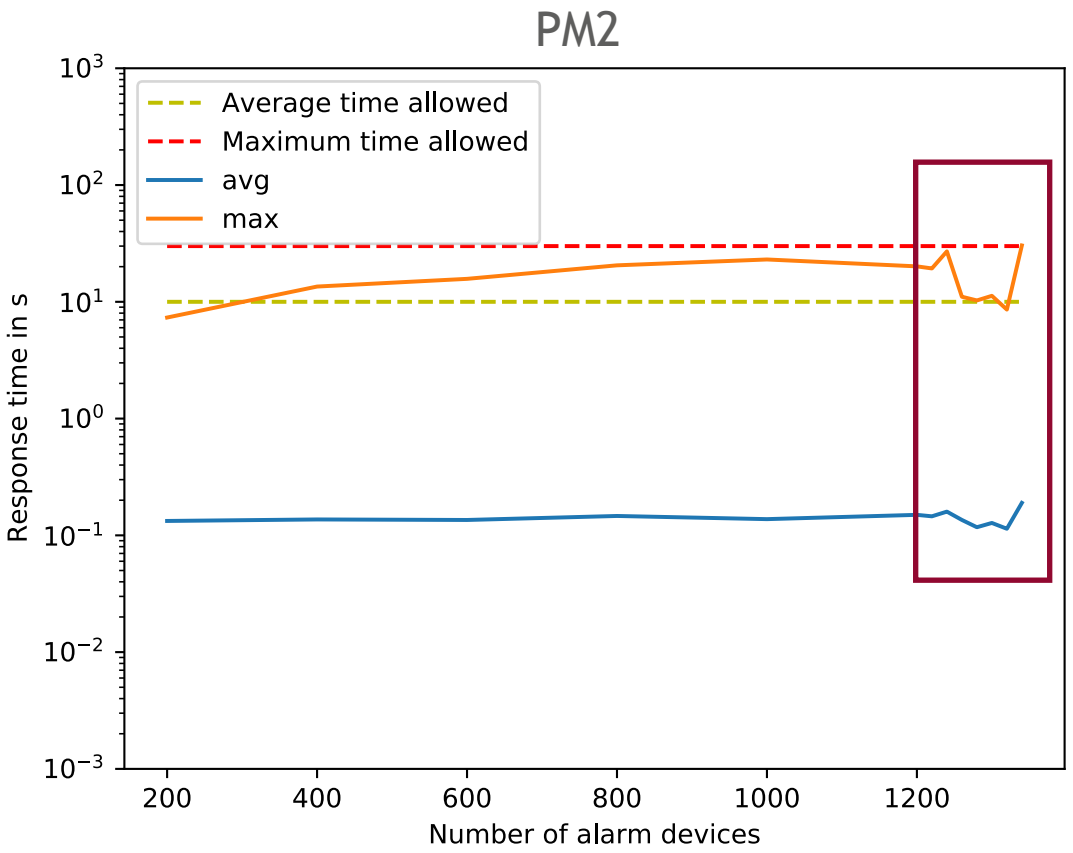


Experimental Results





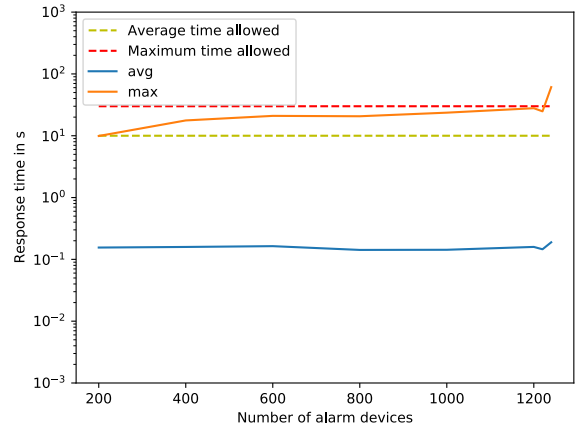
Experimental Results

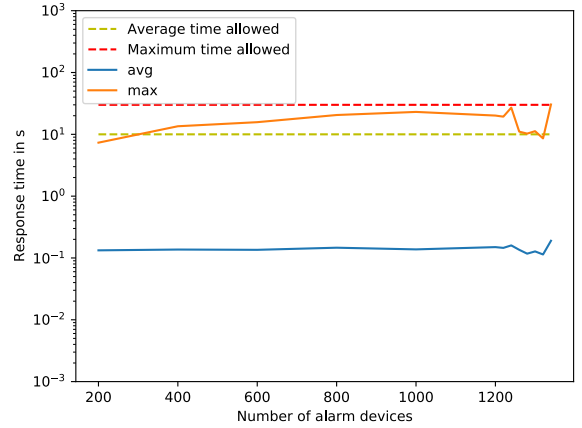


# AD	Avg. Response Time	Max. Response Time
1200	0.1388	21.434
1200	0.1084	10.725
1200	0.1824	27.885
1200	0.1697	20.422
1220	0.1112	6.675
1220	0.1557	29.053
1220	0.2064	33.650
1220	0.1089	7.878
1240	0.1550	30.108
1240	0.1418	16.087
1240	0.1822	34.908
1260	0.1355	11.054
1280	0.1174	10.271
1300	0.1277	11.252
1320	0.1139	8.580
1340	0.1897	30.379

Experimental Results

- Conclusion: The Alarm System can at least handle 1,340 Alarm Devices while complying with the real-time requirements of:
  - average response time below 10 sec,
  - maximum response time below 30 sec.
- Answer to the company's question: No, the Alarm System cannot handle 25,000 Alarm Devices.





Conclusion: Advantages of our Approach

- RAST is an toolset for evaluating the response time of distributed legacy software that runs in production.
- Compared with related work it offers the following advantages:
  - does not require APM tools,
  - does not change the existing system or disrupt its regular operation.
- With the improvements introduced in this paper, RAST provides common methods to fine-tune the predictive model training process.
  - It allowed us to to increase the R^2 score of our previously best model by 31%.
- We successfully used our RAST toolset in an industrial case study.

Conclusion: Limitations of our Implementation

- In our case study, it was not possible to consider the whole system in our model because background programs do not produce log files.
- The current model performs predictions based on only three predictor variables.

Conclusion: Future Work

- Improve the predictive model by extracting additional features from the existing log files, e.g., the specific request types that are being processed in parallel.
- Implement RAST for different (open-source) software systems, e.g., TeaStore.
- https://github.com/jtpgames/Locust_Scripts

Thank you for your attention, questions?

Juri Tomak

jtomak@uni-muenster.de

University of Muenster & GS electronic GmbH, Germany