Dynamic system analysis using telemetry

Oleh V. Talaver[1], Tetiana A. Vakaliuk[1,2,3,4]

[1] *Zhytomyr Polytechnic State University, 103 Chudnivsyka Str., Zhytomyr, 10005, Ukraine*
[2] *Institute for Digitalisation of Education of the NAES of Ukraine, 9 M. Berlynskoho Str., Kyiv, 04060, Ukraine*
[3] *Kryvyi Rih State Pedagogical University, 54 Gagarin Ave., Kryvyi Rih, 50086, Ukraine*
[4] *Academy of Cognitive and Natural Sciences, 54 Gagarin Ave., Kryvyi Rih, 50086, Ukraine*

Abstract

In the modern world of software development, the topic of distributed solutions implementation has become quite common, due to the flexibility it brings to big companies. The downside is that when developing such systems, especially when it comes to many teams, global design problems may not be obvious and lead to a slowdown in the development process or even problems with the location of errors or degradation of overall system performance. In addition, the timely reaction to system degradation is complicated by the distributed nature of the architecture, while manually configuring rules for reporting problematic situations can be time-consuming and still incomplete, automatic detection of possible system anomalies will give engineers (especially Software Reliability Engineers) the focus on problems. For this reason, applications that can dynamically analyze the system for the problems have great potential. Currently, the topic of using telemetry for system analysis is actively studied and gaining traction, so further research is valuable. The aim of the work is to theoretically and practically prove the possibility of using telemetry for the analysis of a distributed information system, detection of harmful architectural practices and anomalous events. To do this, firstly, a detailed overview of the problems related to the topic and the feasibility of using telemetry is provided, the next section briefly describes the history of the monitoring systems development and the key points of the latest OpenTelemetry standard. The main part includes an explanation of the approach used to collect and process telemetry, a reasoning behind the usage of Neo4j as a data storage solution, a practical overview of graph theory algorithms that help in the analysis of the collected data, and a description outlining how the PCA algorithm is employed to detect unusual situations in the whole system instead of individual metrics. The results provide an example of using the software presented in combination with Neo4j Bloom to visualize and analyze the data collected over a period of several hours from the OpenTelemetry Demo test system. The last section contains additional remarks on the results of the study.

Keywords

distributed systems, microservices, dynamic analysis, architectural smells, anti-pattern, visualization, telemetry, anomalies, Open Telemetry, graph theory, statistical analysis

1. Introduction

In recent years, distributed architectures such as microservices have received a lot of attention and popularity due to the opportunities that the architectural pattern opens up in terms

CS&SE@SW 2023: 6th Workshop for Young Scientists in Computer Science & Software Engineering, February 2, 2024, Kryvyi Rih, Ukraine
olegtalaver@gmail.com (O. V. Talaver); tetianavakaliuk@gmail.com (T. A. Vakaliuk)
http://acnsci.org/vakaliuk/ (T. A. Vakaliuk)
0000-0002-6752-2175 (O. V. Talaver); 0000-0001-6825-4697 (T. A. Vakaliuk)

of optimization, technology stack diversification, and more [1]. Distributed systems, when built correctly, simplify the development process when many teams are involved, reduce the complexity of changes or the dependence of teams on each other, and speed up development. The additional complexity that microservices bring [2] slows down the development of such a system, while the general advantage of technological heterogeneity only increases the effort required to maintain the codebase [3, 4, 5]. Therefore, there is a requirement for more tools to analyze the system and respond to problems. The development of distributed information systems requires more effort, especially when it comes to monitoring the entire system, finding problematic areas [2], because, unlike a monolith, such a system has many components developed in parallel, which may have structural flaws [6, 7], also referred to as architectural smells – design decisions that hinder maintainability and extensibility. For global problems location, an analysis of the information system is often conducted to find and quickly address such shortcomings [8]. A couple approaches exist, such as static analysis of the codebase of each of the system components or analysis of the system logs. Both options are quite complex, because they require adjustment for each individual system, technology, programming language, but the static approach, unlike the dynamic one, allows you to analyze the system without the need to run the whole system, which allows you to correct some local code smells, but not the problems of the system as a whole, due to the low accuracy and insufficient information about the runtime behavior [9]. At the same time, dynamic analysis is based on the information gathered in runtime, therefore presents more accurate representation of the system utilization. The most prominent approach of dynamic analysis is through usage of telemetry that combines three pillars of system observability: logs, metrics, and traces. Therefore, the purpose of the work is the theoretical and practical substantiation of the possibility of using OpenTelemetry standard for the purpose of analyzing a distributed information system: detecting and quickly responding to harmful architectural practices and anomalous events. Next, we outline the tasks:

- research of state-of-art approaches of telemetry analysis;
- modeling extract, transform and load (ETL) and further telemetry analysis process;
- analysis of the received data to identify harmful practices and anomalies.

2. Theoretical background

The topic of system observability is far from new. In the world of distributed systems, Google is considered a pioneer in the study of the topic of observability. In 2010 Google engineers published paper called "Dapper – a Large-Scale Distributed Systems Tracing Infrastructure" [10] which prompts the emergence of the first systems for request traces visualization: Jaeger and Zipkin. However, these applications solved the same problem while being incompatible causing vendor lock, so over time, the development of the OpenTracing [11] standard begun. The new standard provides a layer between the application and monitoring systems to track and collect requests. This standard didn't solve the whole problem, so OpenCensus standard was later developed to focus on system metrics and logs collection, but also included an alternative implementation of traces collection, which ultimately created more problems, as developers now had to choose between the two standards. For this reason, in 2019, both standards were combined into OpenTelemetry [12] to solve the following tasks:

- gathering traces, metrics, and logs in one place;
- finding anomalies through charts;
- finding the location and cause of anomalies through the review of problematic request traces.

Also, at that time, quite a lot of applications helping in a system monitoring had already been presented on the market, for this reason, one of the tasks was to maintain compatibility with them, so the latest standard provides a specification that describes approaches for metrics collection, conventional descriptions of processes such as interaction using the HTTP protocol, RPC [13] without forcing vendor lock. As a result, OpenTelemetry is currently the most active project of the Cloud Native Computing Foundation [14].

One of the main notions introduced in the standards is telemetry – a set of metrics, logs and, most importantly, traces [15], which in the case of our topic can be used to build a system model in the form of a directed graph [9] and later used to analyze and identify bad practices and problem areas of the system. The OpenTelemetry standard is relatively new, so there is still active research of the possible use cases, but the main area of use is the visualization of requests (figure 1) with the ability to search for problematic areas, for example, the cause of poor service performance or the root cause of an incorrectly working business process [15].

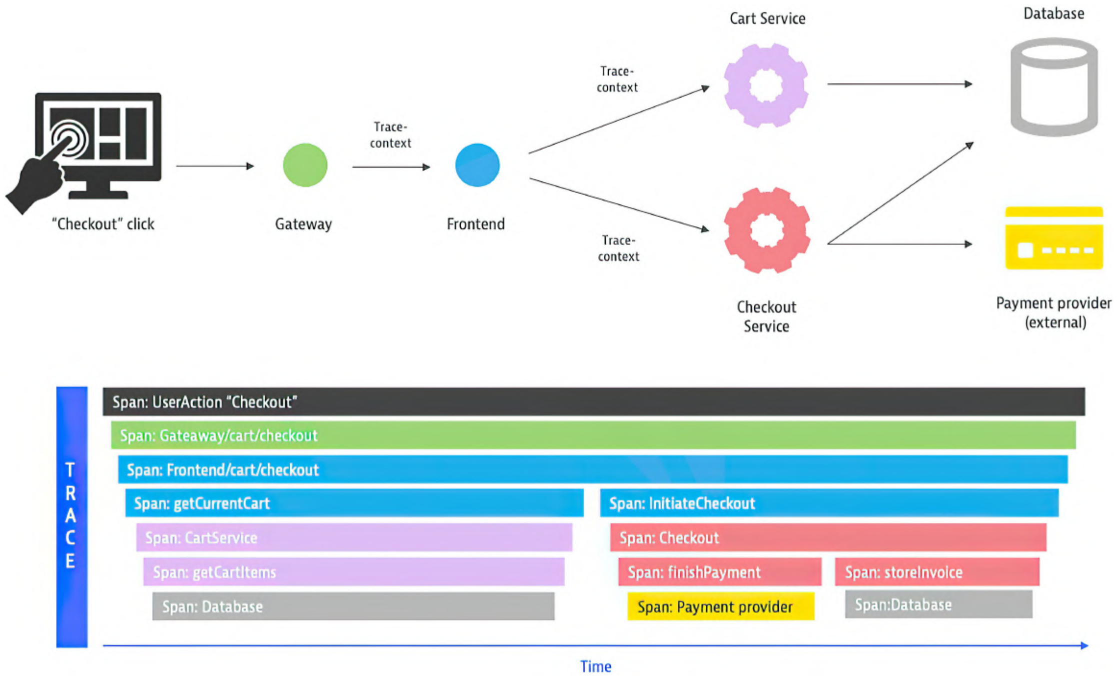

Figure 1: Request trace visualization.

The idea of using telemetry to improve the structure of a system can be traced back to several research papers released in recent years [16, 17, 18], and has a fairly small list of problems that can be identified, which opens up opportunities for further study of this topic [8].

3. Methods

3.1. Defining data and storage for architectural smells detection

The proposed analytical system receives a constant stream of telemetry data, aggregates it by updating the system model in the form of a directed graph stored in a graph database management system (DBMS). After that, the model can be used for analysis, searching for structural anti-patterns.

Constructing the system's graph model involves processing traces of requests (figure 2). Once they are received, the process creates or updates information about available resources (services, storages, proxies) and stores information about changes in the storage. operations available in the service (operation) and individual sub-requests (hop).

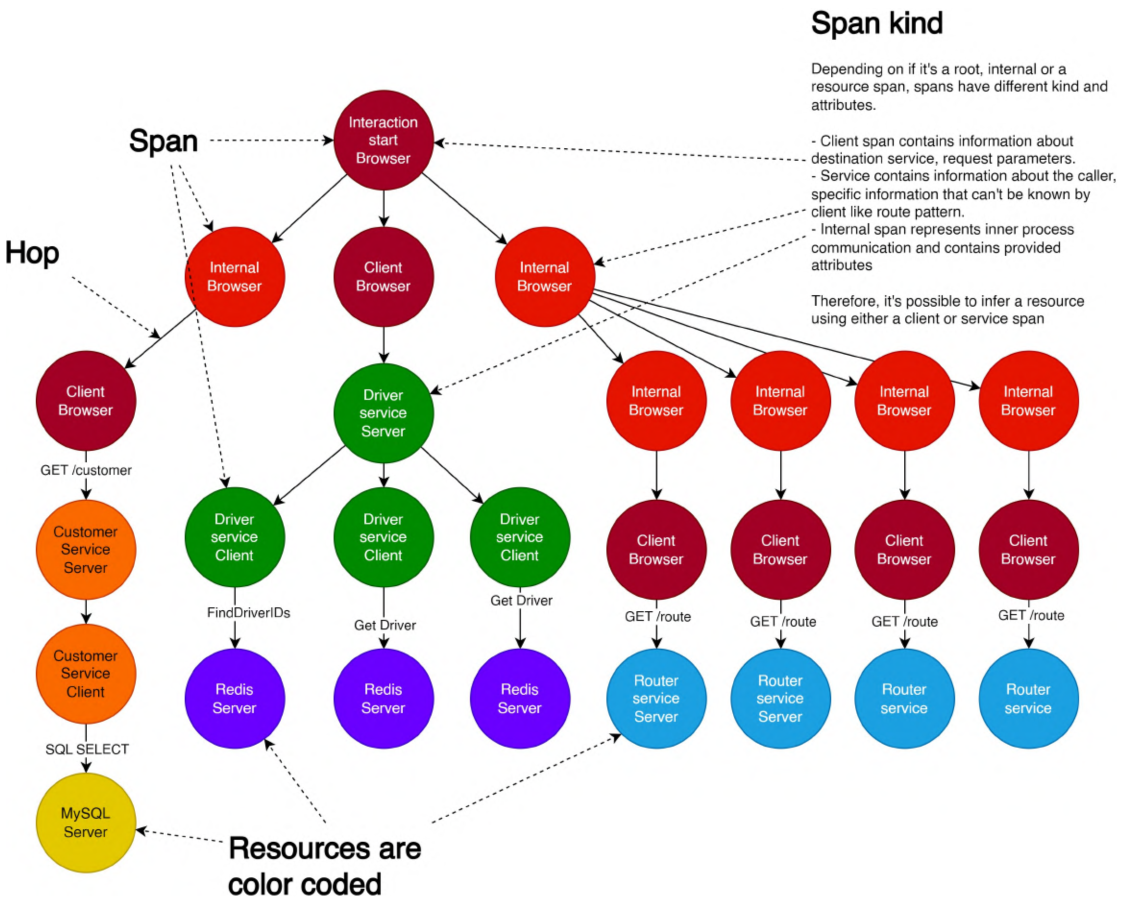

Figure 2: Example of a request tree.

To build a system graph for further analysis, the data storage must have the following information:

- *resources* are interacting components of the system. A resource must have a name, type (service, storage), date of creation and last use;

- *operations* are defined by one resource and called by other ones; have statistics on the number of calls, errors, the last date of creation, use;
- *calls* – connections between a resource and an operation. They has the date of creation, last use, type (synchronous, asynchronous), number of errors.

Neo4j was chosen as the storage of the system model since it physically stores data as a graph, which makes it possible to use graph traversal algorithms to find bad practices in the system, namely:

- clustering coefficient – measures the degree of vertex connectivity; will help show service groups in the system [19];
- degree centrality – measure the number of connections between vertices; makes it possible to calculate the affinity (coupling) metrics of components in the system [20];
- strongly connected components – finds groups where each vertex is accessible from any other; helps to identify cyclic dependencies in the system [21].

The graph DBMS structure is presented in figure 3.

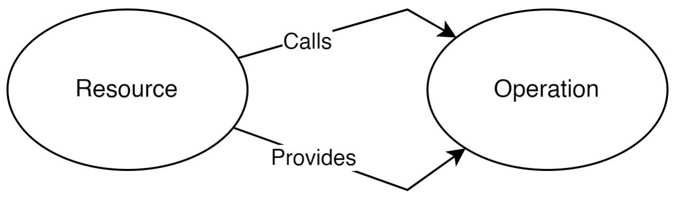

Figure 3: Simplified diagram of the structure of a graph DBMS.

Data storage has two types of nodes: resources and operations. Resources are related to the operations with the “Provides” relation. To show calls, the “Calls” relationship is used, which aggregates statistics for all identical calls from one resource to an operation of another resource.

The ETL process begins with instrumentation of the system – installation of modules for popular libraries that will be collecting the telemetry, manual changes in service code to provide more details of a particular process in the system. Later the telemetry is sent to the OpenTelemetry Collector [22] – a separate modular application developed by the authors of the standard, which allows you to unify the process of collecting, transforming, and exporting telemetry into various popular monitoring systems. There are present numerous available modules, but for this task, it’s required to create a custom exporter, which takes batches of traces, extracts necessary data and unloads it into neo4j.

3.2. Methods of anomalies detection

The problem of finding and analyzing anomalies is quite common in computer science and often varies depending on the domain in which the analysis takes place. For example, when reading data from sensors for further analysis, it is important to find and correct outliers. When analyzing a business process, it is sometimes necessary to find unusual events to analyze what

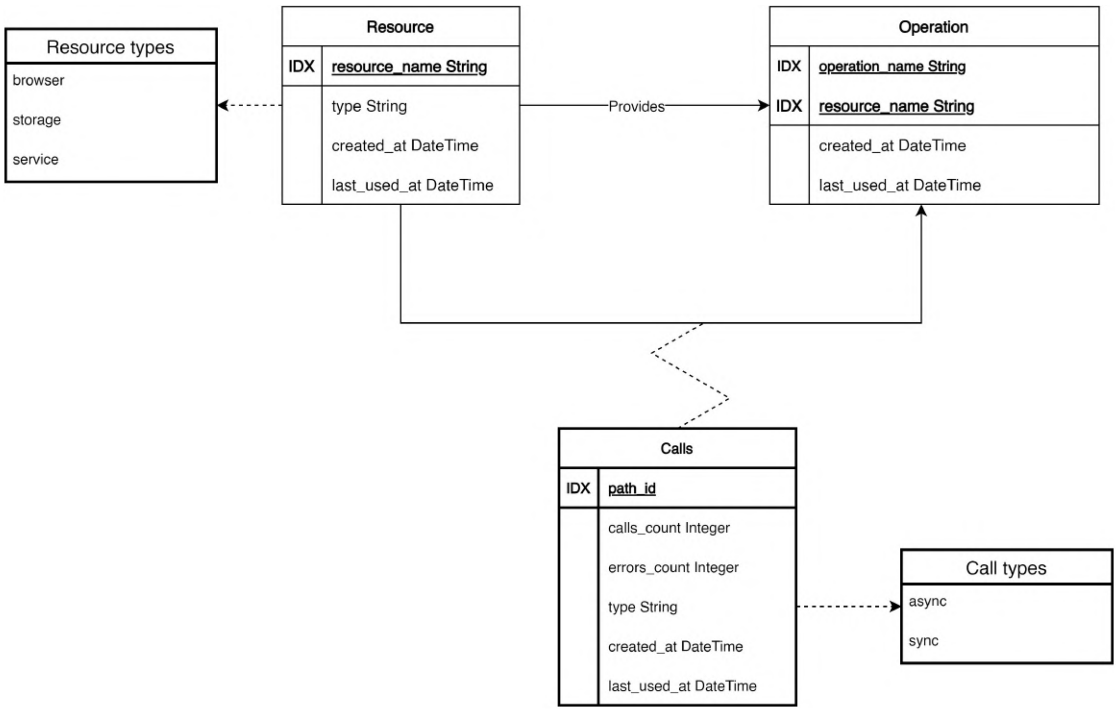

Figure 4: Complete graph DBMS structure diagram.

led to them. In the domain of software reliability engineering, the topic of mean time to detect is one of the most important indicators, because if the problem is found earlier, it is fixed earlier.

Analysis of anomalous changes is already present at least in New Relic, however, it is present at the level of individual services, not the entire system. Although there is not enough data to confirm this, the platform analyzes metrics, including key metrics, using the Exponential Smoothing [23], which is a method of predicting a single variable and, depending on the type, can take into account seasonality [24]. However, it is also possible to find anomalies the opposite way – from a larger scale, using multivariate algorithms, which will be used in this work.

An anomaly is an abnormal situation, essentially defined as a strong difference between expected and actual measurements. Therefore, the process of finding an anomaly includes the process of predicting the value of a certain measurement based on historical data [25].

The problem of finding anomalies in multivariate datasets is quite popular and critical because little to no measurements are univariate [26].

Algorithms are divided into the following training approaches:

- unsupervised – the dataset used for model training does not include labels indicating anomalous situations;
- semi-supervised – the dataset has anomalous situations labeled;
- supervised – the whole dataset is labeled, the least commonly used type of algorithm as it’s difficult to get fully labeled data.

Due to the difficulty of obtaining labeled data, unsupervised models are the most popular, while it's also possible to add the possibility of providing feedback and correction loop when using models for semi-labeled datasets. As part of this work, unsupervised model is reviewed. While "None of the unsupervised methods is statistically better than the others" [25], which is due to the complexity of training on unlabeled data in which extra parameters only interfere, it was decided to choose Principal Component Analysis (PCA) – a statistical method of multivariate analysis used to identify the main structural components in a dataset. The main goal of PCA is to reduce the dimensionality of data while explaining dataset in as much detail as possible, which, due to the simplicity of the approach, is well suited for multivariate datasets and makes it the most common algorithm.

Essentially, PCA converts the initial correlated variables into new linear combinations called principal components. The first principal component is defined in such a way that it explains the largest part of the variance of the data. Each successive principal component is chosen in such a way that it is orthogonal to the previous ones and explains the residual variance as possible.

In the case of identifying anomalies in the system, we are interested in the following information:

- calls – number of incoming, outgoing, and internal calls (synchronous and asynchronous when using a queue or other message brokers) with and without errors;
- duration – time spent processing requests.

To obtain the necessary data in the form of metrics, as well as to group collected data, you need to use a special connector component that transforms traces into call and duration metrics. Thus, the collector receives information about the request via the Open Telemetry Protocol (OTLP) then groups, extracts the necessary metrics to later export it.

Metrics for a certain period are collected for each of the components (resources) of the system. Metrics have different types of values, for example, the number of calls has the sum type that is a counter of certain events for a period and in this case is a monotonous sequence, because the number of calls never decreases.

It is also important to note that the metrics are returned as a delta (the value of aggregationTemporality is 1) and not a cumulative value, because we are interested in the number of calls in a certain period, not the absolute value. Each metric can have multiple points that represent different attribute-defined dimensions (dimensions are customizable), so separate counters have been set up for different request types (span.kind) and statuses (status.code).

Calls duration metric snippet:

```
{
  "name": "duration",
  "unit": "ms",
  "histogram": {
    "dataPoints": [
    {
      "attributes": [
      {
        "key": "service.name",
        "value": {
          "stringValue": "quoteservice"
        }
      },
      ...
      ],
      "startTimeUnixNano": "1685509043760171242",
      "timeUnixNano": "1685509058790174364",
      "count": "1",
      "sum": 0.006665,
      "bucketCounts": [
      "1",
      "0",
      ...
      ],
      "explicitBounds": [
      0.1,
      1,
      ...
      ],
      "exemplars": [
      {
        "timeUnixNano": "1685509058790174364",
        "asDouble": 0.006665,
        "spanId": "ade03fcb73f18048",
        "traceId": "7f6cf387237813d1f3891b5f21b09be2"
      }
      ]
    },
    ],
    "aggregationTemporality": 1
  }
}
```

If we take the call duration metric, then in this case we have a histogram, which is a certain aggregation of values and their distribution over intervals used for easier visualization.

But in this form, we will not be able to use this data. Firstly, all the metrics for individual services are separated and converted to time series to later be combined based on timestamp.

From the intermediate results, you can clearly see the correlation between the different metrics of the system components (figure 5), which is confirmed by a correlation map (figure 6).

The process of identifying anomalies occurs by splitting the data sample into two periods, the first is used to train the PCA statistical model, the second is used to compare with the predicted values obtained from the model and, estimate the error for all and specific metrics.

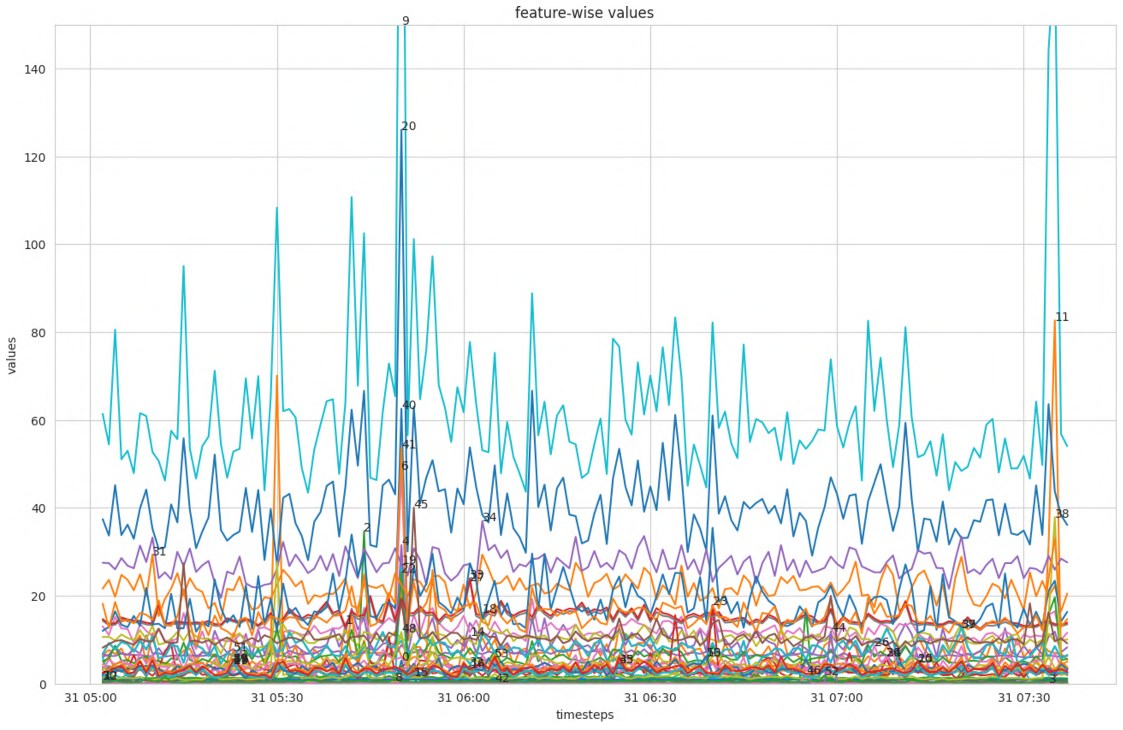

Figure 5: Chart of metric values over time.

4. Results

The OpenTelemetry Demo project was used as a test system [27], which is specially designed for testing applications working with telemetry. This is a distributed system that has components built with different technologies that is automatically loaded using load generator service.

4.1. Visualization of the service graph using Neo4j tools

After some time of running the whole system the graph database has the following data (figure 7). You can see that the graph has many nodes with the type of operation (orange circles), as well as slightly fewer services (purple circles). Between them you can see the “calls” and “provides” relationships depicted as arrows.

To simplify the graph, a function from the APOC library is used [28] for Neo4j in order to visualize the graph projection and show service dependencies (figure 8).

Snippet of a virtual relationship visualization query:

```
MATCH (r1:Resource)-[:Calls]->(:Operation)<-[:Provides]-(r2:Resource)
RETURN r1, r2, apoc.create.vRelationship(r1,'DependsOn',{}, r2) as rel
```

In figure 8 you can see the dependence of the checkout service on many others. To confirm this, let’s use an application called Neo4j Bloom to visualize Local Clustering Coefficient [19] and Degree Centrality [20] algorithms.

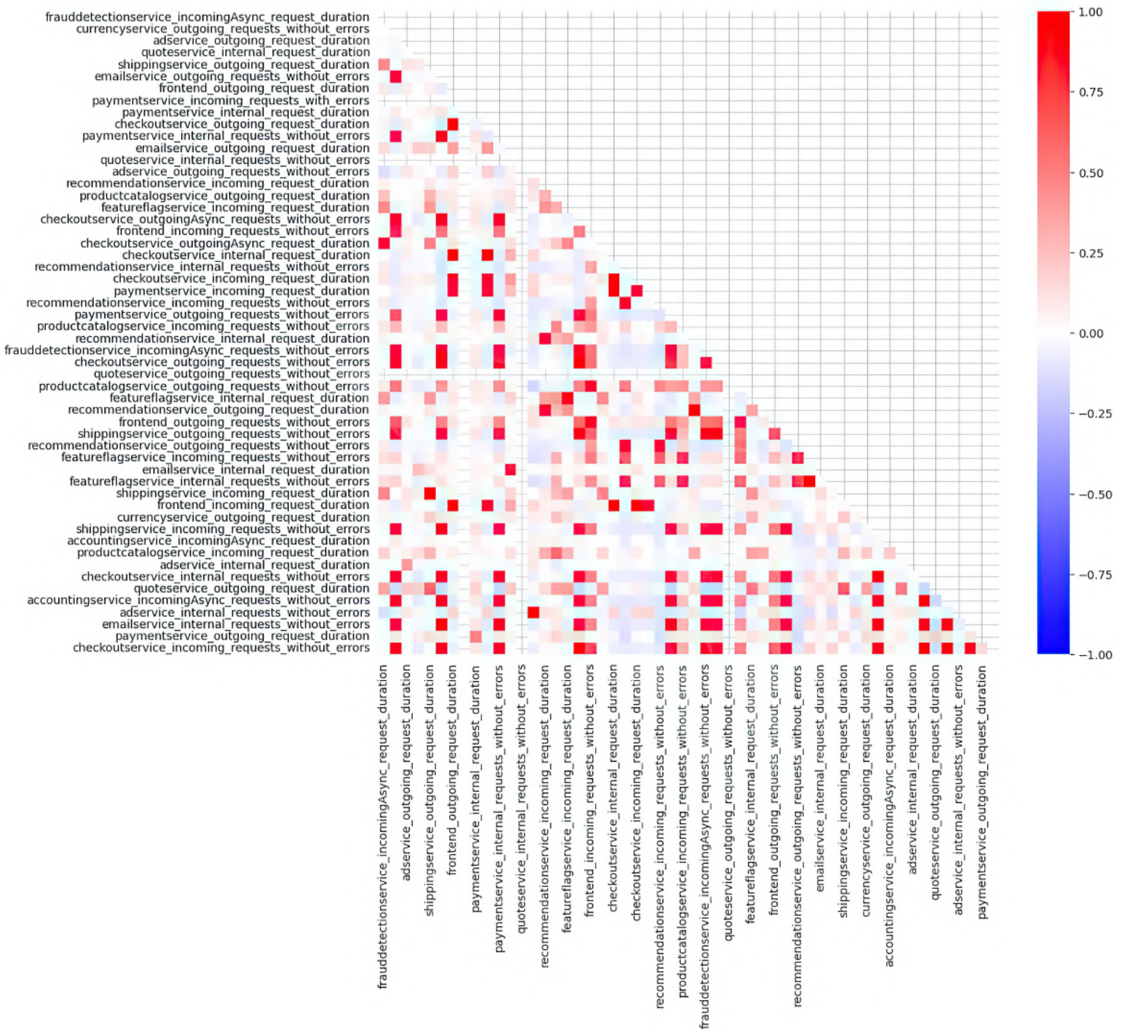

Figure 6: Metrics correlation map.

In the resulting diagram (figure 9) clusters are marked with distinct colors, the dependence of services on peers is indicated with their size. From the diagram it's also clear that the checkout service has many dependencies. This way, you can quickly analyze the architecture of the application, and clearly see parts that must be refactored to prevent a situation where the whole application halts due to a single bottleneck component.

4.2. Time interval anomalies analysis

To detect anomalies, a few hours long time interval was chosen. It has been processed using the PCA algorithm and after receiving errors for each of the time points, a visual analysis can be performed for the presence of spikes in the error values (figure 10).

As you can see in the plot, between 6:50 a.m. and 7 a.m., there were some changes that

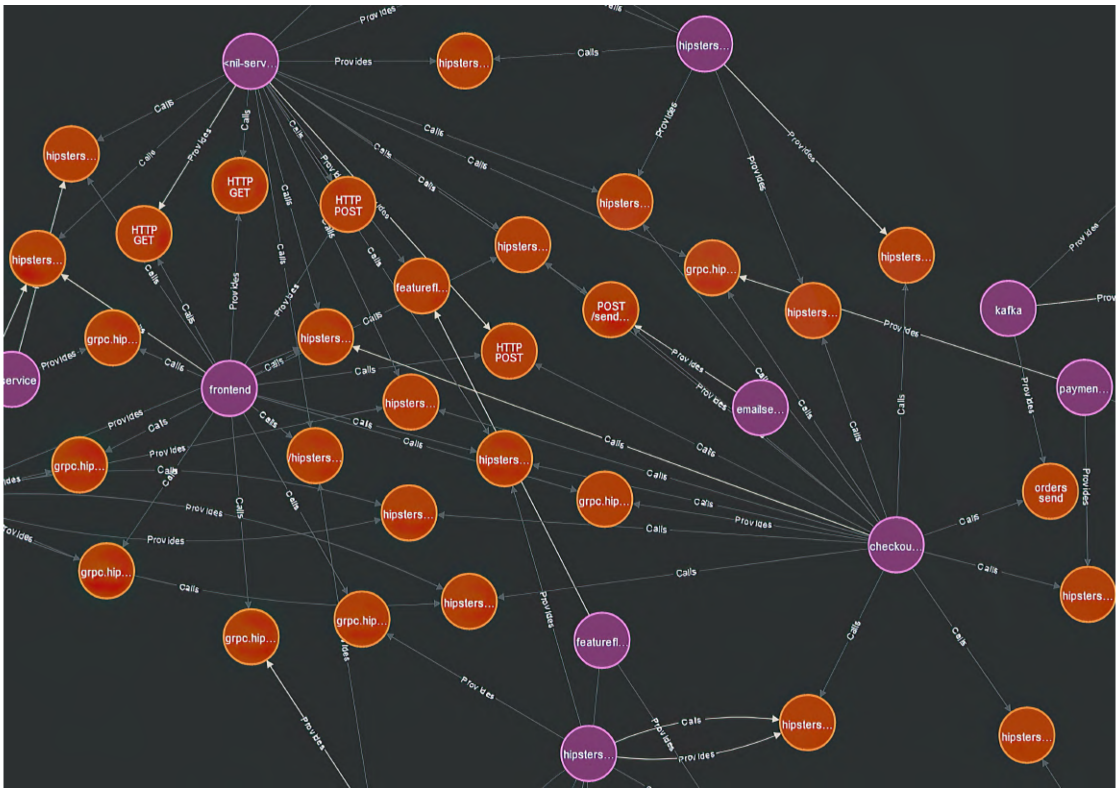

Figure 7: Visualization of the full graph of services, operations and connections between them using Neo4j Browser.

led to a relatively big error. From the error graph for each of the features, it can be seen that feature 44 (accountingservice_incomingAsync_request_duration) is involved in this error, so by conducting a more detailed analysis of the values of this metric, we can see that all values are kept near 0, while there is an outlier with a value of about 12 (figure 11).

5. Discussion

Compared to static analysis approaches, dynamic analysis allows you to see the real picture of the entire system, all possible query paths that are used, accurately indicate the components that cause a problem in the performance of the system at particular moment, in contrast to static analysis of individual modules, which is better suited for the tasks of identifying code smells. Telemetry, in turn, allows you to combine all key indicators and add the additional context that allows you to get more information for analysis.

The practical use of a simple statistical unsupervised PCA algorithm has demonstrated the possibility of using such a model to identify anomalies, which can greatly simplify the work of engineers, because instead of looking on dozens of charts and responding to user messages in support, this statistical analysis suggests the occurrence of anomalous situations in the system automatically. When compared with the approaches of analyzing each metric of the system

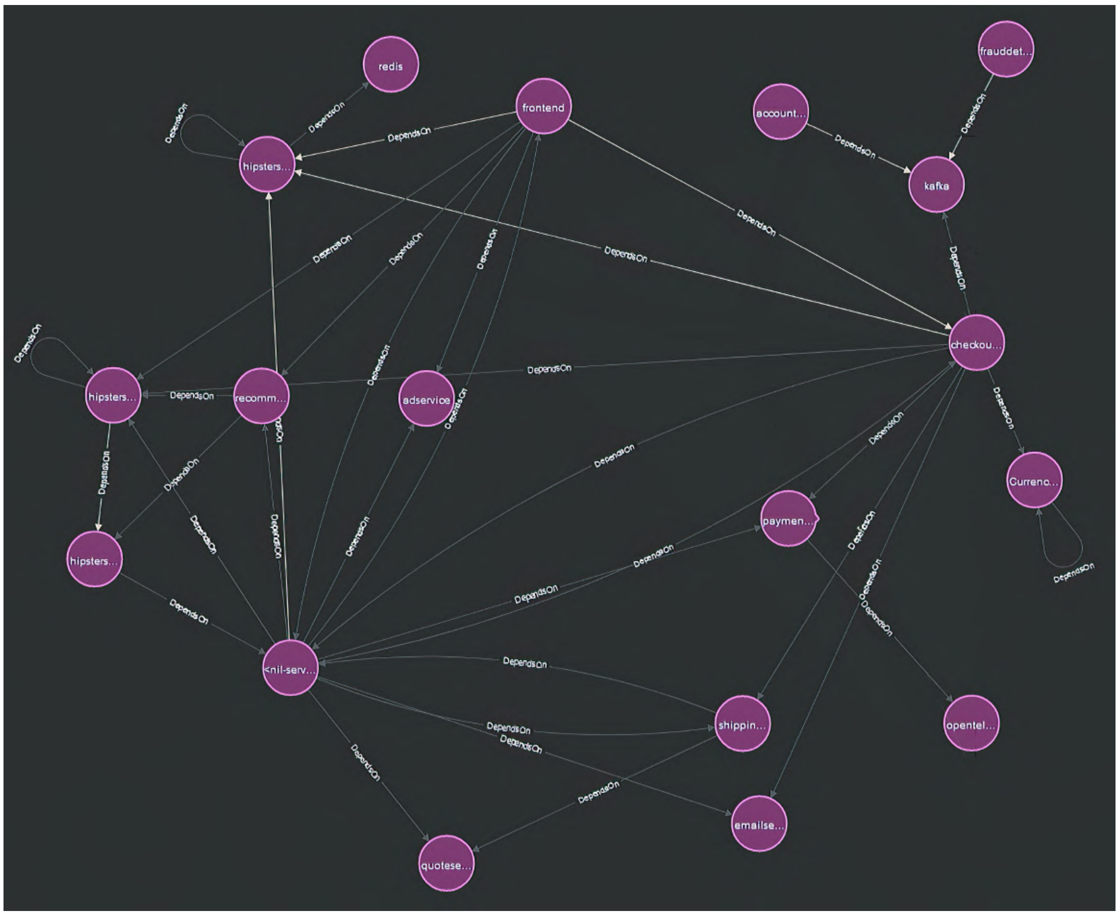

Figure 8: Visualization of the dependency graph in Neo4j Browser.

separately (using appropriate statistical methods, for example, those used in NewRelic [23]), this method gives the general picture, allowing you to understand the situation in the entire system, but also provides the cause of the problem. Compared to supervised algorithms, especially neural networks [29, 30], using the proposed method removes the need to retrain the model to adapt to normal changes (e.g., a natural increase in the number of users of the system), because the analysis takes place in a certain window, although undoubtedly this window should be of a particular size to cover a sufficient amount of data for training and analysis and at the same time not be too sensitive to seasonal changes (for example, activity during the day vs. activity at night), which needs to be tested and determined on a real system.

6. Conclusions

The paper discusses the use of telemetry for dynamic analysis of the system for anomalous events and architectural smells detection.

An analysis of the problems related to distributed systems development was carried out, which

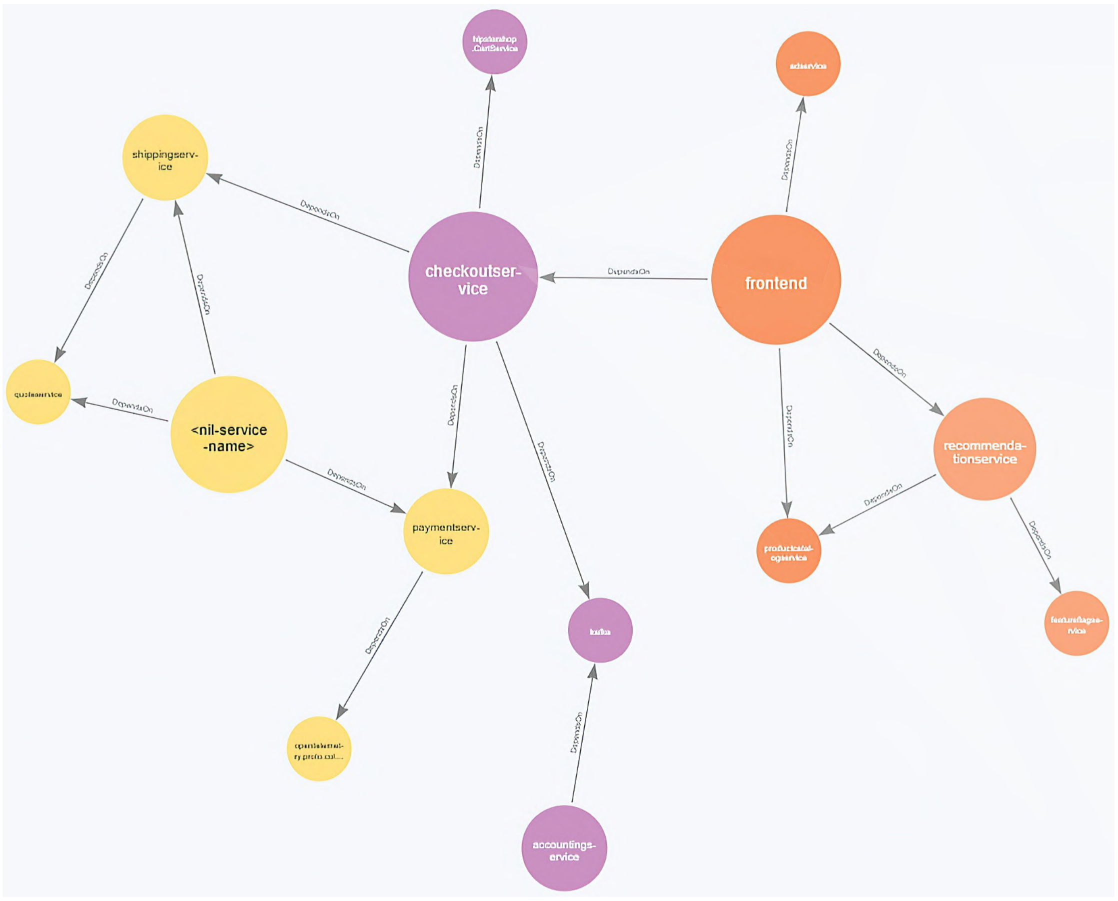

Figure 9: Visualization of the dependency graph of services considering clustering and centrality algorithms in Neo4j Bloom.

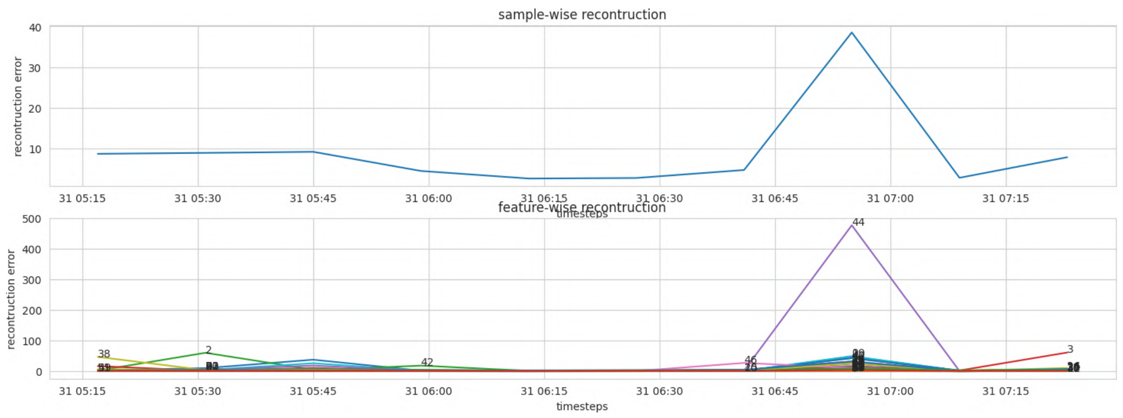

Figure 10: The result of displaying the data reconstruction error for all and individual metrics.

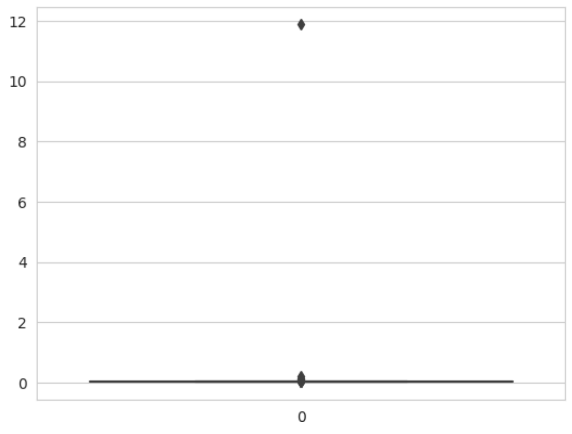

Figure 11: Box plot of values accountingservice_incomingAsync_request_duration.

made it possible to determine the need for applications for monitoring and rapid response to problems in a large system. Studies on the use of telemetry for dynamic system analysis, which have been published in recent years, have shown the potential of this approach. The history of the system monitoring topic development and the main aspects of the latest OpenTelemetry standard were reviewed.

Later, the main data flows that are required for analysis were identified, and a model of a graph DBMS was built. The model includes the following entities: operations, resources and relationships that determine the direction of resource dependence and ownership of operations. After that, the extraction, processing and unloading telemetry using the OpenTelemetry Collector was reviewed. The main types of anomaly detection algorithms were studied and the multivariate PCA statistical method was chosen for the analysis of unlabeled telemetry data. A custom component of the collector application was developed to transform and insert information into Neo4j datastore. The necessary features to be used are defined, as well as the process of collecting appropriate the number and the duration of calls within the system to find anomalies. An algorithm for collecting metrics was described. The next part overviewed the method of using a statistical model of PCA to identify anomalies.

In the next part, aggregated graph model was used to analyze architectural smells. Several possible visualizations of the dependency graph using Neo4j, Neo4j Bloom were provided, clustering and centrality algorithms were used to visually identify problem areas in the system architecture. The results of the statistical model based on the Principal Component Analysis algorithm were also analyzed. The accuracy of this model is sufficient to determine anomalous events.

The topic of telemetry usage to find bad architectural practices has potential for further development [18], after all, the collected data is enough to determine more complex patterns: too long synchronous and asynchronous requests, long chains of synchronous requests, the presence of too many different technologies in a small system, a large time difference between when an event is published and processed. To provide even more opportunities for analysis, it's possible to enrich system resources with additional metadata indicating belonging to a certain bounded context (to compare de jure and de facto clusters of contexts, to identify situations of using the same database in different parts of the application) [1], dates of any changes or releases. Also, this data should be displayed on the main map of the system components. Further development of anomaly analysis includes integration with an application performance monitoring (APM) system, the ability to configure threshold values for reconstruction error, adding custom metric streams for analysis and finally testing on a real system with a comparison with existing approaches.

If we consider other topics of telemetry usage, we cannot omit the topic of analyzing individual use cases in the application, which are sets of requests that go through a bunch of services and have variations depending on some stored state of the application, the analysis includes both the ability to evaluate performance, the number of errors of a particular use case, the ability to subscribe a certain development team to updates for a quick response in case of anomalous situations, and ability to view changes, including performance, between different releases.

References

[1] P. D. Francesco, I. Malavolta, P. Lago, Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption, in: 2017 IEEE International Conference on Software Architecture (ICSA), 2017, pp. 21–30. doi:10.1109/ICSA.2017.24.

[2] M. Söylemez, B. Tekinerdogan, A. Kolukısa Tarhan, Challenges and Solution Directions of Microservice Architectures: A Systematic Literature Review, Applied Sciences 12 (2022) 5507. doi:10.3390/app12115507.

[3] A. Bucchiarone, N. Dragoni, S. Dustdar, S. T. Larsen, M. Mazzara, From Monolithic to Microservices: An Experience Report from the Banking Domain, IEEE Software 35 (2018) 50–55. doi:10.1109/MS.2018.2141026.

[4] M. Villamizar, O. Garcés, H. Castro, M. Verano, L. Salamanca, R. Casallas, S. Gil, Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud, in: 2015 10th Computing Colombian Conference (10CCC), 2015, pp. 583–590. doi:10.1109/ColumbianCC.2015.7333476.

[5] O. V. Talaver, T. A. Vakaliuk, Reliable distributed systems: review of modern approaches, Journal of Edge Computing 2 (2023) 84–101. doi:10.55056/jec.586.

[6] J. Soldani, D. A. Tamburri, W.-J. Van Den Heuvel, The pains and gains of microservices: A Systematic grey literature review, Journal of Systems and Software 146 (2018) 215–232. doi:10.1016/j.jss.2018.09.082.

[7] S. Niedermaier, F. Koetter, A. Freymann, S. Wagner, On Observability and Monitoring of Distributed Systems – An Industry Interview Study, in: S. Yangui, I. Bouassida Rodriguez, K. Drira, Z. Tari (Eds.), Service-Oriented Computing, Springer International Publishing, Cham, 2019, pp. 36–52. doi:10.1007/978-3-030-33702-5_3.

[8] I. Pigazzini, F. A. Fontana, V. Lenarduzzi, D. Taibi, Towards Microservice Smells Detection, in: Proceedings of the 3rd International Conference on Technical Debt, TechDebt '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 92–97. doi:10.1145/3387906.3388625.

[9] T. Cerny, A. S. Abdelfattah, V. Bushong, A. Al Maruf, D. Taibi, Microservice Architecture Reconstruction and Visualization Techniques: A Review, in: 2022 IEEE International

Conference on Service-Oriented System Engineering (SOSE), 2022, pp. 39–48. doi:10.1109/SOSE55356.2022.00011.

[10] B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, C. K. Shanbhag, Dapper, a Large-Scale Distributed Systems Tracing Infrastructure, 2010. URL: https://api.semanticscholar.org/CorpusID:14271421.

[11] The OpenTracing Semantic Specification, 2023. URL: https://opentracing.io/specification/.

[12] OpenTelemetry, 2024. URL: https://opentelemetry.io/docs/what-is-opentelemetry/.

[13] OpenTelemetry Semantic Conventions, 2024. URL: https://opentelemetry.io/docs/specs/semconv/.

[14] OpenTelemetry Project Journey Report – 2023, 2023. URL: https://www.cncf.io/reports/opentelemetry-project-journey-report/.

[15] Observability Primer, 2023. URL: https://opentelemetry.io/docs/concepts/observability-primer/.

[16] G. Parker, S. Kim, A. A. Maruf, T. Cerny, K. Frajtak, P. Tisnovsky, D. Taibi, Visualizing Anti-Patterns in Microservices at Runtime: A Systematic Mapping Study, IEEE Access 11 (2023) 4434–4442. doi:10.1109/ACCESS.2023.3236165.

[17] I. U. P. Gamage, I. Perera, Using dependency graph and graph theory concepts to identify anti-patterns in a microservices system: A tool-based approach, in: 2021 Moratuwa Engineering Research Conference (MERCon), 2021, pp. 699–704. doi:10.1109/MERCon52712.2021.9525743.

[18] X. Guo, X. Peng, H. Wang, W. Li, H. Jiang, D. Ding, T. Xie, L. Su, Graph-Based Trace Analysis for Microservice Architecture Understanding and Problem Diagnosis, in: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2020, Association for Computing Machinery, New York, NY, USA, 2020, p. 1387–1397. doi:10.1145/3368089.3417066.

[19] Neo4j Local Clustering Coefficient, 2023. URL: https://neo4j.com/docs/graph-data-science/current/algorithms/local-clustering-coefficient/.

[20] Neo4j Degree Centrality, 2023. URL: https://neo4j.com/docs/graph-data-science/current/algorithms/degree-centrality/.

[21] Neo4j Strongly Connected Components, 2023. URL: https://neo4j.com/docs/graph-data-science/current/algorithms/strongly-connected-components/.

[22] OpenTelemetry Collector, 2023. URL: https://opentelemetry.io/docs/collector/.

[23] N. D. Boone, Dynamic Baseline Alerts Now Automatically Find the Best Algorithm for You, 2017. URL: https://newrelic.com/blog/how-to-relic/baseline-alerts-algorithm.

[24] J. Brownlee, A Gentle Introduction to Exponential Smoothing for Time Series Forecasting in Python, 2020. URL: https://machinelearningmastery.com/exponential-smoothing-for-time-series-forecasting-in-python/.

[25] S. Han, X. Hu, H. Huang, M. Jiang, Y. Zhao, ADBench: Anomaly Detection Benchmark, in: S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, A. Oh (Eds.), Advances in Neural Information Processing Systems, volume 35, Curran Associates, Inc., 2022, pp. 32142–32159. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/cf93972b116ca5268827d575f2cc226b-Paper-Datasets_and_Benchmarks.pdf.

[26] S. Suboh, I. Aziz, S. Shaharudin, S. Ismail, H. Mahdin, A Systematic Review of Anomaly Detection within High Dimensional and Multivariate Data, JOIV : International Journal

on Informatics Visualization 7 (2023) 122. doi:10.30630/joiv.7.1.1297.

[27] OpenTelemetry Demo, 2023. URL: https://github.com/open-telemetry/opentelemetry-demo.

[28] Neo4j APOC Library, 2023. URL: https://neo4j.com/developer/neo4j-apoc/.

[29] S. O. Semerikov, T. A. Vakaliuk, I. S. Mintii, V. A. Hamaniuk, V. N. Soloviev, O. V. Bondarenko, P. P. Nechypurenko, S. V. Shokaliuk, N. V. Moiseienko, V. R. Ruban, Development of the computer vision system based on machine learning for educational purposes, Educational Dimension 5 (2021) 8–60. doi:10.31812/educdim.4717.

[30] I. A. Pilkevych, D. L. Fedorchuk, M. P. Romanchuk, O. M. Naumchak, Approach to the fake news detection using the graph neural networks, Journal of Edge Computing 2 (2023) 24–36. doi:10.55056/jec.592.