TIMING BEHAVIOR ANOMALY DETECTION IN ENTERPRISE INFORMATION SYSTEMS

Matthias Rohr, Simon Giesecke and Wilhelm Hasselbring
Graduate School TrustSoft, Software Engineering Group, University of Oldenburg, 26111 Oldenburg, Germany
{*rohr*|*giesecke*|*hasselbring*}*@informatik.uni-oldenburg.de*

Keywords: Software Measurement, Software Engineering, Anomaly Detection, Failure Diagnosis.

Abstract: Business-critical enterprise information systems (EIS) have to satisfy high availability requirements. In order to achieve the required availability, automatic failure detection and diagnosis techniques must be used. A major cause of failures in EIS are software faults in the application layer. In this paper, we propose to use anomaly detection to diagnose failures in the application layer of EIS. Anomaly detection aims to identify unusual system behavior in monitoring data. These anomalies can be valuable indicators for availability or security problems, and support failure diagnosis. In this paper we outline the basic principles of anomaly detection, present the state of the art, and typical application challenges. We outline a new approach for anomaly detection in Enterprise Information Systems that addresses some of these challenges.

1 INTRODUCTION

Business-critical enterprise information systems (EIS) have to satisfy high availability requirements. The availability of a system is determined through its reliability (probability of failure free operation over a period of time) and the time required for its repair after a failure has occurred. To achieve a higher availability, the reliability must be increased or the time to repair must be reduced. Automatic failure diagnosis is a strategy to reduce repair times.

Anomaly detection is a promising strategy to automatic failure diagnosis in EIS. Its main idea is to use "unusual" system runtime behavior as an indicator of a problem in the system. System runtime behavior is considered unusual if it deviates from the behavior that was observed earlier. Anomaly detection requires an instrumentation to monitor system runtime behavior. Even without anomaly detection, monitoring data are a valuable source of information in failure diagnosis, especially for failures that are difficult to reproduce. However, as enterprise information systems typically produce a large amount of monitoring data, tool support for anomaly detection is required to efficiently realize failure diagnosis.

This paper motivates the use of an anomaly detection method to increase the dependability of EIS. Existing approaches that apply anomaly detection to EIS are presented and open research questions are identified. Additionally, we outline a new approach for anomaly detection based on timing behavior analysis for distributed, multi-user, enterprise-scale software systems.

This paper is structured as follows: Sections 2 and 3 present the basic concepts of anomaly detection and its application to EIS. A new approach for anomaly detection in EIS is introduced in Section 4 before the paper is summarized in Section 5.

2 FOUNDATIONS OF ANOMALY DETECTION

A *(behavior) anomaly* of a system is a significant deviation from the behavior that was observed earlier. For instance, exceptional high or low component service response times are timing behavior anomalies. The incidence of an anomaly does not automatically imply that the system behavior is incorrect (Maxion, 1990). For example, exceptionally long component service response times may be caused by an excep-

tionally high number of current system users.

The threats to dependability are faults, errors, and failures (Avižienis et al., 2004). In short, a *(system) failure* is the incidence of system behavior that is *incorrect* with respect to stated requirements. A failure is caused by *faults*. Active faults cause incorrect states, which are called *errors*. When using anomaly detection in failure diagnosis (e.g., failure detection, fault localization), it is assumed that active faults and failure processes cause anomalies.

Anomaly detection requires some criterion to discriminate normal and unusual behavior. Many anomaly detection methods require the manual specification of static thresholds for this. Static thresholds require expert knowledge on the behavior of the system under operation and may have to be updated every time the system or its operation conditions changes. More maintainable anomaly detection techniques automatically learn threshold models by statistical analysis of historical system behavior from log files. Usually, only data describing normal system behavior are available (Maxion and Olszewski, 1993), while training data for particular faults or failures is unavailable, in particular for fault classes that have never occurred in the system before.

A typical architecture of an anomaly detection failure diagnosis system monitors different system behavior metrics at measurement points at multiple places in the system architecture. A separate anomaly detector is used for each source of measurement data. In combination, the anomaly detectors produce a pattern of anomalies that is evaluated by so-called *event correlation* (Steinder and Sethi, 2004) for a final failure diagnosis.

In the domain of network management or communication infrastructure management, anomaly detection is used for failure diagnosis by analyzing the monitoring data provided by communication middleware (e.g., Maxion, 1990; Hoke et al., 2006).

In industrial manufacturing, anomaly detection has been well-studied for failure diagnosis based on mathematical prediction models or using computation intelligence methods (Bocaniala and Palade, 2006).

In system security engineering, many intrusion detection approaches assume that user or system behavior is similar to historic data. Hence, strong changes of user or system behavior are indicative of intrusions. For instance, Denning (1987) presented an approach for detecting deviations from the normal sequence of user commands.

2.1 Typical Challenges

This subsection presents typical challenges of anomaly detection. Our approach presented in Section 4 addresses some of these challenges.

False Alarms. A false alarm occurs when an anomaly is mistakenly judged a failure. A false alarm may lead to wrong failure handling or reduce the acceptance by administrators.

Usage Dependence. Changes in the amount or types of user behavior can lead to a changes in system runtime behavior. For instance, the “normal” average response time depends on the degree of resource sharing by concurrent requests.

Modeling Uncertainty, Non-linearity of System Runtime Behavior. In complex systems, behavior metrics such as response times are difficult to predict, because not all relevant influences and dynamics can be efficiently modeled. Therefore, it is difficult to develop criteria that can be used to classify runtime behavior as either normal or unusual. Additionally, only a part of all operational conditions can be monitored, because a complete monitoring causes a monitoring overhead that it is too large to be tolerable.

3 ANOMALY DETECTION IN ENTERPRISE INFORMATION SYSTEMS

Several runtime behavior metrics, such as service response times, memory utilization, or trace shapes are candidates for anomaly detection in EIS. Since timing behavior can be efficiently monitored, it is one of the most common characteristics used to detect operational anomalies. System timing behavior or trace shapes in enterprise information systems are usually strongly dependent on the context and on the system usage. It has been demonstrated that statistical analysis of timing behavior can still be effectively used for failure diagnosis in large software systems.

For instance, Agarwal et al. (2004) demonstrated that timing behavior anomalies are a discriminatory indicator in fault localization. Mielke (2006) reported that end-to-end response times in Enterprise Resource Planning systems are well-described by log-normal distributions. Deviations from this distribution often indicate performance problems (Mielke, 2006).

Maintainability is a major requirement of failure diagnosis tools for EIS. EIS are regularly modified because of changed market requirements, to offer new products, or to increase the level of integration. Traditional software fault tolerance methods, such as multi-version design, multiply the efforts required for changes in system design. A failure diagnosis approach suitable for EIS should be maintainable, i.e., it should be easily adaptable to changes in the system and should not reduce the changeability of the EIS by imposing strict architectural constraints. Two critical maintainability aspects of anomaly detection are the monitoring instrumentation and the threshold specification method.

In the last years, some first approaches to anomaly detection for failure diagnosis in EIS have been proposed. For instance, Agarwal et al. (2004) evaluate response times of internal components using automatically generated threshold models derived form historical violations of end-to-end response time requirements and component dependency graphs. Kiciman (2005) presents an anomaly detection method that is applied to structural system behavior such as component interactions and the "shape" of service execution sequences.

4 APPROACH TO TIMING BEHAVIOR ANOMALY DETECTION

The general strategy of our approach is as follows: The target system is instrumented to monitor operation response times and operation execution sequences to build a profile of "normal" system behavior. For failure diagnosis, for each recent response time the degree of anomaly is computed that quantifies the "usualness" of a response time in the context of the execution conditions given by the operation execution sequences. All anomaly degrees are evaluated by statistical pattern analysis to identify a possible path of a failure process through the software architecture over time, or to discriminate types of failures.

The system is assumed to be assembled out of software components, i.e. reusable building blocks of software offering operations via interfaces. Furthermore, the allocation of software components to execution environments must be known. Each *user request* is processed by a separate *request process* that manages the execution of required component services. A request process has a unique *request identifier*. The middleware platform manages the execution for concurrent user requests by starting a separate *request process* for each incoming user request. The sequence of operation executions for a user request is called *request trace*.

Workload Awareness. In multi-user systems, service response times depend on the current level of concurrent system usage. We define the workload metric $w_{d,t}$ as the number of active request processes for a deployment context d at time t. A request process is active, iff it is not currently waiting for responses from other deployment contexts. This simple metric gives a coarse approximation of resource utilization without the need for additional monitoring.

Our research is based on the hypothesis that timing behavior evaluation is improved by explicitly modeling the workload, i.e. influences on timing behavior that arise from hardware resources being shared by concurrent user requests. From another point of view, the higher robustness against workload changes extends the applicability to multi-user applications in the first place by relieving the requirement of absence of concurrent system usage.

User Request Awareness. The execution of operations can heavily depend on particular request properties such as input parameter values or the origin. For instance, consider an operation that is the only access point exposed by a large legacy system. This operation is used both for requests with short response times from private customers and for batch jobs with long response times. We expect that the evaluation of the response times of this operation can benefit from taking the execution sequences that precede the call into account.

Including the execution sequences into timing behavior anomaly detection improves the awareness of individual characteristics of user requests. Since not all variations in the execution sequence are relevant to timing behavior analysis, we identify patterns within operation execution sequences (traces) that have shown distinguished timing behavior. If such patterns are discovered, they will be handled independently in the timing behavior evaluation, by being represented through separate probability density functions of operation response times. We have the hypothesis that timing behavior evaluation can be more accurate if such trace patterns are explicitly considered.

Maintainability. We employ source code instrumentation (monitoring logic is integrated to the source code) to realize maintainable timing-behavior monitoring for enterprise-scale systems. This is in contrast to middleware-interception methods that use

proprietary interfaces of middleware products or manipulates middleware for monitoring. A disadvantage of code instrumentation is that the primary business logic of the software is mixed with monitoring logic, which may reduce the readability of the code (and therefore, the maintainability). However, code instrumentation avoids dependencies on manipulated middleware or proprietary middleware interfaces. To reduce the problem of mixing business and monitoring logic in the source code, we employ an approach based on Aspect-Oriented Programming (AOP). AOP is a technique to isolate cross-cutting concerns, such as monitoring, from the primary business logic. In a previous case study, we evaluated the monitoring overhead and the maintainability of this monitoring approach in a large business system of a German telecommunication provider (Focke et al., 2007).

Additionally, maintainability is addressed by automatically creating the profile of normal timing behavior from historical data. To derive the model, no intrusive changes to the software system are required, it is merely necessary to perform the system instrumentation as described in the previous section.

5 CONCLUSION

This paper motivates anomaly detection for failure diagnosis in enterprise information systems. Anomaly detection allows to target failures caused by faults in the application-layer. The problem of anomaly detection was successfully targeted for failure diagnosis in industrial manufacturing, network management, or system security. Anomaly detection for failure diagnosis in EIS is far less studied, but promising first results indicated the potential benefits.

We propose a new approach to anomaly detection for EIS based on timing behavior analysis. The major concepts are workload awareness, and user request awareness to improve the anomaly detection quality by reducing the dependence to the changes in the operational profile, which should reduce the number of false alarms. From another point of view, the higher robustness against workload changes increases the applicability to multi-user applications by omitting the requirement of non-concurrent system usage. The maintainability of our monitoring approach has been shown during an evaluation in an EIS of a telecommunication company. In future work we will perform an evaluation of the anomaly detection approach.

We discussed that the dynamic nature of enterprise application systems requires that in particular the maintainability of a failure diagnosis approach for EIS is important and outlined that anomaly detection can be realized in a maintainable way.

ACKNOWLEDGEMENTS

This work is supported by the German Research Foundation (DFG), grant GRK 1076/1.

REFERENCES

Agarwal, M. K., Appleby, K., Gupta, M., Kar, G., Neogi, A., and Sailer, A. (2004). Problem determination using dependency graphs and run-time behavior models. In *15th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM'04)*, volume 3278 of *Lecture Notes in Computer Science*, pages 171–182. Springer.

Avižienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33.

Bocaniala, C. D. and Palade, V. (2006). Computational intelligence methodologies in fault diagnosis: Review and state of the art. In *Computational Intelligence in Fault Diagnosis*, Advanced Information and Knowledge Processing, chapter 1, pages 1–36. Springer.

Denning, D. (1987). An intrusion-detection model. *IEEE Transactions on Software Engineering*, 13(2):222–232.

Focke, T., Hasselbring, W., Rohr, M., and Schute, J.-G. (2007). Instrumentierung zum Monitoring mittels Aspekt-orientierter Programmierung. In *Proceedings Software Engineering 2007, Hamburg*, GI-Edition – Lecture Notes in Informatics. Bonner Köllen Verlag.

Hoke, E., Sun, J., Strunk, J. D., Ganger, G. R., and Faloutsos, C. (2006). Intemon: continuous mining of sensor data in large-scale self-infrastructures. *SIGOPS Oper. Syst. Rev.*, 40(3):38–44.

Kiciman, E. (2005). *Using Statistical Monitoring to Detect Failures in Internet Services*. PhD thesis, Stanford University.

Maxion, R. A. (1990). Anomaly detection for network diagnosis. In Randell, B., editor, *Proceedings of the 20th International Symposium on Fault-Tolerant Computing (FTCS '90)*, pages 20–27. IEEE.

Maxion, R. A. and Olszewski, R. T. (1993). Detection and discrimination of injected network faults. In *Digest of Papers of the 23rd Internation Symposium on Fault-Tolerant Computing*, pages 198–207. IEEE.

Mielke, A. (2006). Elements for response-time statistics in ERP transaction systems. *Performance Evaluation*, 63(7):635–653.

Steinder, M. and Sethi, A. S. (2004). A survey of fault localization techniques in computer networks. *Science of Computer Programming*, 53(2):165–194.