Bei diesem Beitrag handelt es sich um einen wissenschaftlich begutachteten und freigegebenen Fachaufsatz („reviewed paper“).

doi.org/10.37544/1436–4980–2023–05–11

Softwarebasierte Sicherheitssteuerungen

Fehlertolerante Sicherheitssteuerung aus der Cloud

M. Fischer, M. Walker, A. Lechler, O. Riedel, A. Verl

Cloud-Systeme werden zunehmend in der Produktionstechnik eingesetzt. Sicherheitskritische Systeme lassen sich mit dem heutigen Stand der Technik nur schwer integrieren. Zufällige Fehler wie Alterungsprozesse oder Umwelteinflüsse müssen behandelt werden. In diesem Beitrag werden Architekturen vorgestellt, die eine performante Sicherheitssteuerung in Cloud-Systemen erlauben. Darüber hinaus wird die Fehlertoleranz von Hardware- und Softwareausfällen durch Redundanz analysiert und bewertet.

STICHWÖRTER

Sicherheit, Steuerungen, Industrie 4.0

Cloud-based fault tolerance of safety control in the cloud

Cloud systems are increasingly being used in production technology. However, based on current state of the art, safety-critical systems are difficult to integrate. It is absolutely necessary to handle random errors such as ageing processes or environmental influences. This paper presents architectures enabling high-performance safety control in cloud systems. It also analyzes and evaluates the fault tolerance of hardware and software failures due to redundancy .

1 Einleitung

Die Virtualisierung von Hardware- und Softwareressourcen nimmt in der klassischen IT seit vielen Jahren eine zentrale Rolle ein, zum Beispiel um vorhandene Ressourcen besser auszunutzen, Hardware einzusparen oder durch Kapselung eine höhere Sicherheit zu erreichen [1]. Grundsätzlich kann Virtualisierung als Abstraktion von Ressourcen bei erzwungener Modularität verstanden werden [2]. Erzwungene Modularität heißt, dass Klienten der Abstraktion diese nicht umgehen können. Beim Einsatz von virtuellen Maschinen abstrahiert der sogenannte Hypervisor die Hardware eines Servers oder PCs, sodass mehrere Betriebssysteme parallel betrieben werden können. Bei Containern liegt die Abstraktion auf Betriebssystemebene und isoliert Anwendungen hinsichtlich der Ressourcennutzung voneinander. Ein typischer Aufbau eines Rechenzentrums für Cloud Computing besteht aus mehreren Servern, um hochverfügbare Anwendungen zu ermöglichen. Dieser Aufbau wird typischerweise als Cluster bezeichnet.

Auch in der Operational Technology (OT) der Produktionstechnik wird Virtualisierung zunehmend eingesetzt, um neben der Verfügbarmachung der oben genannten Vorteile heterogene Hardwareumgebungen aufzulösen. Dabei sind die Anforderungen der Produktionstechnik wie Echtzeit und Zuverlässigkeit zu berücksichtigen.

Erweiterte Anforderungen haben sicherheitskritische Systeme wie Sicherheitssteuerungen im Sinne der Gefahrenfreiheit. Hier ist eine Vermeidung oder Detektion und Behandlung zufälliger Fehler zum Beispiel durch Alterungsprozesse oder Umwelteinflüsse erforderlich. Typische Fehler sind Hardwarefehler, die aus Umwelteinflüssen wie Temperaturschwankungen, Vibrationen und Strahlung resultieren und sich als sporadische Bitflips in RAM (Random Access Memory) und Prozessor oder auch als permanent unterbrochene Leitungen manifestieren.

Im operativen Einsatz gibt es aktuell zwei Ansätze, um Hardwarefehlern zu begegnen. Zum einen erlaubt eine mehrkanalige Hardware und ein Vergleich der Ergebnisse der Kanäle eine Diagnose von Hardwarefehlern – in der Regel erfolgt die Auslegung zweifach. Dabei wird spezielle Hardware verwendet, die beispielsweise mit zwei CPUs (Central Processing Units) ausgestattet ist, die im Lockstep-Verfahren arbeiten. Zum anderen wird codierte Verarbeitung eingesetzt, welche durch die Erzeugung verschiedener Arten von Daten- und Verarbeitungsredundanz eine Diagnose von Hardwarefehlern ermöglicht. Ersterer Ansatz hat den Nachteil, dass hohe Kosten und hoher Aufwand etwa für die Entwicklung einer zweikanaligen Hardware notwendig sind und ein Einsatz auf Standardhardware nicht möglich ist. Der zweite Ansatz ist hardwareunabhängig und kann auf Standardhardware eingesetzt werden, jedoch mit hohen Performanceeinbußen [3].

Im Kontext der Virtualisierung gewinnt die redundante Ausführung von Sicherheitssteuerungen auf unterschiedlicher Hardware an Bedeutung, da typischerweise Cluster betrieben werden. Im Gegensatz zur mehrkanaligen Hardware werden mehrere PCs oder Server verwendet, die jeweils eine Replika der Sicherheitssteuerung ausführen. Dabei ist der administrative Aufwand für den Betrieb mehrerer Sicherheitssteuerungen aufgrund der Virtualisierung in Kombination mit Werkzeugen zur Verwaltung der virtuellen Systeme gering.

In diesem Beitrag werden daher Architekturen vorgestellt, die unter Ausnutzung eines Clusters den Einsatz von leistungsfähigen Sicherheitssteuerungen auf Standardhardware ohne codierte Ver-

https://doi.org/10.37544/1436-4980-2023-05-11, am 24.11.2024, 19:16:52
Open Access – (cc) BY – https://elibrary.vdi-verlag.de/agb

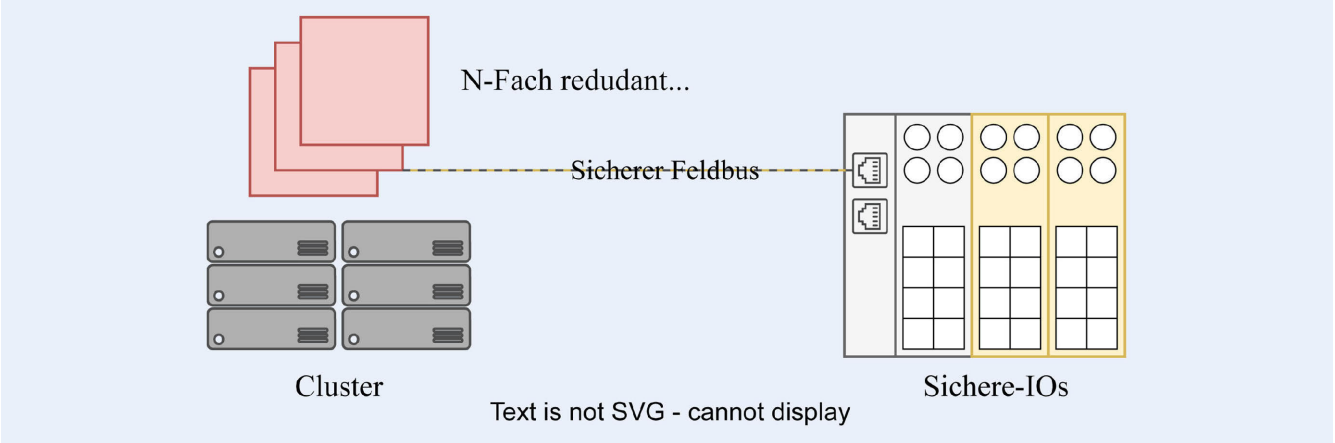

Bild 1. Architekturübersicht für redundante Sicherheitssteuerungen. *Grafik: ISW, Uni Stuttgart*

arbeitung ermöglichen. Des Weiteren wird neben der Detektion von Hardwarefehlern auch die Fehlertoleranz von Hardware- und Softwareausfällen durch Redundanz betrachtet. Ein fehlertolerantes System in dieser Arbeit muss Toleranz gegenüber Hardwarefehlern und Toleranz gegenüber Serverausfällen besitzen.

2 Verwandte Arbeiten

In der Literatur gibt es Arbeiten wie [4, 5], die durch Virtualisierung sowohl sicherheitskritische als auch normalkritische Systeme auf einer Hardware betreiben. Dabei werden FPGAs (Field Programmable Gate Arrays) eingesetzt und es wird Diversität auf Hardwareebene durch unterschiedliche CPUs erzeugt. Hier steht die effiziente Ausnutzung der Hardware im Vordergrund.

Zum anderen gibt es Arbeiten, die Multicore-CPUs einsetzen und Redundanz auf mehreren CPU-Kernen erzeugen [6]. Allerdings werden viele Hardwarekomponenten von den CPU-Kernen gemeinsam genutzt, sodass der Nachweis von Fehlern durch gemeinsame Ursachen der Softwarekanäle schwierig ist. Auch das Zusammenführen der Ergebnisse der verschiedenen Softwarekanäle ist eine Herausforderung, welche etwa durch die codierte Verarbeitung eines Mehrheitsentscheiders in [7] gelöst wird.

3 Architektur und Anforderungen

Die Probleme der verwandten Arbeiten bei Fehlern aufgrund gemeinsamer Ursachen und spezieller Hardware können durch den Einsatz eines Clusters mit Standardhardware umgangen werden. Allerdings müssen die Zeitanforderungen und die Zusammenführung redundanter Ergebnisse berücksichtigt werden.

3.1 Annahmen zur Architektur

Ein Cluster besteht in dieser Arbeit aus mehreren Servern. Als Virtualisierung können sowohl ein Hypervisor mit virtuellen Maschinen als auch Container eingesetzt werden. Echtzeit kann beim Einsatz eines Hypervisors durch statische Partitionierung der CPU und einem echtzeitfähigen Betriebssystem oder durch hierarchisches Scheduling erreicht werden [8–11]. Bei Containern kann Echtzeit durch Echtzeit-Linux realisiert werden [12]. Im Cluster werden N Replikas der Sicherheitssteuerungen ausgeführt, wie in **Bild 1** dargestellt.

Die Kommunikation zu den sicheren E/As (Ein- und Ausgabe-Einheiten) erfolgt über einen sicheren Feldbus nach dem Black-Channel-Prinzip, also der Übertragung sicherheitskritischer Signale über nicht sichere Kommunikationsmedien. Entsprechende Lösungen gibt es für verschiedene Kommunikationstechniken, wie etwa WiFi-Netzwerke [13]. Die Absicherungsmechanismen und Funktionsweisen werden nach DIN EN 61784-3 angenommen, sodass Paketverlust, Zeitverletzungen, Paketduplizierungen und ähnliche Fehler durch das Safety-Protokoll entdeckt werden. Es wird angenommen, dass die N Replikas garantiert auf N verschiedenen Servern ausgeführt werden, sodass die Detektion von zufälligen Hardwarefehlern möglich ist. Hierfür wird ein geeigneter und sicherer Mechanismus vorausgesetzt, der zum Beispiel auf statischen und eindeutigen Identifikationsnummern basiert.

3.2 Anforderungen

In den Annahmen wird gezeigt, dass die Echtzeitfähigkeit für die Virtualisierung als gegeben angenommen werden kann. Dies bedeutet, dass die einzelnen Replikas der Sicherheitssteuerung in Echtzeit ausgeführt werden können. Die Zusammenführung der Ergebnisse kann zusätzliche Kommunikationspfade erfordern und wird im nächsten Kapitel gezeigt. Daher muss ein Nachweis über die maximale Kommunikationslatenz erbracht werden. Die Zusammenführung der Ergebnisse muss Hardwarefehler detektieren oder korrigieren können.

Die Fehlertoleranz erfordert die sichere Fortführung des Betriebs bei Ausfall von Hardware oder Software. Dazu muss bei Ausfall einer oder mehrerer Repliken sichergestellt sein, dass die verbleibenden Replika weiterhin Hardwarefehler detektieren oder korrigieren können. Darüber hinaus muss die Detektion eines Ausfalls in Echtzeit erfolgen.

4 Hardwarefehlerdetektion und Fehlertoleranz

Um Fehlertoleranz zu implementieren, müssen sowohl die Hardwarefehlerdetektion als auch die Toleranz von Ausfällen berücksichtigt werden. In der Literatur zu sicherheitskritischen Systemen und fehlertoleranten Cloud-Systemen können drei relevante Architekturen identifiziert werden. Im Folgenden werden

https://doi.org/10.37544/1436-4980-2023-05-11, am 24.11.2024, 19:16:52
Open Access – (cc) BY – https://elibrary.vdi-verlag.de/agb

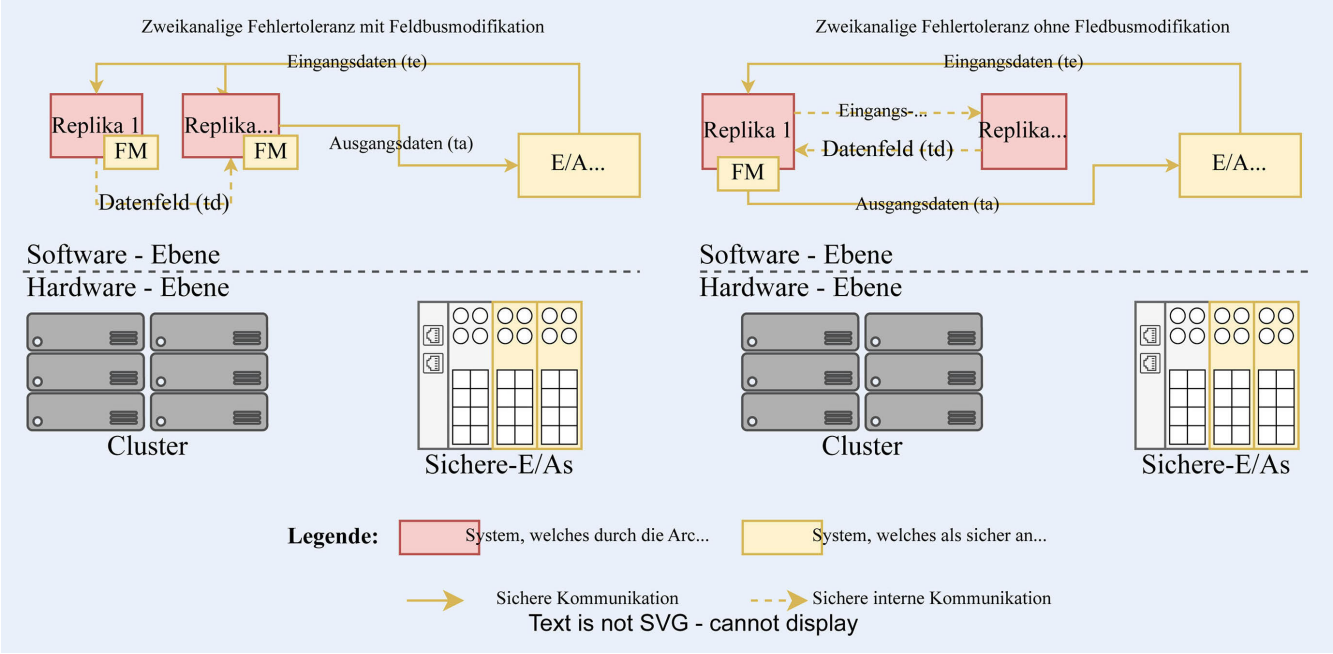

Bild 2. Umsetzungsmöglichkeiten für zweikanalige Fehlertoleranz. *Grafik: ISW, Uni Stuttgart*

diese drei Architekturen vorgestellt und gezeigt, wie sie für die vorliegende Zielsetzung eingesetzt werden können.

4.1 Hardwarefehlerdetektion bei zwei-kanaliger codierter Verarbeitung

Früchtl [14] stellt in seiner Arbeit die Zusammenführung zweier Softwarekanäle über das Feldbustelegramm vor, die durch codierte Verarbeitung gesichert sind. Dabei schreibt der erste Kanal das Datenfeld des Telegramms und der zweite Kanal die Prüfsumme des Telegramms. Die Überprüfung erfolgt im sicheren Kommunikationspartner, der für jedes Paket die Konsistenz von Datenfeld und Prüfsumme überprüft. Bei Inkonsistenzen liegt ein zufälliger Hardwarefehler vor und das Sicherheitssystem geht automatisch in den sicheren Zustand über. Der Vorteil dieses Verfahrens ist, dass es mit jedem existierenden Feldbus realisiert werden kann. Eine erhöhte Ausfallsicherheit ist nicht gegeben, da keine weiteren Kanäle in das Konzept integriert werden können.

In einem Cloud-Cluster kann dieses Prinzip genutzt werden, indem eine Replika das Datenfeld und die andere Replika die Prüfsumme in das Telegramm schreibt. Die erste Replika kann das Paket an die zweite Replika weiterleiten, welche das Paket dann an die Ein- und Ausgabe-Einheit (E/A) sendet, wie in **Bild 2** dargestellt.

Dies erfordert eine Modifikation des Feldbusses, da mehrere Feldbusmaster (FM) nicht möglich sind. Alternativ kann die zweite Replika das Telegramm an die erste Replika zurücksenden, die dann das Telegramm an die E/A sendet. In diesem Fall ist nur eine Modifikation im Feldbusmaster der ersten Replika nötig.

Die Latenz setzt sich dabei aus der Verarbeitung $(t_{v,N})$ je Replika und der Kommunikation zusammen. Für die zweikanalige Fehlertoleranz mit modifiziertem Feldbus ergibt sich eine Gesamtlatenz von $t_{G,mod} = t_e + \max(t_{v,1}+t_d;t_{v,2}) + t_a$. Im Falle eines unmodifizierten Feldbusses ergibt sich eine Gesamtlatenz von $t_{G,un} = t_e + \max(t_{v,1};t_e + t_{v,2} + t_d) + t_a$. Der Ausfall einer Replika kann nicht toleriert werden, da beide für die Detektion von Hardwarefehlern benötigt werden.

4.2 N-Modulare Systeme mit Mehrheitsentscheider

N-modulare Redundanz ist ein Ansatz zur Implementierung fehlertoleranter Systeme. Im Fehlerfall, zum Beispiel bei einem Hardwarefehler, kann die Funktionalität aufrechterhalten werden. Typischerweise wird zwischen passiven und aktiven Systemen unterschieden [15]. Bei passiven Systemen maskiert ein Mehrheitsentscheider (ME) fehlerhafte oder fehlende Ergebnisse unter der Annahme, dass die mehrheitlich gleichen Ergebnisse korrekt sind. Bei aktiven Systemen können zusätzlich fehlerhafte Module abgeschaltet oder ausgetauscht werden. Das Prinzip der n-modularen Redundanz kann auf zwei Arten angewendet werden, wie in **Bild 3** dargestellt.

Entweder befindet sich der Mehrheitsentscheider in den sicheren E/As, was eine Anpassung des Feldbusses erfordert. Dabei eignet sich vor allem eine Kommunikation nach dem Publisher/Subscribe-Prinzip, bei welchem der klassische FM aufgelöst wird. Während der Konfigurationszeit wird die Anzahl der Replikas bekannt gemacht. Alternativ kann der Mehrheitsentscheider im Feldbusmaster liegen. In diesem Fall muss dieser einzelne Fehlerpunkt gegen zufällige Hardwarefehler abgesichert werden, etwa durch codierte Verarbeitung. Im ersten Fall ist eine Fehlertoleranz gegen den Ausfall einzelner Replikas gegeben. Im zweiten Fall kann der Ausfall des Feldbusmasters nicht toleriert werden.

Die Latenz setzt sich dabei aus der Verarbeitung $(t_{v,N})$ und der Kommunikation zusammen. Für den Mehrheitsentscheider im E/A ergibt sich eine Gesamtlatenz von $t_{G,E/A} = t_e + \max(t_{v1}; \ldots ;t_{v,N}) + t_a + t_{ME}$. Für einen Mehrheitsentscheider im Cluster ergibt sich die Gesamtlatenz von $t_{G,Cluster} = t_e + \max(t_{v1}; \ldots; t_{v,N}) + t_i + t_{ME} + ta$.

https://doi.org/10.37544/1436-4980-2023-05-11, am 24.11.2024, 19:16:52
Open Access – (cc) BY – https://elibrary.vdi-verlag.de/agb

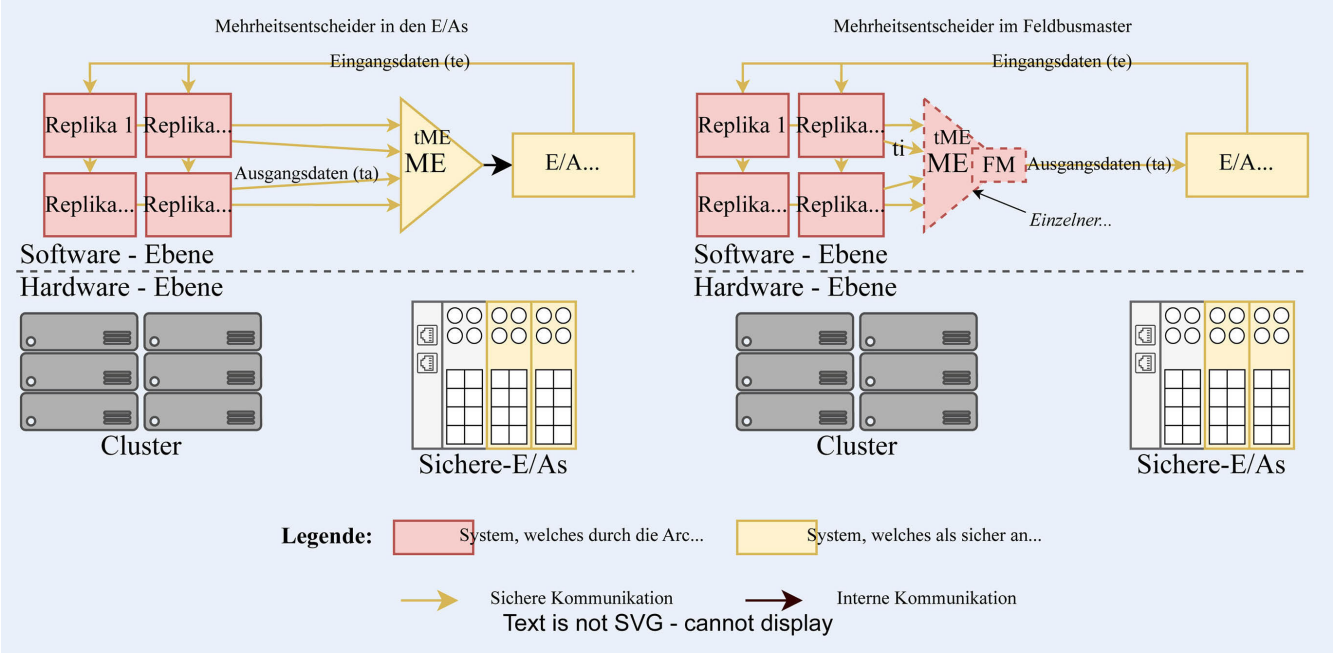

Bild 3. Umsetzungsmöglichkeiten für N-Modulare Systeme. *Grafik: ISW, Uni Stuttgart*

4.3 Fehlertolerante Cloud-Systeme

Kritische Systeme werden fehlertolerant betrieben, um eine hohe Verfügbarkeit und Integrität zu erreichen. In der Literatur [16] gibt es verschiedene Ansätze wie Replikation, Migration oder selbstheilende Systeme. Am häufigsten kommen Ansätze mit replizierter Hardware zum Einsatz, wobei die Software entweder parallel betrieben wird oder im Fehlerfall migriert wird [16]. Um einen einzelnen Ausfallpunkt zu vermeiden, überwachen sich die redundanten Replikas gegenseitig. Dieser Ansatz wird zum Beispiel bei hochverfügbaren Kubernetes-Clustern verwendet [17].

Für die Anwendung des Prinzips in sicherheitskritischen Systemen muss zum einen die Überwachung und die Behandlung eines Fehlerfalls zeitdeterministisch erfolgen. Zum anderen müssen Hardwarefehler detektiert werden. Jedes Replika empfängt die Eingangsdaten und sendet das berechnete Ergebnis an alle anderen Replikas. Die Anzahl an Kommunikationsverbindungen ist daher höher. Der Ausfall einer Replika wird über das Ausbleiben des Ergebnisses erkannt. Dazu ist ein Timer erforderlich. Eine Replika wird als aktiv bestimmt und sendet die Ausgangsdaten zur E/A-Einheit. Fällt die aktive Replika aus, übernimmt ein anderes Replika die aktive Rolle. Um eine Auswahl zur Laufzeit zu vermeiden, wird eine statische Reihenfolge festgelegt. Die maximal tolerierte Anzahl von Ausfällen ist *N-2*.

Zur Detektion von Hardwarefehlern kann entweder das in Kapitel 4.1 oder in Kapitel 4.2 beschriebene Prinzip verwendet werden. **Bild 4** zeigt diese Architektur.

Bei Einsatz der Methodik aus Kapitel 4.1 wird das Telegramm der Ausgangsdaten in der aktiven Instanz unter Verwendung des Ergebnisses der eigenen Berechnung und der einer anderen Replika gebildet. Wird die Methodik aus Kapitel 4.2 genutzt, verwendet das aktive Replika einen Mehrheitsentscheider. Dieser ist ein einzelner Fehlerpunkt und muss zusätzlich abgesichert werden.

Die Latenz setzt sich dabei aus der Verarbeitung (t_v) und Kommunikation zusammen. Beim Zusammensetzen des Telegramms ergibt sich eine Gesamtlatenz von $t_{G,tel} = t_e + \max(t_{v1}; \ldots t_{v,N}) + t_i + t_a$ und bei der Verwendung eines Mehrheitsentscheiders ergibt sich eine Gesamtlatenz von $t_{G,ME} = t_e + \max(t_{v1} + t_{ME}; \ldots t_{v,N} + t_{ME}) + t_i + t_a$.

5 Vergleich der Architekturen

Die **Tabelle** zeigt die drei möglichen Architekturen und die jeweiligen Ausprägungen mit ihren Eigenschaften.

Der Vergleich der Latenz gibt einen Hinweis darauf, welche Elemente zur Gesamtlatenz beitragen. Je mehr serielle Elemente, desto größer die potenzielle Latenz. Für eine konkrete Einordnung ist ein experimenteller Vergleich notwendig. Zur Detektion von Hardwarefehlen werden immer zwei Replikas benötigt, weshalb die Ausfalltoleranz immer *N-2* beträgt. Einige Architekturen können nicht auf bestehende sichere Feldbusse zurückgreifen, sondern benötigen angepasste Kommunikationsmuster. Für den Einsatz in bestehenden Umgebungen ist daher die Unterstützung aktueller sicherer Feldbusprotokolle relevant. Die Anzahl der Kommunikationsverbindungen ist für die Netzwerkauslastung relevant. Je geringer die Anzahl der Kommunikationsverbindungen, desto effizienter kann das Netzwerk ausgelastet werden. Bei einigen Architekturen ist die Absicherung eines einzelnen Fehlerpunktes erforderlich.

6 Zusammenfassung und Ausblick

Der Einsatz von Standardhardware für sicherheitskritische Systeme ist aufgrund der geringeren Kosten attraktiv. Die Ansätze in der Literatur setzen auf codierte Verarbeitung. Durch den Aufbau von Cloud-Systemen in Form von Server-Clustern ist der Einsatz für sicherheitskritische Systeme durch Redundanz möglich. Hierbei können nicht nur Hardwarefehler, sondern auch Systemausfälle toleriert werden. In dieser Arbeit wurden daher Architekturen aus der Literatur identifiziert, die diese Toleranz ermöglichen. Die Architekturen können in unterschiedlichen

https://doi.org/10.37544/1436-4980-2023-05-11, am 24.11.2024, 19:16:52
Open Access – (cc) BY – https://elibrary.vdi-verlag.de/agb

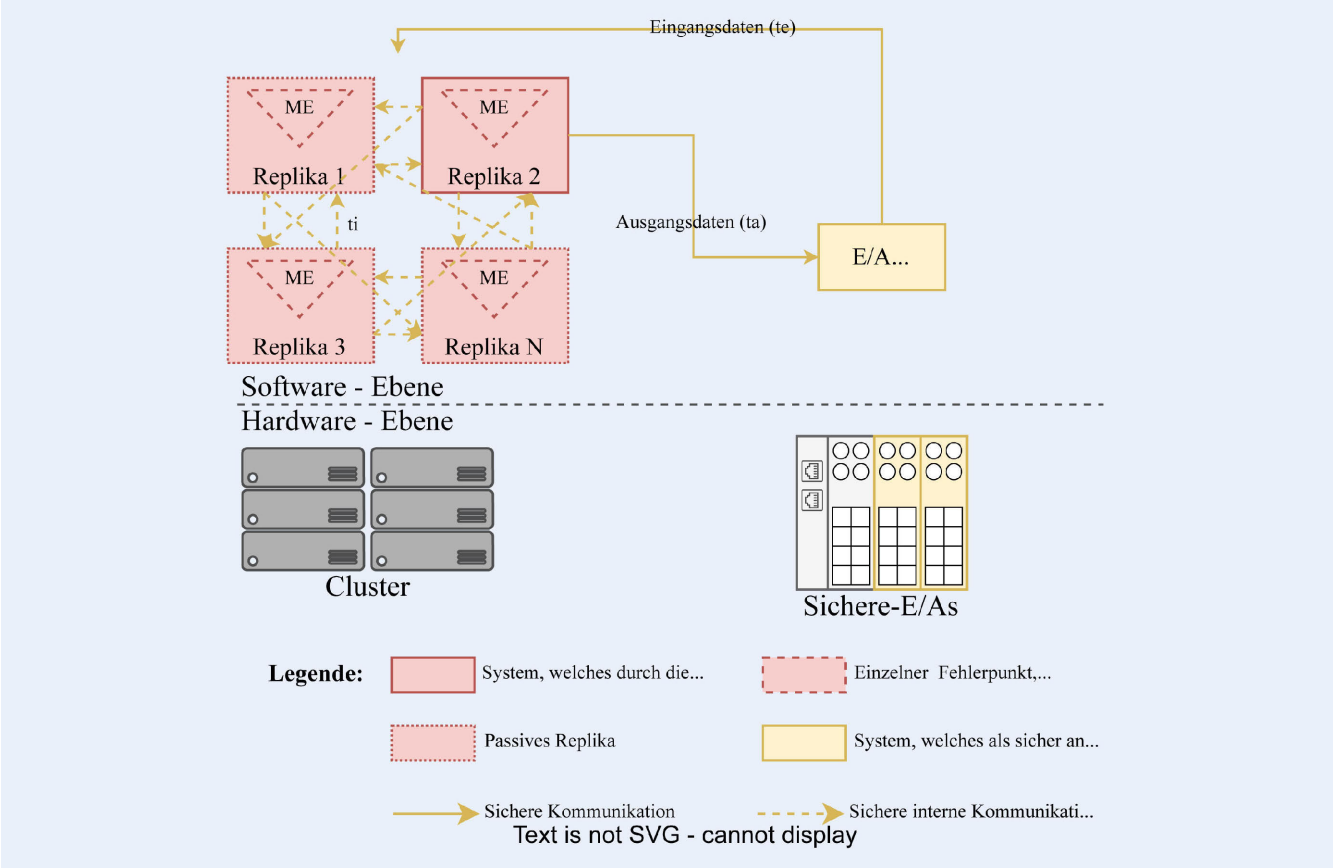

Bild 4. Umsetzungsmöglichkeit nach dem Prinzip fehlertoleranter Cloud-Systeme. *Grafik: ISW, Uni Stuttgart*

Tabelle. Vergleich der fehlertoleranten Architekturen.

Eigenschaften Architektur	Latenz	Ausfalltoleranz	Feldbusmodifikation notwendig	Anzahl der logischen Kommunikationsverbindungen	Absicherung eines einzelnen Fehlerpunkts notwendig
Zweikanalig (modifiziert)	$t = t_e + \max(t_{v,1} + t_d; t_{v,2}) + t_{aGmod}$	$N-2=0$	Ja	4	Nein
Zweikanalig (unmodifiziert)	$t_{G,un} = t_e + \max(t_{v,1}; t_e + t_{v,2} + t_d) + t_a$	$N-2=0$	Nein	4	Nein
Mehrheitsentscheider (E/A)	$t_{G,E/A} = t_e + \max(t_{v1}; \ldots; t_{v,N}) + t_a + t_{ME}$	$N-2$	Ja	$2N$	Nein
Mehrheitsentscheider (Cluster)	$t_{G,Cluster} = t_e + \max(t_{v1}; \ldots; t_{v,N}) + t_i + t_{ME} + t_a$	$N-2$ 0 unter Berücksichtigung des Mehrheitsentscheiders	Nein	$2N+1$	Ja
Fehlertolerantes Cloudsystem (Telegramm)	$t_{G,tel} = t_e + \max(t_{v1} + t_{ME}; \ldots; t_{v,N} + t_{ME}) + t_i + t_a$	$N-2$	Ja	$N+1+N(N-1)=N^2+1$	Nein
Fehlertolerantes Cloudsystem (ME)	$t_{G,ME} = t_e + \max(t_{v1} + t_{ME}; \ldots; t_{v,N} + t_{ME}) + t_i + t_a$	$N-2$	Ja	$N+1+N(N-1)=N^2+1$	Ja

https://doi.org/10.37544/1436-4980-2023-05-11, am 24.11.2024, 19:16:52
Open Access – (cc) BY – https://elibrary.vdi-verlag.de/agb

Ausprägungen realisiert werden. Ein Vergleich zeigt, dass neben der Ausfalltoleranz auch andere Eigenschaften wie die Latenz, die Anzahl der Kommunikationsverbindungen, die Absicherung einzelner Fehlerpunkte oder die Möglichkeit der Nutzung vorhandener sicherer Feldbusse berücksichtigt werden müssen.

In weiteren Arbeiten ist aufzuzeigen, welche Ausprägung am besten geeignet ist. Dazu müssen vor allem die Grundlagen für eine sichere Kommunikation in verteilten Systemen gelegt werden, um darauf aufbauend die Architekturmöglichkeiten weiter zu analysieren.

Literatur

[1] Anand, A.; Chaudhary, A.; Arvindhan, M.: The Need for Virtualization: When and Why Virtualization Took Over Physical Servers. In: Hura, G. S.; Singh, A. K.; Siong Hoe, L. (eds.): Advances in Communication and Computational Technology. Singapore: Springer Singapore 2021, pp. 351–1359

[2] Baun, C.: Virtualisierung. In: Baun, C. (Hrsg.): Betriebssysteme kompakt. Berlin: Springer Vieweg 2017, S. 231–241

[3] Fischer, M.; Riedel, O.; Lechler, A.: Comprehensive Analysis of Software-Based Fault Tolerance with Arithmetic Coding for Performant Encoding of Integer Calculations. In: Trapp, M.; Saglietti, F.; Spisländer, M. et al. (Hrsg.): Computer Safety, Reliability, and Security. 41st International conference SAFECOMP. [S.l.]. Cham: Springer 2022, pp. 144–157

[4] Perez, J.; Gonzalez, D.; Trujillo, S. et al.: A safety concept for a wind power mixed-criticality embedded system based on multicore partitioning. Stand: 2014. Internet: www.uni-siegen.de/dreams/publications/presentation/a_safety_concept_for_a_wind_power_mixed-criticality_embedded_system_based_on_multicore_partitioning-paper.pdf. Zugriff am 28.03.2023

[5] Agirre, I.; Azkarate-Askasua, M.; Larrucea, A. et al.: A Safety Concept for a Railway Mixed-Criticality Embedded System Based on Multicore Partitioning. 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM), liverpool, United Kingdom, 2015, pp. 1780–1787

[6] Cerrolaza, J. P.; Obermaisser, R.; Abella, J. et al.: Multi-core Devices for Safety-critical Systems. ACM Computing Surveys 53 (2021) 4, pp. 1–38

[7] Ulbrich, P.; Hoffmann, M.; Kapitza, R. et al.: Eliminating Single Points of Failure in Software-Based Redundancy. 2012 Ninth European Dependable Computing Conference (EDCC), Sibiu, 2012, pp. 49–60

[8] Li, H.; Xu, X.; Ren, J. et al.: ACRN: a big little hypervisor for IoT development. 15th ACM SIGPLAN/SIGOPS International Conference, Providence, RI, USA, 2019, pp. 31–44

[9] Abeni, L.; Faggioli, D.: Using Xen and KVM as real-time hypervisors. Journal of Systems Architecture 106 (2020), # 101709

[10] Intel (ed.): Achieving Real-Time Performance on a Virtualized Industrial Control Platform. White Paper. Stand: 2014. Internet: cdrdv2-public.intel.com/330740/industrial-solutions-real-time-performance-white-paper.pdf. Zugriff am 28.03.2023

[11] Abeni, L.; Biondi, A.; Bini, E.: Hierarchical scheduling of real-time tasks over Linux-based virtual machines. Journal of Systems and Software 149 (2019), pp. 234–249

[12] Struhár, V.; Behnam, M.; Ashjaei, M. et al.: Real-Time Containers: A Survey. Stand: 2020. Internet: drops.dagstuhl.de/opus/volltexte/2020/12001/pdf/OASIcs-Fog-IoT-2020–7.pdf. Zugriff am 28.03.2023

[13] Peserico, G.; Morato, A.; Tramarin, F. et al.: Functional Safety Networks and Protocols in the Industrial Internet of Things Era. Sensors 21 (2021) 18, doi.org/10.3390/s21186073

[14] Früchtl, M.: Sicherheit eingebetteter Systeme auf Basis arithmetischer Codierungen. Dissertation, Universität Kassel, 2014

[15] Dubrova, E.: Fault-Tolerant Design. New York: Springer 2013

[16] Hasan, M.; Goraya, M. S.: Fault tolerance in cloud computing environment: A systematic survey. Computers in Industry 99 (2018), pp. 156–172

[17] The Linux Foundation: Options for Highly Available Topology. Stand: 17.01.2022. Internet: kubernetes.io/docs/setup/production-environment/tools/kubeadm/ha-topology/. Zugriff am 28.03.2023

Marc Fischer, M.Sc.

Moritz Walker, M.Sc.

Dr.-Ing. **Armin Lechler**

Prof. Dr.-Ing. **Oliver Riedel**

Prof. Dr.-Ing. **Alexander Verl**

Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen (ISW)
Universität Stuttgart
Seidenstr. 36, 70174 Stuttgart
Tel. +49 711 / 685 82534
marc.fischer@isw.uni-stuttgart.de
www.isw.uni-stuttgart.de

LIZENZ

Dieser Fachaufsatz steht unter der Lizenz Creative Commons Namensnennung 4.0 International (CC BY 4.0)

https://doi.org/10.37544/1436-4980-2023-05-11, am 24.11.2024, 19:16:52
Open Access – (cc) BY – https://elibrary.vdi-verlag.de/agb