Karlsruher Institut für Technologie

Symptombasierte Fehlererkennung in heterogenen Systemen

Bachelorarbeit
von
Nico Rudolf
[Wissenschaftliches Schreiben: 2 ECTS gewünscht]

an der Fakultät Informatik

in dem Studiengang
Informatik

eingereicht am 15.09.2018
beim Institut für Technische Informatik
des Karlsruher Instituts für Technologie

Erstbetreuer: Prof. Dr. Wolfgang Karl
Zweitbetreuer: Thomas Becker

KIT – Universität des Landes Baden-Württemberg und
nationales Forschungszentrum in der Helmholtz-Gemeinschaft

Zeitablauf für die Anfertigung der wissenschaftlichen Arbeit

Arbeitsphase/Woche	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Betreuerabsprache	x	x				x				x				x		x
Thema entwickeln	x	x														
Gliederung erstellen		x	x													
Recherche		x	x	x												
Untersuchung			x	x	x	x	x									
Ergebnisse							x	x	x							
Schreiben										x	x	x	x			
Korrektur und Abgabe														x	x	
Präsentation																x

*Präsentation 4 Wochen nach Abgabe

1. Motivation
2. Theorie
    - 2.1 Grundidee der Symptombasierten Fehlererkennung
    - 2.2 Beschreibung des Versuchsaufbaus
3. Material und Methoden
    - 3.1 Symptommetriken
        - 3.1.1 Hardware Perfomance Counter
        - 3.1.2 Performance Application Programming Interface
    - 3.2 Fehlerinjizierung (Fault Injection)
        - 3.2.1 Physikalische Fault Injection
            - 3.2.1.1 Ausfall von CPU-Cores
            - 3.2.1.2 Pin-Level Injection
        - 3.2.2 Softwarebasierte Fault Injection
            - 3.2.2.1 Fault Injection in Userspace (Libfiu)
            - 3.2.2.2 Linux Fault Injection (Linux FI)
    - 3.3. Rodinia Benchmark Suite
4. Durchführung der Messungen
    - 4.1 Rechensystem: HPC Cluster
    - 4.2 Implementierung der Messungen
    - 4.3 Messergebnisse
5. Evaluation
    - 5.1 Vergleich der Ausführungsmetriken
    - 5.2 Indizierung der Fehlerklassen
    - 5.3 Anwendbarkeit der Symptombasierten Fehlererkennung
6. Zusammenfassung und Ausblick

Literaturverzeichnis

[1] Arlat, J. et al. 1990: Fault injection for dependability validation: a methodology and some applications. *IEEE Transactions on Software Engineering*, 16(2). pp.166-182.

[2] Badhwar, S. 2016. Performance profiling with perf - Fedora Magazine. [online] Fedora Magazine. Verfügbar unter: https://fedoramagazine.org/performance-profiling-perf/ [Zugriff am 20 Jun. 2018].

[3] Che, S. et al. 2009: Rodinia: A benchmark suite for heterogeneous computing. *2009 IEEE International Symposium on Workload Characterization (IISWC)*.

[4] Echtle, K. und Silva, J. 1998: Fehlerinjektion - ein Mittel zur Bewertung der Maßnahmen gegen Fehler in komplexen Rechensystemen. Informatik-Spektrum. 21(2). Pp. 328-336.

[5] Gil, D. et al. 2003: Study, comparison and application of different VHDL-based fault injection techniques for the experimental validation of a fault-tolerant system. *Microelectronics Journal*, 34(1), pp.41-51.

[6] Großpietsch, K. und Saglietti, F. 2000: Verlässlichkeit und Fehlertoleranz in Rechensystemen: Fortschritte, Probleme, Herausforderungen. *it - Information Technology*, 42(3).

[7] Intel® (2018). *Intel® Core™ i9-7900X Prozessor*. [online] Intel. Verfügbar unter: https://www.intel.de/content/www/de/de/products/processors/core/x-series/i9-7900x.html [Zugriff am 18 Jul. 2018].

[7] Jarboui, T. et al., J. 2002: Experimental analysis of the errors induced into Linux by three fault injection techniques. *Proceedings International Conference on Dependable Systems and Networks*.

[8] Li, L. 2016: Fault Detection and Fault-Tolerant Control for Nonlinear Systems. Wiesbaden : Springer Vieweg

[9] Performance Application Programming Interface. [Online]. Verfügbar unter: http://icl.cs.utk.edu/papi/ [Zugriff am 18 Jul. 2018].

[10] Reisner, S 2010: IT-Performance richtig testen und optimieren. Frankfurt am Main: entwickler.press.

[11] Svenningsson, R. et al. 2010: Model-Implemented Fault Injection for Hardware Fault Simulation. *2010 Workshop on Model-Driven Engineering, Verification, and Validation*.

[12] Tausche, K. (2014). Perfomance Counter: Non-Uniform Memory Access Seminar. [online] Dcl.hpi.uni-potsdam.de. Available at: https://www.dcl.hpi.uni-potsdam.de/teaching/numasem/slides/NUMASem_Performance_counter.pdf [Zugriff am 18 Jul. 2018].

Motivation

Aufgrund der stetig wachsenden Komponentenzahl in modernen Rechensystemen geht eine Steigerung der Ausfallwahrscheinlichkeit und Fehleranfälligkeit der Komponenten einher.

Eine Möglichkeit die Verlässlichkeit von Rechensystemen zu steigern, ist die Vermeidung von Fehlern während der Berechnung. Klassische Methoden der Fehlertoleranz wie Checkpoints oder Redundanz sind allerdings in modernen Systemen aufgrund der Vielzahl der Komponenten ineffizient und zu teuer. Deshalb ist ein alternativer Ansatz der Verzicht auf die Fehlervermeidung und auf die Erkennung von Fehlern zu setzen.

Man unterscheidet hierbei zwischen transienten Fehlern (soft errors) und intransienten Fehlern (hard errors). Soft errors sind von zufälliger und temporärer Natur, das bedeutet, dass ein soft error während einer Berechnung zu einem fehlerhaftem Ergebnis führen kann, bei Wiederholung der Berechnung wird dieser Fehler wahrscheinlich jedoch nicht auftreten.
Fehler die dauerhaft existieren, werden als hard errors bezeichnet. Beispiele für hard errors sind Hardware Ausfälle, denn ein Rechensystem würde unter diesen Umständen dauerhaft falsche Ergebnisse liefern.

Zurückzuführen sind transiente Fehler auf kosmische Partikel oder elektromagnetische Störfelder. Diese verursachen unter anderem ungewollte Bitflips in den Registern des Prozessors oder den Speicherzellen des Hauptspeichers. Dass Systeme immer häufiger von transienten Fehlern betroffen sind, ist im Wesentlichen auf die niedrigeren Versorgungsspannungen zurückzuführen. Hintergrund dafür ist zum einen der Wunsch nach energieeffizienten Produkten, aber auch zum anderen die zunehmende Integrationsdichte.

Neben der Verfälschung von Berechnungsergebnissen und dem Hervorrufen von kritischen Systemfehlern, die das System zum Absturz bringen, kann ein transienter Fehler auch folgenlos bleiben. Um Computersysteme vor solchen Fehlern zu schützen, gibt es mehrere Ansätze, wie die Redundanz.
Zum Einen gibt es die temporale Rendundanz, bei der eine Berechnung auf dem gleichen System mehrmals ausgeführt wird. Das berechnete Ergebnis wird dann verglichen und falsche Ergebnisse werden erkannt. Allerdings bedeutet das, dass die Kosten (Zeit) sich mindestens verdoppeln.

Räumliche Redundanz wird durch die parallele Ausführung auf mehreren (identischen) Rechensystemen realisiert. Beispielsweise benutzt das Space Shuttle fünf Computersysteme, um so per Mehrheitsentscheid bestimmen zu können. Die (Hardware-) Kosten steigen hierbei extrem an, allerdings ist es im Gegensatz zur temporalen Redundanz möglich, nicht nur soft errors sondern auch hard errors erkannt werden.

Eine Alternative zur teuren Redundanz, stellt eine leichtgewichtige Methode zur Erkennung von aufgetretenen Fehlern, wie die Symptom-basierte Fehlererkennung, dar.

Die Grundidee ist die Erkennung von Fehlern mittels Abweichungen vom üblichen Ausführungsverhalten. Das Verhalten wird mittels Metriken, sogenannten Performance Counter, wie z.B. die Anzahl an TLB Misses, charakterisiert. Zur Festlegung des normalen Ausführungsverhalten wird eine Datenbank mit Werten verschiedener Ausführungsmetriken von korrekten Ausführungen angelegt.

Daraufhin werden Ausführungen gezielt mit Fehlern injiziert um die Metriken auf signifikante Abweichungen zu untersuchen. Zur Laufzeit werden dann die aktuellen Werte mit der Datenbank verglichen. Sofern die Abweichungen einer oder mehrerer Metriken einen festgesetzten Grenzwert überschreiten könnte dies als Indikator für diesen Fehler gelten.

Ziel dieser Bachelorarbeit ist die Untersuchung der Anwendbarkeit der Symptombasierten Fehlererkennung. Dabei wird gezeigt ob das Auftreten von Fehlern oder sogar die aufgetretene Fehler(-art) bestimmt werden können.

3.1.1 Hardware Perfomance Counter

Hardware Perfomance Counter sind spezielle Mehrzweckregister die in modernen Prozessoren eingebaut sind. Diese Register werden von der on-chip Performance Monitoring Unit (kurz: PMU) verwaltet und beschrieben. Das Ziel der PMU ist das Dokumentieren/Zählen von Aktivitäten, wie z.B. Zyklen, Cache Misses oder Floating Point Operations.

Ein Pentium III Prozessor (1999) besitzt zwei Hardware Perfomance Counter, wohingegen ein aktueller Prozessor wie der Intel Xeon E5-2620 (2014) 634 Perfomance Counter zur Verfügung stellt[12], wie der Tabelle im Anhang entnommen werden kann.

3.1.2 Performance Application Programming Interface

Die University of Tennessee entwickelt das Performance Application Programming Interface (kurz: PAPI) um einen leichteren Zugriff auf perfomance counter anzubieten. Das einheitliche PAPI-Interface bietet Entwicklern eine Abstraktion der nativen Interfaces und somit eine vereinfachte Entwicklung. [9]

PAPI unterstützt alle aktuellen Intel und AMD x86 sowie x86_64 Prozessoren (CPU). Durch die Erweiterung PAPI CUDA wird der Zugriff auf perfomance counter von nVidia Grafikkarten (GPU) erweitert.

Es wird zwischen high-level interface sowie low-level interface unterschieden. Das high-level interface bietet einen einfachen sowie schnellen Zugriff, wohingegen das low-level interface mehr Kontrolle und einen detailliertes Vorgehen erlaubt. In dieser Arbeit wird vor allem das low level interface benutzt. Informationen über die Implementierungsarbeit ist in Abschnitt 4 zu finden.

Jede Plattform (CPU, GPU,..) besitzt sogenannte native events, die aus den unterstützen hardware Performance Countern abgeleitet werden. Die Namen dieser events sind plattformabhängig. Durch generische preset events abstrahiert PAPI diese nativen events und ermöglicht dem Entwickler eine einheitliche Sicht ohne exakte Kenntnisse der native events zu fordern. Sofern ein preset event nicht verfügbar ist versucht PAPI dieses von anderen events abzuleiten, z.B. Total Number of L2 Cache Misses = L2 Cache Access - L2 Cache Hits.

Die Konfiguration der Events wird durch EventSets ermöglicht. Diese bestehen aus einem oder mehreren Events und klassifizieren wie die events gemessen werden sollen, z.B. ob der code im user space oder kernel space ablaufen soll.

3.2 Fehlerinjizierung (Fault Injection)

Um die Anwendbarkeit der symptombasierten Fehlererkennung zu untersuchen ist eine Simulation von Fehlern notwendig. Im Folgenden werden verschiedene Arten der Fehlerinjektion vorgestellt die in dieser Thesis angewandt wurden.

3.2.1 Physikalische Fehlerinjektion

Moderne Prozessoren verfügen über eine hohe Anzahl von Rechenkernen. So besitzt beispielsweise der Intel Core i9-7900X über 10 physikalische Rechenkerne [7]. Die effektive Nutzung mehrerer Kerne erfordert, dass die Software so programmiert ist, dass sie Daten parallel verarbeitet und so mehrere Kerne gleichzeitig genutzt werden.

Eine beobachtbare Fehlerinjektion ist das gezielte Deaktivieren von Kernen über das BIOS. Das Verhalten einer parallelen Anwendung wird untersucht unterschiedlicher Anzahl deaktivierter Kerne. Das Verhalten wird durch Performance counter klassifiziert und signifikante Abweichungen evaluiert.