Lehre & Studium

ENTWICKLUNG EINER PROTOTYPISCHEN ANWENDUNG ZUR ANALYSE DER INTERAKTION VON OPC UA CLIENTS MIT EINEM OPC UA SERVER

LUKAS RUF (BACHELORSTUDIUM INFORMATIK)

Betreuer: Prof. Dr. Marcel Tilly, Prof. Dr. Wolfgang Mühlbauer

1.Einleitung und Zielsetzung

In der Industrie 4.0 ist die standardisierte Kommunikation zwischen Maschinen und übergeordneten Steuerungssystemen oft anzutreffen. Die Open Platform Communications Unified Architecture (OPC UA) hat sich hierfür als führendes, plattformunabhängiges Protokoll etabliert [1]. Gegenstand meiner Arbeit ist die Analyse der Interaktion zwischen OPC UA Clients und Servern. Das primäre Ziel bestand in der Konzeption und Entwicklung einer prototypischen Diagnoseanwendung, die anomales Verhalten von Clients erkennt und Systemintegratoren sowie Servicetechnikern ein konfigurierbares Werkzeug zur Verfügung stellt [2]. Da Fehlkonfigurationen oder ineffiziente Client-Implementierungen im industriellen Umfeld zu signifikanten Produktionsausfällen führen können, fokussiert sich die Anwendung auf die Detektion von Ineffizienzen und protokollwidrigem Verhalten, ohne den Server selbst invasiv modifizieren zu müssen.

2.Methodik und Versuchsaufbau

Zur Detektion von Anomalien evaluierte ich diverse Parameter, darunter die Häufigkeit von Verbindungsaufbauten, exzessives Read Polling, Fehlerquoten sowie die Speicherauslastung (Resident Set Size, RSS) des Serverprozesses [3]. Ein Machine-Learning-Ansatz wurde aufgrund fehlender annotierter Datensätze zugunsten regelbasierter, konfigurierbarer Heuristiken verworfen und zukünftigen Arbeiten überlassen.

Der Kern meiner Methodik basiert auf der nicht-invasiven, passiven Beobachtung. Wie in Abbildung 1 dargestellt, agiert die entwickelte Applikation als Performance Tracker. Der OPC UA Server protokolliert seine Aktivitäten in ein Serverlog. Der Tracker liest dieses Log mittels eines Logwatchers parallel aus und korreliert die Ereignisse mit der Arbeitsspeicherauslastung. Externe Testapplikationen simulieren dabei sowohl reguläre als auch anomale Lastszenarien. Der Aufbau neuer Sessions ist ein ressourcenintensiver Vorgang [4], [5], weshalb ineffiziente Session-Erstellungen gezielt überwacht werden.

Die in Abbildung 1 visualisierte Architektur gewährleistet eine strikte Trennung von Server und Analyseinstrument. Dies ist insofern von Bedeutung, als dass typische Produktionsumgebungen keine Modifikation der laufenden Steuerungssoftware zulassen. Durch das Auslesen der Logs und das Monitoring auf Betriebssystemebene bleibt die Integrität des Servers und die Portabilität des Trackers gewahrt. Steigt der Speicherbedarf signifikant, während die Event-Rate konstant bleibt, droht im schlimmsten Fall die Terminierung durch den OOM-Killer (Out of Memory) des Betriebssystems [6], [7].

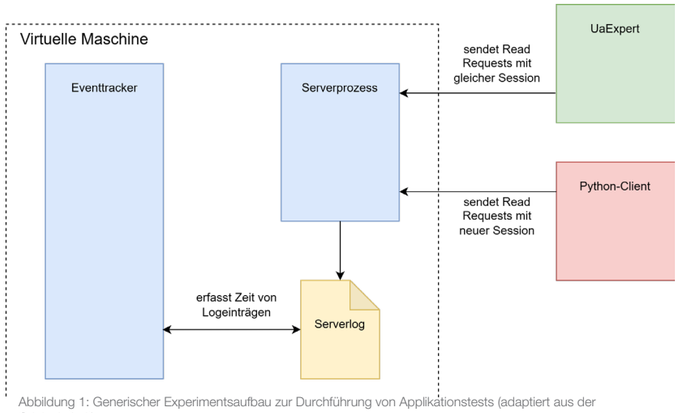

Abbildung 1: Generischer Experimentsaufbau zur Durchführung von Applikationstests (adaptiert aus der Originalarbeit).

3.Implementierung und Lösung

Die Implementierung des Prototyps erfolgte in Python. Für das Parsen der unstrukturierten Serverlogs nutzte ich reguläre Ausdrücke in Kombination mit speicheroptimierten Datenklassen (__slots__), um auch bei hohem Log-Aufkommen eine hohe Performanz zu garantieren [8], [9]. Zur Persistierung und effizienten Abfrage der strukturierten Daten wählte ich die eingebettete SQL-Datenbank SQLite, die sich durch ihre Ressourcenfreundlichkeit für den Betrieb auf industrieller Edge-Hardware eignet [10], [11].

Ein wesentlicher Schwerpunkt lag auf der Detektion von Memory Leaks. Da ein isolierter Anstieg des Arbeitsspeichers auch durch legitime Lastspitzen begründet sein kann, entwickelte ich einen Algorithmus, der den Trend der Speicherauslastung mittels linearer Regression mit der Frequenz der Server-Events korreliert. Steigt der Speicherbedarf signifikant, während die Event-Rate konstant bleibt oder sinkt, klassifiziert das System dies als potenzielle Anomalie. Die Ergebnisse werden über ein Webinterface via WebSocket in Echtzeit für den Anwender aufbereitet [12]. Die Evaluation in einer isolierten virtuellen Maschine bestätigte, dass die definierten Heuristiken ineffiziente Session-Erstellungen, exzessives Read Polling und künstlich induzierte Memory Leaks detektieren.

4.Fazit

Die vorliegende Arbeit liefert ein erweiterbares Framework zur Diagnostik von OPC UA-Netzwerken. Durch den Verzicht auf serverseitige Modifikationen ist die Anwendung portabel. Sie bietet eine fundierte Basis für künftige Forschungen, etwa der Integration von Machine-Learning-Modellen zur Mustererkennung oder der Analyse von IT-Sicherheitsvorfällen im Kontext von Denial-of-Service-Angriffen [13].

Glossar

OPC UA (Open Platform Communications Unified Architecture): Ein offener, plattformunabhängiger Standard für die industrielle Kommunikation, der den sicheren Datenaustausch in der Industrie 4.0 ermöglicht [2].
Session: Eine logische, authentifizierte Verbindung zwischen einem OPC UA Client und einem Server, über welche Services abgewickelt werden [5].
Read Polling: Ein ineffizientes Verhaltensmuster, bei dem ein Client kontinuierlich Leseoperationen ausführt, anstatt den dafür vorgesehenen Subscription-Mechanismus zu nutzen.
RSS (Resident Set Size): Der Teil des Arbeitsspeichers eines Prozesses, der im physischen RAM resident ist und nicht ausgelagert wurde [3].
SQLite: Eine leichtgewichtige, serverlose SQL-Datenbank-Engine, ideal für ressourcenbeschränkte Umgebungen [10].
WebSocket: Ein Kommunikationsprotokoll für bidirektionale Echtzeit-Datenübertragung, hier genutzt für das Live- Update des Webinterfaces [12].

Bibliografie

[1] M. Hermann, T. Pentek, und B. Otto, „Design Principles for Industrie 4.0 Scenarios“, in 2016 49th Hawaii International Conference on System Sciences (HICSS), 2016, S. 3928–3937. doi: 10.1109/HICSS.2016.488.
[2] J. Lange, F. Iwanitz, und T. Burke, OPC: von Data Access bis Unified Architecture. VDE-Verlag, 2010.
[3] C. Siebenmann, „Understanding Resident Set Size and the RSS problem on modern Unixes“. Zugegriffen: 9. Dezember 2024. [Online]. Verfügbar unter: https://utcc.utoronto.ca/~cks/space/blog/unix/UnderstandingRSS
[4] H. F. Nielsen und others, „Network performance effects of HTTP/1.1, CSS1, and PNG“, in Proceedings of the ACM SIGCOMM '97 Conference, 1997, S. 155–166. doi: 10.1145/263105.263157.
[5] OPC Foundation, „UA Part 4: Services - 5.7 Session Service Set“, technical report, 2017. [Online]. Verfügbar unter: https://reference.opcfoundation.org/Core/Part4/v105/docs/5.7
[6] J. Corbet, „Another OOM killer rewrite“. Zugegriffen: 6. Dezember 2024. [Online]. Verfügbar unter: https://lwn.net/Articles/391222/
[7] K. Billimoria, Linux Kernel Debugging. Packt Publishing, 2022.
[8] C. Cardea und W. Hawkins, „Using Slots“. Zugegriffen: 27. Februar 2025. [Online]. Verfügbar unter: https://wiki.python.org/moin/UsingSlots
[9] Python Software Foundation, „Python Data model (__slots__)“. [Online]. Verfügbar unter: https://docs.python.org/3.12/reference/datamodel.html#slots
[10] R. Hipp, D. Kennedy, und J. Mistachkin, „About SQLite“. Zugegriffen: 16. Dezember 2024. [Online]. Verfügbar unter: https://sqlite.org/about.html
[11] R. Hipp, D. Kennedy, und J. Mistachkin, „SQLite Is Serverless“. [Online]. Verfügbar unter: https://sqlite.org/serverless.html
[12] mozilla.org contributors, „The WebSocket API (WebSockets)“. Zugegriffen: 23. Januar 2025. [Online]. Verfügbar unter: https://developer.mozilla.org/de/docs/Web/API/WebSockets_API
[13] M. Dahlmanns und others, „Easing the Conscience with OPC UA: An Internet-Wide Study on Insecure Deployments“, in Proceedings of the ACM Internet Measurement Conference (IMC '20), 2020, S. 101–110. doi: 10.1145/3419394.3423666.