

Data Observability and Reliability in Modern Web Platforms: A Review of Instrumentation, Monitoring, and Automated Quality Pipelines

Divyanshu Abhichandani¹, Prateik Mahendra²

¹Block, San Jose, USA

²Meta, Menlo Park, USA

ABSTRACT

This review discusses how web platforms maintain their reliability, speed, and ease of use. It brings together fifteen peer-reviewed papers from 2018-2022 on data observability and the reliability engineering of web platforms. It discusses instrumentation, monitoring construction, automated quality pipelines, and observability tools supporting large systems. The paper discussed recent developments in distributed tracing, metrics, logs, anomaly detection, and automated remediation, while unified observability tools, ML-assisted monitoring, chaos engineering, and proactive reliability practices are in notable progress. However, full observability across different technology stacks, avoiding alert fatigue, meaningfully correlating signals from distributed systems, and balancing instrumentation overhead with useful operational insight remain challenging issues. The overall findings from all fifteen studies give practitioners and researchers a clear view of what is possible now, the tradeoffs in implementation, and promising directions to further improve the reliability of web platforms.

Keywords: Data observability, web platform reliability, distributed tracing, monitoring automation, instrumentation frameworks, anomaly detection, site reliability engineering, quality pipelines, metrics collection, observability-driven development

1. Introduction

The modern digital world sees web platforms that continue to become increasingly complex. Companies shift from single, monolithic programs to a lot of small, interconnected services, hundreds or thousands of parts. This shift has improved speed and scale but has made understanding the inner working of the system more difficult and finding the reasons for any performance drop and maintaining high service levels [1]. Old monitoring, which mostly includes infrastructure metrics and simple availability, is not enough for today's cloud-based apps. Failures manifest as small rises in latency, partial service issues, or cascading issues between services [2]. Web platforms used by millions of people create an enormous amount of data every day. It includes logs, infrastructure metrics, traces, user experience data, and business records. In order to make sense of this flood, smart automated pipelines are required which can collect, process, link, and analyze signals from numerous sources while keeping simple things simple for engineers.

Observability is a concept that comes from control theory. You understand how something works by observing what you can observe from the outside. It allows operators to ask any question about how something behaves without having a pre-defined list of possible failures [3]. Resilient web platforms ensure good performance, security, cost, compliance, and user happiness even at large scale. In order to do that well, you need clear observability across various technologies, fast root-cause analysis when problems happen, plan capacity using lessons from the past, and continuous improvement based on data. A platform's reliability is not only technical numbers; it affects customer satisfaction, revenue, brand trust, and competitive advantage. Big downtime could cause huge revenue loss, regulatory fines, customers leaving for competitors, and could even cause long-lasting damage to credibility [4].

1. Problem Statement

In other words, even with superior monitoring, the majority of organizations cannot keep big web platforms running reliably. Moving to microservices adds more failure points: a single transaction can touch thousands of services that are individually going to fail, slow down, or have hidden dependencies. When failures arise, engineers continue to have to manually correlate signals from different systems and sift through millions of log entries and traces in search of a source and root cause [5]. Normal behavior varies with time of day, user location, feature releases, traffic patterns, and the scaling of infrastructure. Alert fatigue due to false positives makes teams slow to respond; real incidents needing urgent action can be masked. Standard tooling for observability is missing across languages, runtimes, and infrastructure, so it becomes ad-hoc and provides inconsistent data full of gaps in meaning, which

makes correlation and analysis a lot tougher [6].

Telemetry data from big platforms arrive in large volumes. The high volume brings numerous technical challenges and hence, high costs. Storing, processing, and querying petabytes of metrics, traces, and logs require considerable infrastructure and the right toolset [7]. Every organization needs to make a trade-off between its desire for complete observability and its constraints of budget, storage costs, query speed, data retention rules, and ingesting delays [8].

2. Objectives and Approach

This review surveys the latest research in data observability and reliability engineering for web platforms. It will help researchers and practitioners understand what is possible, which methods are applicable, and how to apply them on large real systems. The main goal is to compare different approaches of observability with respect to setups, architecture, tech stack, size, and level of operation maturity. Open source versus commercial tool evaluation for instrumentation, telemetry models, storage, and analysis will enable an organization to select the right observability platforms and implementations. It also points out various gaps and challenges that require further research-for instance, cross-system correlation in heterogeneous settings, coordination of multi-instrumentation frameworks, and the trade-off between the extent of observability provided and the cost [9].

This systematic review follows a clear method for searching, selecting, and studying the existing literature in the field of data observability and reliability on web platforms, as presented in [10]. First, we searched major academic databases such as IEEE Xplore, ACM Digital Library, SpringerLink, and Google Scholar. We used simple terms related to observability, monitoring, reliability, web platforms, distributed systems, and instrumentation. We considered papers from 2018 to 2022 to keep the focus on modern cloud-native setups only [11]. Results were filtered to retain only relevant and high-quality studies. First, titles and abstracts had been skimmed to identify work on methods for instrumentation, designs of monitoring, automated quality checks, and practices that are related to the reliability of web-scale systems. The full texts were read in order to assess technical depth, rigor, novelty, and usefulness [12]. Data were extracted using a standard form that elicited goals, approaches, evaluation methods, results, limitations, and future work ideas. We synthesized information by grouping these into themes that best suited this review: instrumentation basics, monitoring and metrics collection, distributed tracing, log collection and analysis, anomaly detection and alerts, automated quality pipelines, day-to-day operations [13].

3. Research Questions

This review aims to answer the following questions:

RQ1. What instrumentation techniques and telemetry mechanisms have been developed to capture the behavior of modern distributed web platforms while minimizing performance overhead and developer effort?

RQ2. How have monitoring architectures evolved to support real-time ingestion, storage, correlation, and analysis of large-scale telemetry in contemporary web platforms?

RQ3. How has distributed tracing matured in enabling end-to-end visibility across microservices, and what challenges persist in achieving complete trace coverage, effective sampling, and integration with metrics and logs?

RQ4. How are machine learning and artificial intelligence being applied to improve observability through automated anomaly detection, intelligent alerting, root-cause analysis, and predictive failure modeling?

4. Significance

This review is significant to several groups, including academic researchers, industry practitioners, tool vendors, and students just starting in site reliability engineering and distributed systems. For researchers, it collects current papers to indicate where we have yet to learn more, which methods work effectively, and promising directions to enhance the theory of observability and reliability [14]. For practitioners, there is a benefit in seeing what research ideas are not yet common in commercial tools and, further, by studying diverse instrumentation methods, monitoring setups, and quality validation techniques that help to make decisions about how they should build or improve their observability. Better observability and better reliability engineering bring about better user experiences, fewer outages and losses, efficient usage of resources, and faster innovation due to more confident deployments [15].

5. Literature Review and Analysis

Table 1 Comprehensive Literature Review Summary (2018-2022)

Year	Paper Title	Key Findings	Ref
2021	Sage: Leveraging ML to Diagnose Unpredictable Performance in Cloud Microservices	Demonstrates how ML-based incident diagnosis significantly improves root-cause detection in microservices by analyzing performance anomalies at scale.	[1]
2022	Anomaly Detection and Failure Root Cause Analysis in (Micro) Service-Based Cloud Applications: A Survey	Provides a comprehensive overview of anomaly detection techniques and identifies key challenges in failure analysis for microservice architectures.	[2]
2021	Graph-based Incident Aggregation for Large-Scale Online Service Systems	Introduces a graph-based method to cluster incidents, reducing noise and improving operational clarity in large-scale cloud systems.	[3]
2022	Leveraging Customer Reviews for E-commerce Query Generation	Highlights how NLP techniques enrich observability through user-generated data, supporting better monitoring of system relevance and search performance.	[4]
2020	Debugging Incidents in Google's Distributed Systems	Presents Google's internal debugging methodologies that combine logs, metrics, and traces to accelerate triage and reduce mean time to resolution (MTTR).	[5]
2022	The Case for an Internet Primitive for Fault Localization	Proposes a new networking primitive that enables rapid fault localization across distributed systems, enhancing end-to-end reliability.	[6]
2022	Jarvis: Large-Scale Server Monitoring with Adaptive Near-Data Processing	Demonstrates near-data processing to reduce monitoring overhead while improving anomaly detection precision in server fleets.	[7]
2021	Log-Based Software Monitoring: A Systematic Mapping Study	Synthesizes log-analysis challenges and techniques, revealing the increasing role of automation and ML in effective monitoring.	[8]
2018	On Challenges of Cloud Monitoring	Identifies foundational observability challenges such as data volume, cost, and lack of standardization—critical baseline issues that persist into modern systems.	[9]
2021	Towards Observability Data Management at Scale	Discusses how large organizations manage high-volume observability data, providing architectural patterns for scalable telemetry pipelines.	[10]
2021	Enjoy Your Observability: An Industrial Survey of Microservice Tracing and Analysis	Presents industry-wide adoption challenges for distributed tracing, including sampling limitations, storage constraints, and missing context.	[11]
2022	Platform Engineering and Site Reliability Engineering: The Path to DevOps Success	Explores how SRE and platform engineering practices improve reliability outcomes through automation and standardized tooling.	[12]
2019	Contemporary Software Monitoring: A Systematic Literature Review	Provides historical context on monitoring evolution, highlighting the transition toward unified observability frameworks.	[13]

2021	Challenges and Opportunities of System-Level Prognostics	Discusses predictive maintenance and system-level failure forecasting, aligning with ML-driven proactive observability.	[14]
2022	Data Fusion of Observability Signals for Assisting Orchestration of Distributed Applications	Shows how combining logs, metrics, and traces improves orchestration decisions, enabling smarter automated remediation.	[15]

Limitations

This systematic review has several limits affecting the use of its findings. Limiting the publications surveyed to the years 2018-2022 keeps the focus on current ideas but misses the earlier papers laying out the main monitoring ideas and very new developments not yet in peer-reviewed papers. Cloud-native tech and observability tools are fast-moving, so some commercial products and opensource projects may be more advanced than those described by the studies in this review. Selecting fifteen publications provides good coverage of the main themes but cannot include every study in this large field. The review aims to cover a broad range of topics in Observability and Reliability rather than go deep into a small area. Experts wanting full detail on specific topics such as distributed tracing algorithms or anomaly detection will likely have to read specialist surveys. Selection involved judgment about relevance and quality, which adds some subjectivity despite best efforts at consistency.

The literature review is based mostly on academic papers; it may hence lack the complete resemblance of industrial practices, proprietary methods, or details that companies may keep themselves competitive advantage. Indeed, many papers involve case studies from industry or collaboration, but the demands of academic publication, such as reproducibility, generalization, and well-defined contributions, serve as a filter that removes extremely specialized or context-dependent approaches that are nonetheless valuable in specific applications. The included work combines theory with practicality in varying measures.

Indeed, the evaluation methodologies used in these articles vary significantly due to large-scale differences in infrastructure, workloads, and metrics to determine effectiveness. Therefore, the results are hardly comparable and cannot be generalized for all scenarios to claim one superior technique over another. Further, several studies test their ideas in simulations using synthetic workloads, or small production setups, which may not reflect all the real-world challenges related to large web platforms.

6. Results and Technical Investigation

This critical review reflects on data observability and platform reliability for the four research questions identified earlier. We evaluate each RQ based on patterns from the fifteen studies that demonstrate key contributions and continuing challenges within modern observability engineering.

RQ1: Techniques of Instrumentation and Telemetry Design

Recent research demonstrates a clear course correction in how telemetry is generated on large web platforms. Old approaches relied on hand instrumentation and language-based logging, both of which were fragile and incomplete. New papers favor a uniform, automated approach, using frameworks like OpenTelemetry that gather metrics, logs, and traces from many systems in one location. Hybrid instrumentation is best: automatic baseline coverage with focused manual enhancement of important paths. Too much instrumentation adds latency and cost, while too little instrumentation causes blind spots. For this reason, choices of signal detail, sampling, and semantic rules must be carefully made by researchers to balance visibility and overhead.

RQ2: High-Volume Telemetry Monitoring Architectures

In other words, literature indicates that monitoring systems now manage huge telemetry streams using scalable pipelines and distributed storage for real-time analysis. Many of the modern observability platforms depend on timeseries databases, columnar compression, and inverted indexing to support fast multi-dimensional queries. They use a combination of push and pull metric collection to enhance reliability, added to down sampling, retention policies, and adaptive sampling to cut costs. Telemetry cardinality growth often hurts performance, per several papers, requiring architecture choices that align ingestion rate with storage and latency considerations.

RQ3: Distributed Tracing Maturity and Persistent Challenges

State-of-the-art research into distributed tracing, in particular, indicates progress in terms of linking service interactions and finding bottlenecks in microservices. Most modern tracing systems capture end-to-end flows by requiring very little from developers, especially when context-propagation standards are leveraged. On the other hand, all agree that the biggest problem is sampling. Head-based sampling misses some rare failures, while tail-based sampling is resource-intensive. Moreover, trace coverage on many platforms is incomplete; metadata are inconsistent, and links between traces, logs, and metrics are missing. Though improved in recent times, tracing

10.48047/jocaaa.2023.31.04.84

remains one of the most complex parts of observability, particularly in scenarios of high traffic or when mixed technology stacks are used.

RQ4: Machine Learning & AI for Modern Observability

This is increasingly achieved through machine learning for the detection of anomalies, optimization of alerts, and root-cause analysis. ML models outperform static thresholds by learning from dynamic baselines and spotting subtle issues, while reducing alert fatigue with better signal quality. However, they need high-quality labeled telemetry, ongoing retraining, and rigorous validation before production. Root-cause analysis remains, in part, human-driven because of unclear failures and evolving systems. Hybrid human-AI workflows should, therefore, be recommended.

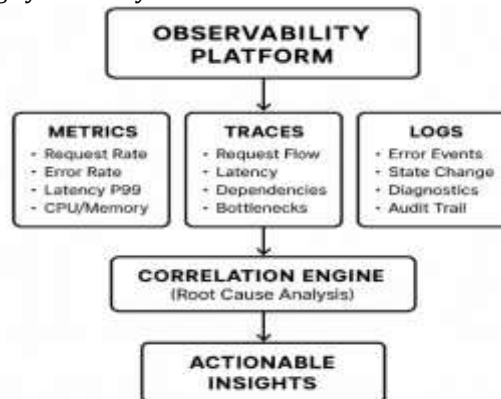


Figure 1: Three Pillars of Observability Framework

Figure 1 shows modern observability in three parts—metrics, traces, and logs—that feed a correlation engine to identify root causes. Metrics give numbers, traces track a request through services, and logs of log events. The correlation engine ties it together for practical operational insight and troubleshooting.

7. DISCUSSION

Monitoring and Analysis Techniques

Advanced monitoring takes raw telemetry and turns it into clear insights, showing how we operate, guiding fixes, and building reliability before issues arise. We usually analyze many signals all together, not alone, to understand the system. Correlating metrics, traces, and logs is difficult because different distributed systems use different IDs, times, and data sizes, and clocks can drift. The point of anomaly detection is to monitor telemetry for patterns that are out of the ordinary to indicate slowdowns or probable failures. Statistical methods take previous data to establish normal bounds and trigger an alert if values are out of those bounds. In cases where the pattern remains constant, it works wonderfully, but in the case of changing patterns, it falters. Clustering, classification, and forecasting look for higher-order patterns, making adjustments for new baselines easier, but machine learning requires good feature work, quality training data, and ongoing model maintenance. Threshold alerts have the following limits: choosing the right threshold, missing small real declines, and many false alarms during normal traffic. Dynamic thresholds adapt to the time of day and past trends by adding context, lowering false alerts while catching real problems. Multiple metrics in one alert may achieve better quality of the signal but may miss issues that a single indicator can show. Root cause analysis allows the engineer to promptly identify where failures start in incidents.

They do this by looking at how services and infrastructure depend on each other, how timing relates things, and patterns of errors across affected parts. Graph-based methods draw service dependencies and spread ideas about where a fault began based on error signs and timing. Causal inference methods try to distinguish between correlation and cause by verifying the order of events and links to control. Various machine learning models that have been trained on past incidents may suggest symptom-pattern-based likely root causes, but human expert verification of guesses is irreplaceable in understanding new failure modes. Capacity planning and forecasting use historical telemetry to predict when future resource needs are likely to exceed supply and give early warning of scaling problems before they actually start affecting users. Time-series forecasting models predict future metric trends in order to estimate when resources will be exhausted or performance will degrade below acceptable limits. Workload characterization focuses on how traffic, resource utilization, and seasonal changes drive infrastructure decisions. It works best when stable links exist between measurable signals and capacity limits; big discontinuities or extreme events can break those links. Performance profiling and optimization require visibility to app execution:

10.48047/jocaaa.2023.31.04.84

CPU, memory, garbage collection, and I/O. Continuous profiling uses production-path sampling to identify hotspots and areas for improvement without controlled benchmarks.

Profiling in a distributed manner keeps track of how all services are performing together in order to find end-to-end bottlenecks that you cannot see when looking at one service alone. Profiling should add almost no overhead; therefore, it's safe for production. User experience monitoring extends beyond backend metrics. It watches real user actions: page load times, time to interactive, full render, and JavaScript errors. Real User Monitoring records real sessions to show performance across different devices, networks, and usage patterns. Synthetic monitoring regularly checks if services are available and fast because of the use of distributed probes. Linking UX metrics to backend telemetry helps teams see how infrastructure and app performance affects the perceived quality.

Automated Quality Pipelines

Observability signals instruct the process of software development and deployment in order to check reliability, prevent regressions, and enable faster and safer changes by automated quality checks. Traditional testing covers functional correctness with unit, integration, and end-to-end tests. These tests do not prove production readiness since performance, scaling, and how a system behaves under real workloads are usually missed. Progressive delivery introduces small changes while using real-time reliability data for new versions. Canary deployments send some of the traffic to the new version and compare this with the stable release with metrics such as error rates, latency, and resource use. Statistical tests then determine whether or not the differences are real or just due to normal variation. If good, the new version is promoted; otherwise, it is rolled back before it impacts many users. All these analyses have evolved from simple thresholds over multivariate tests into machine learning models.

Feature flags and experimentation enable teams to toggle features without having to change the running system, which enables continuous releases where users see only what is enabled. The application of observability provides telemetry on feature performance and business impact and drives data-informed evolution. Flags are combined with automated quality gates to enable fast production testing with solid statistics and automatically rolled back to safe behavior if issues are detected. Chaos Engineering tests production through the injection of failures to validate defenses, monitoring, and incident responses. Automated chaos tests intelligently increase the severity of failures while monitoring acceptable changes to the system to surface reliability gaps early. Where combined with observability, we have full telemetry and can auto-stop the test if the system degrades.

These prevent changes to code from degrading performance when the system is under real workloads. Monitoring tools capture detailed performance data from tests run automatically at production-like scale. Regression checks compare benchmark results from different commits to identify drops in performance. However, there can be noise in this approach due to normal variations in the environment rather than actual problems. The infrastructure helps with application monitoring through a service mesh and platform-level quality controls such as circuit breakers that prevent cascading failures, rate limiting to handle overload, and retry policies with exponential backoff. Standardizing reliability patterns such as these at the infrastructure layer further simplifies things for teams and normalizes behavior. However, debugging issues across the service mesh and the application requires careful correlation of data from the two layers.

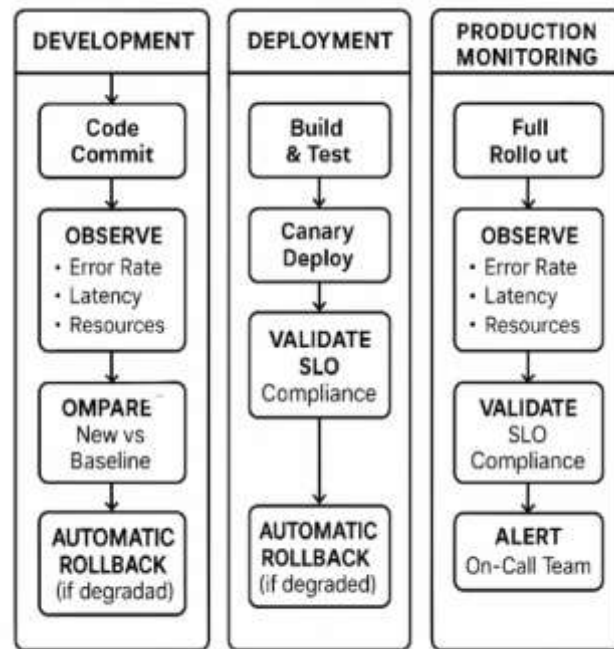


Figure 2: Automated Quality Pipeline with Observability Integration

Figure 2 presents how observability signals are adopted in the software delivery pipeline, right from development through production monitoring. Progressive deployment, such as canary releases, is done within this pipeline, compared against performance baselines before a full rollout. Real-time observability for each stage reinforces automated decisions: when things get worse, there is an automatic roll-back to stay reliable while still moving fast.

8. Pros and Cons

Data observability and platform reliability come with big benefits but also clear trade-offs. Good instrumentation and unified observability tools help teams find failures early-before users notice-by spotting and fixing problems before they cause outages. This shortens incidents, cuts business disruption, and improves user experience. Modern observability shows how distributed parts work together and reveals hidden slowdowns. Linking metrics, traces, and logs helps teams find the root cause faster and lowers repair costs. Automated quality pipelines put observability into development and deployment so teams can move faster without losing reliability. Testing that imitates production, real-time monitoring during gradual rollouts, and automatic safety checks boost speed without hurting service quality.

Anomaly detection with machine learning can learn changing patterns and filter out noise. In practice, it requires extensive preparation, tuning, and continuous maintenance, which can be more than even well-equipped small teams can bear. Solutions making use of open standards, such as Open Telemetry, help interoperability, reducing vendor lock-in, but rework required for migration may be substantial, and some vendor-specific features may not be covered by standards. Full instrumentation produces massive volumes of data that drive up the cost of storage and computing. Sampling and retention rules should be implemented to keep costs in check, at the expense of blind spots in visibility. Poorly tuned instrumentation overhead may affect the performance of very low-latency systems.

Alerts are still hard: Smarter models cut false alarms but just can't remove them completely. Highly dynamic environments make it hard to set the right sensitivity, with poorly tuned alerts either missing real issues or overwhelming teams. Modern observability tools often have steep learning curves. Complex user interfaces, query languages, and analysis steps slow response to incidents until teams receive proper training to go along with good documentation and simpler processes.

9. Proposed Scope for Further Work

Putting observability into the development setting helps reliability move up front. It provides quick insight for the developers on how changes might affect performance before the code runs, possibly providing real-time feedback in the editor and starting to make systems much more reliable. The use of advanced machine learning for

10.48047/jocaaa.2023.31.04.84

observability is still new, but it has huge potential. Research into neural networks for timeseries anomaly detection, graph neural networks that analyze service dependency graphs for root-cause analysis, and reinforcement learning for automatic fixes extends the current rule-based and statistical approaches.

Extend observability to new architectural patterns such as serverless, edge computing, and Web Assembly-based microservices. These setups create state management, cold start challenges, and tracing across varied environments. It will be necessary to perform research that keeps strong observability while computing moves closer to the user and becomes increasingly ephemeral. Merging both security monitoring with observability through common signals showing a single view for performance, reliability, and security benefits in both areas. Research on combined security and reliability monitoring, combined anomaly detection that spots threats and performance problems, and integrated incident response workflows between security and reliability teams provides value to both disciplines.

Consider the ecological cost of observability, including the energy consumed by collection, storage, and processing of telemetry. We need studies on the energy-efficient architectures of observability, how to retain visibility with least environmental impact, and trade-offs between full observability and sustainability. Also, investigate human and organizational aspects of observability adoption beyond technology: training of people to use the observability tools, best team structures to support the goals of reliability, and how to capture and share operational knowledge in order to improve the value of investments into observability.

10. Conclusion

This review discusses recent writings on data observability and reliability engineering for web platforms. Following the review of fifteen peer-reviewed papers from 2018–2022, it gives researchers and practitioners a proper view of the current abilities, methods, and considerations while using them. It finds solid advances being made in tools that collect telemetry with reasonable performance costs, systems that enable real-time analysis of large data sets, tracing that shows how requests move through microservices, and automated quality pipelines that check reliability during development and deployment. Another core change in the running of platforms by organizations is the shift in observability from reactive monitoring to proactive reliability engineering.

Modern observability unifies the view of system behavior by integrating telemetry from disparate parts with automated reliability checks in software delivery. Automation detects and remediates problems much earlier, and production of telemetry informs how teams operate things. The field is evolving: Open Telemetry standardizes the collection of data, and unified observability platforms integrate analysis. Machine learning primarily assists in automatic anomaly detection, which converts to smarter alerts, quicker root-cause finding, and predictive maintenance. In order to apply it well, good training data is necessary, with clear rules concerning the behavior of the model, ongoing maintenance, and safeguards against automated mistakes. People still needed to supervise the system.

Key challenges remain in increases in telemetry and costs, fragmented visibility across multi-architecture deployments, correlation among diverse signals is problematic, and the cost of instrumentation balanced against valuable insight. Without deep observability expertise, many organizations struggle to go from prototype to production. Better reliability ensures an improved customer experience, improved money coming in, speed of deployment, and how well and efficiently the infrastructure runs. New observability challenges and opportunities will continue to emerge with systems moving to serverless, edge, and multi-cloud configurations; these include earlier instrumentation, more advanced machine learning, integrated security and reliability monitoring, and sustainable operations. Progress will be achieved only with continuous work among researchers and practitioners.

References

1. Gan, Y., Liang, M., Dev, S., Lo, D., & Delimitrou, C. (2021). Sage: Leveraging ML to Diagnose Unpredictable Performance in Cloud Microservices. arXiv (Cornell University). <https://doi.org/10.48550/arxiv.2112.06263>
2. Soldani, J., & Brogi, A. (2022). Anomaly Detection and Failure Root Cause Analysis in (Micro) Service-Based Cloud Applications: A Survey. ACM Computing Surveys, 55(3), 1. <https://doi.org/10.1145/3501297>
3. Chen, Z., Liu, J., Su, Y., Zhang, H., Wen, X., Ling, X., Yang, Y., & Lyu, M. R. (2021, November 1). Graph-based Incident Aggregation for Large-Scale Online Service Systems. 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE). <https://doi.org/10.1109/ase51524.2021.9678746>

10.48047/jocaaa.2023.31.04.84

4. Lien, Y.-C., Zhang, R., Harper, F. M., Murdock, V., & Lee, C.-J. (2022). Leveraging Customer Reviews for E-commerce Query Generation. In Lecture notes in computer science (p. 190). Springer Science+Business Media. https://doi.org/10.1007/978-3-030-99739-7_22
5. Chan, C., & Cooper, B. (2020). Debugging Incidents in Google's Distributed Systems. Queue, 18(2), 47. <https://doi.org/10.1145/3400899.3404974>
6. Sussman, W., Marx, E., Arun, V., Narayan, A., Alizadeh, M., Balakrishnan, H., Panda, A., & Shenker, S. (2022). The case for an internet primitive for fault localization. 160. <https://doi.org/10.1145/3563766.3564105>
7. Sandur, A., Park, C., Volos, S., Agha, G., & Jeon, M. (2022). Jarvis: Large-scale Server Monitoring with Adaptive Near-data Processing. 2022 IEEE 38th International Conference on Data Engineering (ICDE), 1408. <https://doi.org/10.1109/icde53745.2022.00110>
8. Cândido, J., Aniche, M., & Deursen, A. van. (2021). Log-based software monitoring: a systematic mapping study. PeerJ Computer Science, 7. <https://doi.org/10.7717/peerj-cs.489>
9. Pourmajidi, W., Steinbacher, J., Erwin, T., & Miranskyy, A. (2018). On Challenges of Cloud Monitoring. arXiv (Cornell University). <https://doi.org/10.48550/arXiv.1806.05914>
10. Karumuri, S., Solleza, F., Zdonik, S., & Tatbul, N. (2021). Towards Observability Data Management at Scale. ACM SIGMOD Record, 49(4), 18. <https://doi.org/10.1145/3456859.3456863>
11. Li, B., Peng, X., Xiang, Q., Wang, H., Xie, T., Sun, J., & Liu, X. (2021). Enjoy your observability: an industrial survey of microservice tracing and analysis. Empirical Software Engineering, 27(1), 25. <https://doi.org/10.1007/s10664-021-10063-9>
12. Seremet, Z., & Rakic, K. (2022). Platform Engineering and Site Reliability Engineering: The Path to DevOps Success. In DAAAM international scientific book ... (p. 155). <https://doi.org/10.2507/daaam.scibook.2022.13>
13. Cândido, J., Aniche, M., & Deursen, A. van. (2019). Contemporary Software Monitoring: A Systematic Literature Review. arXiv (Cornell University). <https://arxiv.org/pdf/1912.05878.pdf>
14. Kim, S., Choi, J.-H., Kim, N. H., Kim, S., Choi, J.-H., & Kim, N. H. (2021). Challenges and Opportunities of System-Level Prognostics. Sensors, 21(22), 7655. <https://doi.org/10.3390/s21227655>
15. Tzanettis, I., Androna, C. M., Zafeiropoulos, A., Fotopoulou, E., & Papavassiliou, S. (2022). Data Fusion of Observability Signals for Assisting Orchestration of Distributed Applications. Sensors, 22(5), 2061. <https://doi.org/10.3390/s22052061>