Fehlertolerante Systeme durch effektive Laufzeitdiagnose mit SFL

Philipp Dürnay*, **Hans-Gerhard Groß,**

Fakultät Informationstechnik der Hochschule Esslingen - University of Applied Sciences

Wintersemester 2015/2016

Fehlertolerante Systeme haben die Fähigkeit, trotz einer begrenzten Anzahl fehlerhafter Subsysteme ihre spezifizierte Funktion weitestgehend zu erfüllen [1]. Damit wird die Zuverlässigkeit des Systems erhöht. Fehlertoleranz spielt somit eine wichtige Rolle für autonome und sicherheitskritische Systeme.

Spectrum-based Fault Localization (SFL) ist ein Verfahren zur Fehlerlokalisierung und eignet sich aufgrund seiner geringen Ressourcenanforderung für den Einsatz zur Laufzeit. Eine Herausforderung des Verfahrens bleibt jedoch der Overhead, der durch die Sammlung der Daten (**Monitoring**) entsteht. In diesem Artikel wird das Verfahren SFL im Rahmen von fehlertoleranten Systemen vorgestellt. Dazu wird zunächst der Begriff des Fehlers erläutert und eine Übersicht über fehlertolerante Systeme gegeben, bevor genauer auf das Verfahren SFL eingegangen wird. Im Anschluss wird ein Ansatz geschildert effizienteres Monitoring mithilfe von Publish/Subscribe-Systemen zu realisieren.

1 Grundlagen

Ein Fehler besteht nach Laprie [1] aus drei Aspekten:

- **Fehlerursache** (eng. fault)
  Die Fehlerursache ist der eigentliche Grund für den Fehler wie zum Beispiel eine falsch programmierte Anweisung.
- **Fehlzustand** (eng. error)
  Der Fehlzustand ist ein interner Zustand des Systems, der die Folge der Fehlerursache ist wie zum Beispiel ein falsch berechneter Zwischenwert.
- **Fehlerwirkung** (eng. failure)
  Die Fehlerwirkung ist die nach außen hin sichtbare Auswirkung des Fehlzustandes, z.B. eine fehlerhafte Ausgabe am Bildschirm oder der Ausfall des Systems.

Nicht jeder Fehlzustand resultiert zwangsläufig in einer direkt verbundenen Fehlerwirkung. Fehlzustände treten nur auf, wenn eine Fehlerursache ausgelöst wird und können selbst dann weit entfernt von der eigentlichen Ursache zum Tragen kommen. Im Falle von Mehrfachfehlern können diese sich gegenseitig beeinflussen wodurch die Identifizierung der Fehlerursache zusätzlich erschwert wird.

Fehlertolerante Systeme können Fehler erkennen und entsprechende Gegenmaßnahmen ergreifen. Fehler zu erkennen und eine detaillierte Diagnose über Ursache und potentielle Auswirkungen zu erstellen, ist Aufgabe der Fehlerdiagnose, welche im nächsten Abschnitt näher erläutert wird. Zunächst sollen jedoch die Möglichkeiten der Fehlerbehandlung geschildert werden.

Die Verfahren der Fehlerbehandlung können in drei Klassen eingeteilt werden [2]:

- **Fehlerbehebung**
  Eine Fehlerbehebung entfernt den ursprünglichen Fehlzustand beispielsweise durch das Neustarten eines bestimmten Systemteils. Ein Beispiel hierfür ist das Recovery Scheme Block-Verfahren.
- **Rekonfiguration**
  Bei der Rekonfiguration wird die fehlerhafte Komponente vom weiteren Systemablauf ausgeschlossen. So können zum Beispiel auszuführende Aufgaben auf noch intakte Komponenten verteilt werden (sogenannte Graceful Degradation). Darüber hinaus gibt es die Möglichkeit, die fehlerhafte Komponente durch eine Ersatzkomponente (Standby-Komponente) zu ersetzen.
- **Kompensation**
  Bei der Fehlerkompensation wird der fehlerhafte Zustand nicht beseitigt. Stattdessen wird der Fehler von anderen Komponenten, einer höheren Ebene ausgeglichen. Ein klassiches Beispiel hierfür sind sogenannte N-aus-M Systeme mit Mehrheitsentscheid.

1.1 Fehlerdiagnose

Das Erstellen einer Fehlerdiagnose besteht aus zwei Schritten. Zunächst muss der Fehler erkannt werden, bevor seine Ursache weiter eingegrenzt werden kann.

Zur **Fehlererkennung** existieren verschiedene Möglichkeiten wie beispielsweise Replikationstests, Konsistenztests, Umkehrungstests, Prüfsummen oder Zeitüberprüfungen. Die Verfahren können in unterschiedlicher

*Diese Arbeit wurde durchgeführt bei der Firma IT Designers, Esslingen

Kombination und Ausprägung zum Einsatz kommen [3].

Für eine Beurteilung des Systemszustandes wird häufig ein Modell des Systems verwendet. Das Modell kann beispielswese auf Basis des zugrunde liegenden physikalischen Prozesses erstellt werden, in dessen Kontext das System betrieben wird. So kann erkannt werden, ob der Zustand eines Systems plausibel oder unplausibel ist und welcher Teil eines Systems den Fehler verursacht. Diese sogenannten **modellbasierten** Ansätze hängen jedoch stark vom verwendeten Modell ab. Das Modell muss zusätzlich zum System entwickelt werden und kann somit auch selbst fehlerhaft sein. Bei Änderungen am ursprünglichen System muss zudem auch das Modell angepasst werden. Nur ein detailliertes und möglichst fehlerfreies Modell eignet sich für eine präzise Fehlerdiagnose [4].

Der Systemzustand kann auch anhand einer längeren Beobachtung des Systems beurteilt werden. Auf Basis vergangener Zustandswerte kann der Plausibilitätsbereich für zukünftige Werte festgelegt werden. So kann ungewöhnliches Verhalten erkannt werden. In diesem Kontext werden unter anderem statistische Methoden, Trendanalysen und neuronale Netze verwendet. Diese Methoden arbeiten unabhängig von einem zugrunde liegenden spezifischen Systemmodell und können daher universeller entwickelt werden [4].

SFL bietet einen Ansatz innerhalb dieser sogenannten **beobachtungsbasierten** Fehlerdiagnose (engl. observation based). Hier wird die Beteiligung der Komponenten am Systembetrieb analysiert um dadurch fehlerhafte Komponenten zu lokalisieren. Wie das Verfahren funktioniert wird im Folgenden erläutert.

2 Spectrum-based Fault Localization

SFL analysiert die Beteiligung von Komponenten während der Systemausführung. Das Verfahren basiert auf der Annahme, dass eine Komponente die häufiger in Zusammenhang mit fehlerhaften Systemabläufen (**Transaktionen**) verwendet wird, wahrscheinlicher fehlerhaft ist, als Komponenten, die mit korrekten Ausführungen verwendet werden.

Der Anteil einzelner Komponenten an Transaktionen wird ausgewertet. Treten Fehler im System auf, kann eine Rangordnung erstellt werden, welche der Komponenten höchstwahrscheinlich den Fehler verursacht.

Für das Verfahren SFL werden folgende Parameter definiert:

- M Potentiell fehlerhafte Komponenten C
- N Transaktionen T
- N binäre Transaktionsergebnisse O
- Eine Aktivitätsmatrix A der Größe N x M

Für jede Transaktion kann ein Aktivitätsvektor gebildet werden, welche für jede Komponente die Information enthält, ob sie an der Transaktion beteiligt war oder nicht. Der letzte Vektor enthält die binäre Information, ob die jeweilige Transaktion erfolgreich war oder nicht. Aus den Aktivitätsvektoren entsteht eine Aktivitätsmatrix wie in folgender Abbildung dargestellt:

C	t_1	t_2	t_3	t_4	t_5	t_6
c_0	1	1	1	1	1	1
c_1	1	1	1	1	1	1
c_2	1	1	1	1	0	1
c_3	1	1	1	1	0	0
c_4	1	1	1	1	0	1
c_5	1	1	0	0	0	0
c_6	1	1	1	1	0	1
c_7	0	1	0	1	0	0
c_8	1	0	1	0	0	1
c_9	1	0	1	0	0	1
c_{10}	1	1	1	1	1	1
O	1	1	1	1	0	0

Abbildung 1: Beispielhafte Aktivitätsmatrix

Anhand der Aktivitätsmatrix kann mithilfe binärer Vergleichsverfahren die fehlerhafte Komponente ermittelt werden. Dabei wird O mit dem Beteiligungsvektor (Spektrum) der Komponente an den verschiedenen Transaktionen verglichen. Je größer die Übereinstimmung desto wahrscheinlicher handelt es sich um die fehlerhafte Komponente [5].

Der geringe Bedarf an Ressourcen macht SFL zu einem effektiven Verfahren für die Fehlerlokalisierung zur Laufzeit. Anhand von binären Beteiligungsvektoren kann eine Aussage getroffen welche Komponente möglicherweise den Fehler verursacht. Das Verfahren benötigt dafür kein Modell des Systems und kann auch Mehrfachfehler lokalisieren. Somit kann mit geringem Implementierungsaufwand eine sehr exakte Fehlerdiagnose integriert werden [5].

Eine Herausforderung des Verfahrens bleibt der Overhead, der durch die Beobachtung des Systems entsteht. Chen et al. [6] verwenden SFL für Services mithilfe des Frameworks Turmeric. Das verwendete online [1] Monitoring verursacht einen Overhead von 700% im überwachten System (Zielsystem) [6].

Das Entwickeln neuer Methoden zur effektiven Überwachung eines Systems ist Gegenstand aktueller Forschung und Entwicklung der ITDesigners GmbH und der Hochschule Esslingen. Im nächsten Kapitel wird ein Ansatz einer Arbeit beschrieben, wie er momentan entwickelt und evaluiert wird.

[1] Monitoring wird im gleichen Thread ausgeführt wie das Zielsystem.

3 Dynamisches Monitoring für Publish/Subscribe-Systeme

In Publish/Susbscribe-Systemen (P/S-Systeme) kommunizieren die verschiedenen Komponenten eines Systems über ein gemeinsames Kommunikationssystem. Komponenten veröffentlichen (eng. publishen) Nachrichten auf diesem System unter bestimmten Kanälen (eng. Topics) und können gleichzeitig bestimmte Kanäle abonnieren (eng. subscriben). Veröffentlichen Komponenten Nachrichten unter einem Kanal, erhalten alle Abonennten die Daten [7].

Publish/Subscribe-Systeme erlauben eine sehr entkoppelte Systemarchitektur. Dadurch ist es möglich ein System sehr dynamisch aufzubauen und beispielsweise bestimmte Teilkomponenten zur Laufzeit auszutauschen. Damit eignen sich P/S-Architekturen gut für dynamische und fehlertolerante Systeme.

Zudem ist es sehr einfach möglich das Kommunikationssystem zu überwachen. Dies ist essentiell für die Diagnose eines Systems. Überwachungskomponenten (Monitore) können als zusätzliche Interessenten auf den verschiedenen Kanälen mithören und die Daten auswerten. Die Auswertung der Überwachung kann somit offline [2] stattfinden wodurch der Overhead im Zielsystem durch das Überwachungssystem deutlich reduziert werden kann.

In einer aktuellen Arbeit wird dieser Ansatz untersucht. Im Rahmen der Arbeit wird ein Framework entwickelt, um P/S-Systeme zu überwachen. Für die Fehlerdiagnose mit SFL ist es notwendig Abläufe in einem System zu verfolgen (**Tracing**). Transaktionen können jedoch nicht nur streng sequentiell sondern auch parallel und asynchron verlaufen. Während sequentielle Abläufe eines System relativ einfach zu verfolgen sind, gestaltet sich das Verfolgen von parallelen Abläufen weitaus schwieriger. Zudem ist in einem P/S-System die Überwachung auf das Kommmunikationssystem beschränkt. Werden innerhalb einer Komponente des Systems Daten asynchron verarbeitet, können Transaktionen nicht so einfach verfolgt werden.

Für die verschiedenen Arten von Transaktionen gibt es verschiedene Methoden um sie zu verfolgen. Diese unterscheiden sich am Grad des Systemeingriffs. Die verschiedenen Ansätze unterteilt Deen [8] in drei Klassen, die durch das Framework realisiert werden können:

- **Monitoring ohne Systemeingriff (eng. non-intrusive):** Monitore überwachen lediglich das Kommunikationssystem.
- **Monitoring mit leichtem Systemeingriff (eng. weak-intrusive):** Der Nachrichtenverkehr des Zielsystems wird zur Überwachung manipuliert.
- **Monitoring mit Systemeingriff (eng. intrusive):** Das Zielsystem wird weitergehend verändert, beispielsweise werden die Komponenten um einen ID-Mechanismus ergänzt.

Der Ansatz wird anhand eines Beispielsystems entwickelt und in zwei Fallstudien evaluiert. Zunächst wird das bestehende System zu einem P/S-System umgebaut. Als Kommunikationssystem wird das OpenSource-System Redis [9] verwendet. Da einzelne Komponenten über das externes Kommunikationssystem miteinander kommunizieren ist ein Overhead durch den Umbau nicht vermeidbar. In den Fallstudien werden die verschiedenen Möglichkeiten des Frameworks umgesetzt und demonstriert. Im Zuge der Arbeit sollen die folgenden Fragen beantwortet werden:

- **RQ1:** Wie stark sind die Auswirkungen der Monitoringarchitektur auf die Systemarchitektur?
- **RQ2:** Wie stark ist der Einfluss der Monitoringarchitektur auf die Performanz des Zielsystems?

[1] J.-C. Laprie, Ed, Dependability: basic concepts and terminology, Wien: Springer-Verlag, 1992.
[2] W. Görke and A. Endres, Fehlertolerante Rechensysteme. München: Oldenbourg, 1989.
[3] W. Torrespomales, Software Fault Tolerance: A Tutorial, pp. 2000-210616
[4] V. Venkatasubramanian, R. Rengaswamy, K. Yin, and S. N. Kavuri, A review of process fault detection and diagnosis, Computers and Chemical Engineering, vol. 27, no. 3, pp. 293-311, 2003.
[5] R. Abreu, P. Zoeteweij, and A. J. C. van Gemund, An observation-based model for fault localization, in the 2008 international workshop on dynamic analysis, p. 64.
[6] C. Chen, H.-G. Gross, and A. Zaidman, Improving Service Diagnosis through Increased Monitoring Granularity, in 2013 7th IEEE International Conference on Software Security and Reliability (SERE), pp. 129-138.
[7] Vargas, Pesonen et al. 2007 - Transactions in Content-Based Publish Subscribe Middleware
[8] Gert-Jan Deen 2012 - Dynamic Analysis of SOA, Master-Thesis, TU Delft
[9] http://www.redis.io/

[2] Monitoring wird außerhalb des Zielsystemthreads ausgeführt.