Applikationsgeführte softwareinduzierte Fehlerinjektion eines fehlertoleranten Stellwerkscomputers

Saša Vulinović [1], Bernd-Holger Schlingloff[2,3]

[1]Technische Universität Berlin,
Institut für Technische Informatik
und Mikroelektronik
Franklinstr. 28/29, 10587 Berlin
vulinovi@cs.tu-berlin.de

[2]Fraunhofer Institut
für Rechnerarchitektur und
Softwaretechnik FIRST
Kekulèstr. 7, 12489 Berlin
Holger.Schlingloff@first.fhg.de

[3]Humboldt-Universität zu Berlin
Institut für Informatik
Rudower Chaussee 25,
12489 Berlin
hs@informatik.hu-berlin.de

Abstract: Dieser Bericht beschreibt die Implementierung und den Einsatz einer softwarebasierten Fehlerinjektionsumgebung (SWIFI) bei der Zertifizierung eines sicherheitskritischen, auf Intel Pentium MMX Prozessoren basierenden Rechnersystems. Die Auswahl der eingestreuten Fehler erfolgt dabei auf funktionaler Ebene, das heißt, auf Grund einer statischen und dynamischen Analyse des Programmcodes der Applikation. Die Fehlerinjektion wird mit Standardwerkzeugen auf Assemblerebene per Debugger durchgeführt. Kritische Ausfälle bei der Simulation geben Hinweise auf mögliche Schwachstellen im Entwurf. Falls bei der Simulation keine kritischen Ausfälle auftreten, kann die Aussagesicherheit der geforderten Ausfallrate auf Grund der vollständigen Automatisierung der Simulationsumgebung beliebig gesteigert werden.

Keywords: Fault injection, Fault tolerance, Fail-safety, Cenelec EN 50129, ETCS, SIL4, SWIFI, COTS Certification

Einführung

Die im Februar 2003 verabschiedete Cenelec-Norm EN 50129 für Bahnanwendungen (Telekommunikationstechnik, Signaltechnik und Datenverarbeitungssysteme — Sicherheitsrelevante elektronische Systeme für Signaltechnik) fordert in Abschnitt B.3.1 für Systeme der Sicherheitsstufe SIL3 oder SIL4, dass auch bei beliebigen anzunehmenden Einzelfehlern kein kritischer Ausfall eintritt. Dieses „fail-safety"-Konzept kann dadurch erreicht werden, dass jede sicherheitsrelevante Funktion von wenigstens zwei unabhängigen Einheiten erbracht wird (composite fail-safety). Falls aus gewissen Gründen eine sicherheitskritische Funktion nur von einer Komponente erbracht werden kann, so ist nachzuweisen, dass alle denkbaren Ausfälle entweder unkritisch sind oder ihr Eintreten praktisch unmöglich ist, zum Beispiel auf Grund unverlierbarer physikalischer Eigenschaften (inherent fail-safety).
In der vorliegenden Arbeit beschäftigen wir uns mit der einkanalig sicheren Ausgabe von Fahrtfreigaben der Streckenzentrale an die Bordcomputer im Funkfahrbetrieb des ETCS (European Train Control System). Dieses System soll in absehbarer Zukunft die verschiedenen nationalen Zugsicherungssysteme für den Hochgeschwindigkeitsverkehr in Europa ersetzen und die Netze interoperabel machen. ETCS überwacht die Bewegung von Zügen zum Beispiel hinsichtlich der Freigabe von Fahrwegen. Neu dabei ist, dass sowohl Schienenfahrzeuge als auch Streckenzentralen mit Rechnern ausgestattet sind, welche mittels des neuen Mobilfunkstandards GSM-R miteinander kommunizieren. Die (sicherheitskritische) Berechnung der Freigabemeldungen erfolgt dabei gemäß dem composite-fail-safety Konzept dreifach fehlertolerant im „Radio Block Center" RBC, dem Rechner der Streckenzentrale. Zur Ausgabe werden die verschiedenen Nachrichten durch ein n-aus-m-Voting-Verfahren konsolidiert. Da die Ausgabe einkanalig erfolgt, muss diese Funktion nach dem inherent-fail-safety Konzept realisiert werden. Die Besonderheit des vorliegenden Systems ist nun, dass das RBC mit aktuell verfügbarer Standardhardware (COTS) realisiert wurde und das Voting komplett in Software stattfindet. Da die geplante Produktlebensdauer mehr als 25 Jahre beträgt, sind Hardwareausfälle (z.B. SEU, single event upsets) nicht auszuschließen. Um die oben genannten Sicherheitskriterien eines SIL4-Systems nach Cenelec zu erfüllen, muss nachgewiesen werden, dass es praktisch unmöglich ist, dass solche SEUs zu einem kritischen

Ausfall führen. Nachfolgend beschreiben wir eine Methode, mit der dieser Nachweis durch eine Quantifizierung der kritischen Ausfallrate erbracht wird.
Um die durch Ausfälle von Hardware bedingte Gefährdungsrate zu ermitteln, wurde am Fraunhofer Institut FIRST eine Simulationsumgebung entwickelt, die es ermöglicht, sicherheitskritische Systeme mittels Fehlerinjektion zu analysieren (Software Implemented Fault Injection, SWIFI). Dabei werden gezielt Fehler in das System eingestreut, die den Effekt von Hardware-Fehlern imitieren. Das Zielsystem wird durch diese Simulationsumgebung mit einer großen Anzahl statistisch ausgewählter solcher »Fehler« konfrontiert, die systematisch alle möglichen Hardwareausfälle abdecken. Im Vergleich zu einer Hardware-Fehlerinjektion ist eine Software-Fehlerinjektion zerstörungsfrei und reproduzierbar; im Vergleich zu analytischen Methoden ist sie skalierbar und liefert statistisch relevante Aussagen. Die angewendeten Techniken sind dabei vergleichbar mit der Vorgehensweise beim modellbasierten Softwaretest.

Diese Arbeit ist folgendermaßen gegliedert: zunächst beschreiben wir das zu untersuchende System und die mathematischen Grundlagen der Analyse. Danach geben wir einen kurzen Überblick über die von uns implementierte Simulationsumgebung. Der nächste Abschnitt dient der Beschreibung des Auswahlverfahrens für die einzustreuenden Fehler in der Fehlerdatenbank. Dann geben wir die Ergebnisse der Simulation für das RBC an und diskutieren die daraus resultierenden Konsequenzen. Im letzten Abschnitt fassen wir unsere Ergebnisse zusammen und stellen einige Ideen für weitergehende Forschungsarbeiten vor.

Problemstellung

Das zu bewertende System besteht aus einem Schrank mit drei Pentium® MMX Boards, welche untereinander ringförmig verschaltet sind. Als Betriebsystem wird eine modifizierte Version des SUN™ ChorusOS Kerns verwendet (EOS, Extended Operating System). Der Kern ist so erweitert worden, das die darauf laufenden Applikationen die Möglichkeit haben, untereinander Nachrichten zu versenden (vergleichbar den POSIX message queues). Die Erweiterung ermöglicht dabei eine implizite Konsolidierung (n aus m Voting) der redundant versendeten Nachrichten. Die Nachrichten selbst sind durch einen Sicherungsanhang geschützt; ihre Erzeugung sowie die Generierung des Sicherungsanhanges erfolgt auf allen Repliken parallel und unabhängig. Die Konsolidierung der Nachrichten erfolgt durch einen Software-Voter, und zwar für die Applikationen transparent. Auf allen drei Repliken läuft die selbe Applikationssoftware und Kommunikations- und Fehlertoleranzschicht, welche das System in zyklischen Abständen auf permanente Fehler prüft. Dabei ist der Ausgabekanal nur an eine der drei Repliken angeschlossen. Im Rahmen der Problemstellung werden nur zufällige Fehler (*random faults)* analysiert; systematische Fehler im Entwurf der Soft- oder Hardware werden nicht betrachtet.

Eine Analyse der möglichen Ausfallarten ergab, dass der einzige denkbare Ausfall, der zu einem kritischen Versagen des Systems führen könnte, darin besteht, dass ein und derselbe Hardwarefehler auf der ausgebenden Replik

- die Applikationssoftware so beeinflusst, dass eine fehlerhafte Nachricht erzeugt wird,
- auf die Software zur Erzeugung der Sicherungsanhänge keinen Einfluss hat, und dann
- das Voting so zerstört, dass nicht die zweifach vorliegende korrekte, sondern die verfälschte Nachricht am Ausgabekanal erscheint.

Obwohl intuitiv klar ist, dass diese Ereignisfolge praktisch unmöglich und somit bei einer Fehlerbaumanalyse nicht anzunehmen ist, muss für eine Zertifizierung ihre Wahrscheinlichkeit quantitativ nach oben abgeschätzt werden. Hierfür wurde, wie oben beschrieben, ein experimenteller Ansatz mittels SWIFI gewählt, bei dem das Verhalten des Systems unter zufälligen eingestreuten Fehlern beobachtet wurde.
Unter der Annahme, dass ein System bei allen Beobachtungen niemals versagt, ist die Versagenswahrscheinlichkeit p des Systems natürlich nicht bestimmbar; bei einem System, welches bei keinem einzigen bisherigen Simulationslauf versagt hat, lässt sich ohne weitere Informationen keine Wahrscheinlichkeit angeben, mit der beim nächsten Lauf ein gefährlicher Ausfall auftreten wird. Für die Aufgabenstellung ist jedoch lediglich eine obere Schranke für p zu ermitteln. Es soll gezeigt werden, dass p unterhalb eines bestimmten Grenzwertes bleibt: $p \leq \mu$, wobei μ hier in der Größenordnung von 10^{-10} Fehlern/Betriebsstunde festgelegt wurde.

(Cenelec EN50129 schreibt eine tolerierbare Fehlerquote THR zwischen 10^{-8} und 10^{-9} für SIL4 vor). Mit Simulationsläufen ist diese Aussage zwar nicht nachweisbar, jedoch ihre Wahrscheinlichkeit, das heißt die *Aussagesicherheit β* der Hypothese *(p ≤ μ),* kann abgeschätzt werden:

$$\beta = P[p \le \mu] .$$

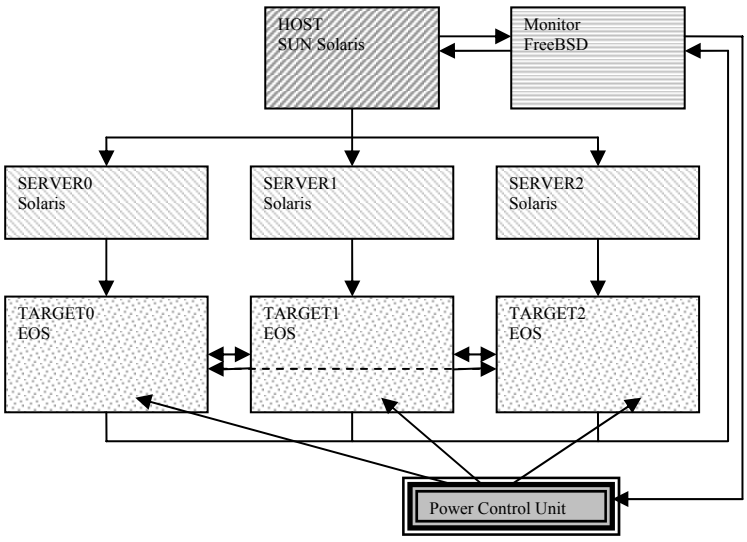
Eine mathematische Analyse ergibt für *β* den Zusammenhang

$$\beta = 1 - (1 - \mu)^n .$$

Die aus einer Hardwareanalyse ermittelte intrinsische Fehlerrate des Systems liegt bei 10^{-6} Fehlern/Betriebsstunde (Summe der Ausfallwahrscheinlichkeiten der einzelnen HW-Komponenten). Jede Fehlerinjektion simuliert also einen Zeitraum von ca. 10^6 Stunden. Das bedeutet also, um mit 98% Wahrscheinlichkeit sagen zu können, dass die Fehlerwahrscheinlichkeit kleiner als 10^{-10} ist, müssen 40000 Simulationsläufe ohne einen einzigen gefährlichen Ausfall durchgeführt werden. Diese Grenze wird von der Zulassungsbehörde festgesetzt. Vorraussetzung ist dabei eine größtmögliche *Betriebstreue*, das heißt, die Verteilung der injizierten Fehler entspricht der antizipierten Verteilung im realen Betrieb.

Aufbau der Simulationsumgebung

Die nachstehende Abbildung zeigt den Aufbau der Simulationsumgebung. Das Zielsystem besteht aus den drei Repliken des RBC, hier als „Target“ gekennzeichnet. Die softwaregesteuerten Fehlerinjektionen werden mit Hilfe des Debuggers „chorus-gdb“, einem Derivat des Gnu „gdb“, durchgeführt, welcher auf den Targets und den Servern ausgeführt wird. Der Host agiert als Master im System und steuert den Injektionsablauf. Wegen der hohen Realzeitkritikalität und den empfindlichen Synchronisationsmechanismen ist nötig, für jedes Target einen separaten Server als Host für den „chorus-gdb“ und NFS-Server bereit zu stellen. Der Monitor beobachtet die Reaktionen des Zielsystems auf die eingestreuten Fehler und protokolliert die Ergebnisse. Um die Targets in einen definierten Ausgangszustand zu versetzen, können sie von der Simulationsumgebung mit der Power Control Unit neu gestartet werden. Die Realisierung der Simulationsumgebung nahm etwa ein halbes Personenjahr in Anspruch. Hauptprobleme waren die Synchronisation der drei Repliken und die Komplexität des Befehlssatzes der Pentium CPU.



Auswahl der zu injizierenden Fehler

Auf Grund der oben genannten Forderung nach Betriebstreue reicht es nicht aus, wahllos Fehler in das Zielsystem zu injizieren; die Fehler müssen nach einer überprüfbaren Strategie ausgewählt werden. Dieses Problem ist vergleichbar mit der Aufgabe, eine Testsuite mit

hinreichend großem Überdeckungsgrad für ein gegebenes System zu konstruieren. Während im Bereich der Testautomatisierung verschiedene Strategien zur Generierung von Testsuiten mit hoher Überdeckung existieren, ist dies für die SWIFI ein offenes Problem. Im Gegensatz zu herkömmlichen Ansätzen der Fehlerinjektion, bei denen die Strategie dem Aufbau der Hardware folgt, wird nachfolgend ein Auswahlverfahren beschrieben, welches die Nutzung der Hardware durch die gegebene Software widerspiegelt. Abgesehen davon, dass Details zur physikalischen Implementierung des Pentiums Prozessors nur schwer zugänglich sind, wird auf diese Weise eine realistischere Abdeckung der Fehler, die im tatsächlichen Betrieb zu erwarten sind, erreicht.
Im vorliegenden Fall beruht das Auswahlverfahren auf folgenden Annahmen:

i. *Jeder* Fehler, der im System passiert, hat auch eine Auswirkung auf die untersuchte Fehlertoleranzschicht (worst-case Annahme).
ii. Komponenten, die nicht verwendet werden, können auch keine Fehler verursachen und brauchen nicht betrachtet zu werden.

Weiterhin nehmen wir an, dass die Fehlertoleranzschicht nicht von sich aus neue Nachrichten mit gültigem Sicherungsanhang erzeugen kann, und dass jeder kritische Ausfall dadurch erkennbar ist, dass eine Nachricht, welche in der Minderheit im Voter vorliegt, als konsolidierte Nachricht ausgegeben wird.
Wegen i) und ii) manifestieren sich alle potentiell kritischen Fehler auf funktionaler Ebene, zu dem Zeitpunkt und an dem Ort, an dem eine fehlerhafte Komponente verwendet wird. Das heißt, dass alle relevanten Fehler durch entsprechende Manipulationen auf Assembler-Ebene nachgebildet werden können. Die Verwendung der Komponenten der CPU ist auf Assembler-Ebene implizit im Programmcode des Voters enthalten. Durch statische und dynamische Untersuchung des Programmcodes kann daher eine gewichtete Liste von CPU-Komponenten zusammengestellt werden, welche für eine Injektion von Fehlern in Betracht kommt: Register, Speicheradressen, Maschinenbefehle, Operandenwerte usw.

Im untersuchten System gibt es, abhängig von der jeweiligen Lastsituation, viele mögliche Programmpfade. Um eine Aussage über die dynamische Verwendung von Ressourcen des Systems zu erhalten, werden mehrere so genannte Referenzläufe durchgeführt. Als Gewichtung für die Auswahl des Injektions-Ortes werden statistische Mittelwerte aus diesen Referenzläufen herangezogen. An häufig besuchten Programmstellen werden somit also statistisch öfter Fehler injiziert als an anderen. Das System erhält dabei verschiedene Eingangsdaten, welche mit unterschiedlichen Verzögerungen an den jeweiligen Targets als Last anliegen. Hieraus ergibt sich ein typisches Ausführungsprofil.

In unserem Verfahren werden die so erhaltenen Daten verwendet, um eine Fehlerdatenbank, d.h., eine Datei mit zu injizierenden Fehlern, zu generieren: Mit einer parametrisierbaren Gewichtungsfunktion werden dabei (statistisch zufällig) Komponenten ausgewählt und jeweils entsprechende zu injizierende Fehler generiert. Eine Manipulationsadresse ist dabei die Angabe der zu manipulierenden Operation, des Operanden, der Speicheradresse oder des Registers. Eine Manipulation besteht aus dem Invertieren bestimmter Bits dieser Adresse. Ein Fehler ist also gegeben durch das Tupel

(laufende Nummer, Unterbrechungspunkt, Manipulationsadresse, Manipulation, Lastfall)

Bei der Ausführung wird die Fehlerdatenbank zeilenweise abgearbeitet. Beim (ggf. wiederholten) Erreichen des Unterbrechungspunktes wird das entsprechende Target angehalten, per Debugger der Fehler injiziert und das System für eine bestimmte Zeit fortgesetzt. Die Reaktion des Systems wird an Hand des Ausgabekanals vom Monitor festgestellt und protokolliert.

Durchführung und Ergebnisse

Da die Simulationsumgebung vollständig automatisiert ist, können beliebig viele Simulationsläufe praktisch ohne manuellen Aufwand durchgeführt werden. Unter der Voraussetzung, dass kein gefährlicher Ausfall auftritt, kann somit die Aussagesicherheit für die Unterschreitung einer vorgegebenen Fehlerrate beliebig gesteigert werden. Die Durchführung eines Simulationslaufes dauert dabei, bedingt durch den Neustart des Zielsystems, etwa zwei Minuten.

In diesem Projekt wurden 40.000 Simulationsläufe durchgeführt, die an Hand der oben beschriebenen Kriterien ausgewählt wurden. Die an Hand der Referenzläufe ausgewählten Fehler wurden in den meisten Fällen bei der Simulation auch aktiviert (ausgeführt) und führten meist zu einem nicht kritischen Ausfall (Absturz einer oder mehrere Repliken oder zu einem automatischen Reset der betroffenen Replik). Injizierte Fehler, die nicht auf Grund des Referenzlaufs ausgewählt wurden, wurden meist nicht aktiviert und führten daher auch nicht zu einem Ausfall des Systems. Bei ähnlichen Projekten ist also zu beachten, dass auf die Auswahl der zu injizierenden Fehler entsprechendes Augenmerk gerichtet wird, wobei die Referenzläufe unter betriebstreuen Bedingungen durchzuführen sind.
Bei der Durchführung der Simulation traten auch zwei kritische Ausfälle auf. Der erste kritische Ausfall betraf den Aufruf einer Prozedur, die vom Fehlerinjektionsmechanismus verfälscht wurde, was nachfolgend vermutlich dazu führte, dass die Programmlogik in einen falschen Zustand übergeht. Der zweite kritische Ausfall trat beim Rückkehr von einem Aufruf einer Prozedur auf, bei der der Rückgabewert im Prozedurkeller vom Fehlerinjektionsmechanismus verfälscht wurde.

Auf Grund dieser Resultate wurden weitere Maßnahmen in die Voting-Software integriert, die diese und ähnliche kritische Ausfälle auch unter anderen eingestreuten Fehlern verhindern sollen. Um die Wirksamkeit dieser Änderungen zu überprüfen, ist es notwendig, die Simulationsläufe erneut durchzuführen. Diese Wiederholungs-Simulationen werden derzeit durchgeführt.

Ausblick

In diesem Papier haben wir eine Methode zur applikationsgesteuerten Fehlerinjektion für die quantitative Evaluation sicherheitskritischer Systeme vorgestellt. Im Gegensatz zu bekannten Fehlerinjektions-Verfahren orientiert sich die Auswahl der zu injizierenden Fehler dabei an der tatsächlichen Nutzung der Systemressourcen durch die jeweilige Software. Dadurch wird einerseits ein höherer Abstraktionsgrad als auch ein besserer Überdeckungsgrad als bei herkömmlichen Methoden erreicht. Die Resultate beweisen, dass mit dem vorgestellten Verfahren tatsächlich kritische Programmstellen im Zusammenspiel von Hard- und Software identifiziert werden können, die sich anderen Analysemethoden entziehen. Bei erfolgreicher Durchführung einer Suite von Simulationsfällen erlaubt die Methode, die Aussagesicherheit für die Einhaltung einer maximalen kritischen Ausfallrate beliebig zu steigern.

Die vorliegende Arbeit lässt eine Reihe weitere Fragen offen: Zur Zeit führen wir einen quantitativen Vergleich unseres Verfahrens mit herkömmlichen Methoden, zum Beispiel der Simulation auf Gatterebene, durch. Auch die Frage der Modellierung von physikalischen Fehlerursachen und Fehlereffekten auf funktionaler Ebene, also das Zusammenspiel von fehlerhafter Hardware mit korrekter Software, ist noch nicht hinreichend geklärt. Des weiteren wäre interessant, die hier vorgestellten Ansätze zur Fehlerinjektion mit formalen Verifikationstechniken wie z.B. Model Checking oder Codeanalyse zu kombinieren. Erste Überlegungen in dieser Richtung haben bereits angefangen.

Literatur

[1] CENELEC EN 50129: Railway applications – Communication, signalling and processing systems – safety related electronic systems for signalling, Europäische Norm, Okt. 2003.

[2] Rodriguez, M., Salles, F., Fabre, J.C., Arlat, J.: MAFALDA: *Microkernel Assessment by Fault Injection and Design Aid*. Proceedings of the 3rd European Dependable Computing Conference (EDCC-3), 143–160, 1999

[3] M.C.Calzarossa and S. Tucci (Eds.): *Measurement-Based Analysis of System Dependability Using Fault Injection and Field Failure Data.* Performance 2002, LNCS 2459, pp.290-317, 2002.

[4] Lorinc Antoni, *Injection de Fautes par Reconfiguration Dynamique de Reseaux Programmables*, These pour obtenir le grade de Docteur d´l Inpg , l´Ecole Doctorale d´Electronique, Electronique, Automatique, Telecommunications, Signal, 2003,

[5] W. Ehrenberger : Software-Verifikation ; Verfahren für den Zuverlässigkeitsnachweis von Software. Hanser, 2002

[6] B. Parrotta, M. Rebaudengo, M. Sonza Reorda, M. Violante: *Speeding-Up Fault Injection Campaigns in VHDL Models*, SAFECOMP 2000, pp. 27-36, 2000.

[7] A. Benso, M. Rebaudengo, L. Impagliazzo, P. Marmo: *Fault-list collapsing for fault injection experiments*, Proc. Ann. Reliability & Maintainability Symp., pp. 383-388, 1998.