

Verlässliche Echtzeitsysteme

Übungen zur Vorlesung

Codierung, Fehlerinjektion

Florian Schmaus, Simon Schuster

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)
<https://www4.cs.fau.de>

14. Mai 2018



- 1 Fehlerinjektion mit FAIL*
- 2 Wiederholung Software-TMR
- 3 Eliminierung von Bruchstellen in TMR
- 4 Aufgabenstellung



- 1 Fehlerinjektion mit FAIL***
- 2 Wiederholung Software-TMR
- 3 Eliminierung von Bruchstellen in TMR
- 4 Aufgabenstellung



 https://www4.cs.fau.de/Lehre/SS18/V_VEZS/Uebung/Folien/Fail.pdf



```
1 #ifndef _include_fail_h
2 #define _include_fail_h
3
4 //Mark start of injection
5 void __attribute__((noinline)) fail_start_trace(void);
6 //Mark end of injection
7 void __attribute__((noinline)) fail_stop_trace(void);
8
9 //everything ok: valid code and right values
10 void __attribute__((noinline)) fail_marker_positive(void);
11 //everything ok: valid code but wrong values
12 void __attribute__((noinline)) fail_marker_negative(void);
13 //invalid code
14 void __attribute__((noinline)) fail_marker_detected(void);
15
16 #endif /* _include_fail_h */
```

- Markierung Start/Ende für „Golden Run“
- Ziel: Minimierung Auftreten von fail_marker_negative



FAIL* – Beispiel: Verwendung Marker

```
1 void cyg_user_start() {
2     ...
3     fail_trace_start() // Start Fehlerinjektion
4     filter()
5     vote()             // (codierter) Mehrheitsentscheid
6     fail_trace_stop() // Ende Fehlerinjektion
7
8     actuate()         // Signaturen entfernen, Fehler überprüfen
9
10    ...
11 }

1 void actuate() {
2     ...
3     if (checksum == expected_checksum){
4         fail_marker_positive() // SDC behandelt :-
5     }else{
6         fail_marker_negative() // undetektierte SDC :-
7     }
8 }
9
```



- Datei anlegen: `~/ .my.cnf`
- Folgende Daten eintragen:
`[client]`
`user=vezsXX`
`password=XXXXXXXXXXXXXXXXXX`
`host=i4jenkins.informatik.uni-erlangen.de`
`database=vezsXX`
- Zugangsdaten wurden verteilt



1. make trace

- Erstellung des „Golden Runs“
- Speicherung der Experimente in Datenbank
- Faltung des Fehlerraums

2. make inject

- Durchführung Fehlerinjektion
- Überblick der Ergebnisse in lokal angelegter Datei: `ean_result.csv`
 - 👁️ Abschätzung des zeitlichen Aufwands

3. make resultbrowser

- Startet lokalen Webserver
- Ergebnisse per Webbrowser einsehbar



The screenshot shows a web browser window with the URL `localhost:5000/instr_details?table=result_GenericExperimentMessage&instr_address=1048643&variant_id=1`. The page title is "FAIL*" and it has "Home" and "About" links. The main heading is "Result by Instruction".

Metadata:

- Result Table: `result_GenericExperimentMessage`
- Variant: `main`
- Benchmark: `mem`
- Details
- Instruction Address: `0x100043 (Dec: 1048643)`
- Total Results: `32`

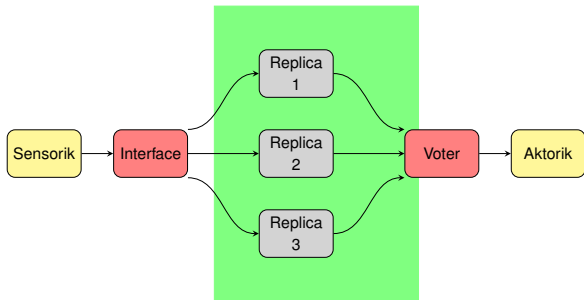
Code

Address	Opcode	Disassembly	Comment
0x10002f	90	<code>nop</code>	
0x100030	c7052010200064	<code>movl \$0x64,0x201020</code>	
0x100037	0000	<code>00</code>	
0x10003a	c3	<code>ret</code>	
0x10003b	6690	<code>xchg %ax,%ax</code>	
0x10003d	6690	<code>xchg %ax,%ax</code>	
0x10003f	90	<code>nop</code>	
0x100040	83ec0c	<code>sub \$0xc,%esp</code>	
0x100043	d9442418	<code>flds 0x10(%esp)</code>	
0x100047	dc442410	<code>faddl 0x10(%esp)</code>	
0x10004b	dd1c24	<code>fstpl (%esp)</code>	
0x10004e	dd0424	<code>fldl (%esp)</code>	
0x100051	83c40c	<code>add \$0xc,%esp</code>	
0x100054	c3	<code>ret</code>	
0x100055	6690	<code>xchg %ax,%ax</code>	
0x100057	6690	<code>xchg %ax,%ax</code>	
.....

- 1 Fehlerinjektion mit FAIL*
- 2 Wiederholung Software-TMR**
- 3 Eliminierung von Bruchstellen in TMR
- 4 Aufgabenstellung



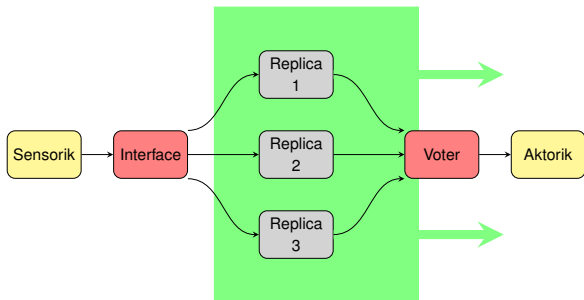
Klassische “Triple Modular Redundancy” (TMR)



- Schnittstelle sammelt Eingangsdaten (Replikdeterminismus)
- Verteilt Daten und aktiviert Replikate
- Mehrheitsentscheider (Voter) wählt Ergebnis
- Ergebnis wird an Aktuator versendet



Klassische "Triple Modular Redundancy" (TMR)



Redundanzbereich

Ausschließlich Replikatausführung

- ~> Erweiterung der Ausgangsseite mit Informationsredundanz
- ~> Mehrheitsentscheid über codierte Prüfsumme



- 1 Fehlerinjektion mit FAIL*
- 2 Wiederholung Software-TMR
- 3 Eliminierung von Bruchstellen in TMR**
- 4 Aufgabenstellung



Erweiterte arithmetische Codierung

nach Forin 1989: "Vital coded microprocessor principles and application for various transit systems" [1]

- Arithmetisch codierter Wert V_C
- Ausgangswert

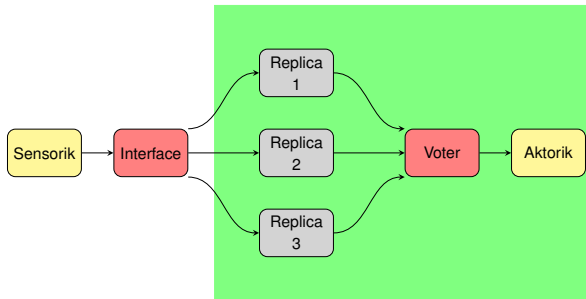
$$V_C = V * A + B_V + D$$

- Schlüssel
- Variablenspezifische Signatur
- Zeitstempel



- Schlüssel A sollte so groß wie möglich sein:
 - ↪ Möglichst geringe Restfehlerwahrscheinlichkeit ($P = 1/A$)
- Wertebereich des dynamischen Zeitstempels
 - $D = \{x \mid x \in \mathbb{N}_0 \wedge x \leq D_{max}\}$
 - Zeitstempel darf überlaufen: $D_{max} + 1 = 0$
- Für jede Signatur B_* muss dann gelten
 - $B_* + D_{max} < A$
 - Die minimale Distanz zwischen jeweils zwei Signaturen im System muss kleiner D_{max} sein: $\forall i, j : |B_i - B_j| < D_{max}$





- Replikate liefern arithmetisch codierte Ergebnisse
- Mehrheitsentscheid auf codierten Prüfsummen
- Übertragung codierter Ergebnisse



Für diese Übungsaufgabe:

- Kein Zeitstempel
- Nur Absicherung der Ausgangsseite!



EAN Vergleichsoperator

- Voting basiert auf codierter Vergleichsoperation:

$$\rightsquigarrow X_C = Y_C \Rightarrow X * A + B_X = Y * A + B_Y$$

- Im fehlerfreien Fall gilt:

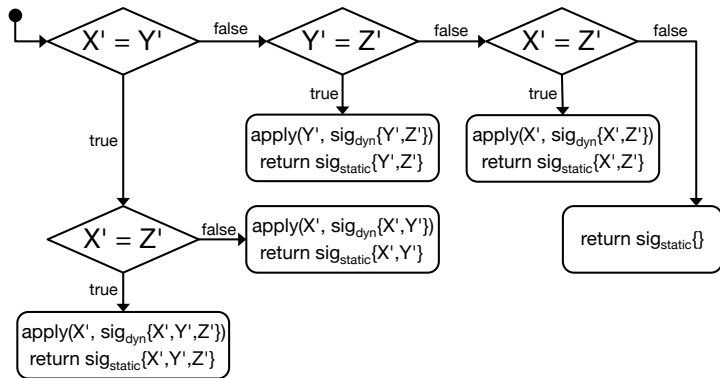
$$X = Y, A = A \text{ aber } B_X \neq B_Y !$$

- Rohwerte sind identisch
- Schlüssel ist per Definition identisch
- Signaturen sind unterschiedlich (aber konstant!)

Bestimmung der Gleichheit durch Differenzbildung:

$$\rightsquigarrow X_C - Y_C = B_X - B_Y = \text{const.}$$





■ Bestimmung von dynamischer und statischer Signatur:

~> $sig_{dyn}(X', Y') : X' = Y' \Rightarrow X' - Y'$

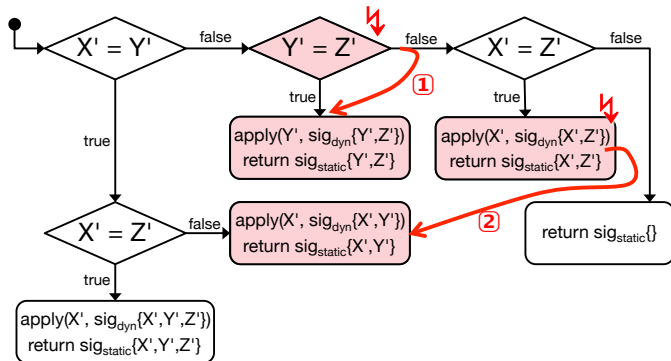
~> $sig_{static}(X', Y') : X' = Y' \Rightarrow B_X - B_Y$

1. Vergleichsoperation wird durchgeführt (z. B. $X' = Y' \wedge X' = Z'$)
 - Berechnung von sig_{dyn}
 - Vergleich mit sig_{static}
2. Verzweigungsentscheidung wird nachberechnet:
 - Wiederholte (redundante) Berechnung von sig_{dyn}
 - Addiere sig_{dyn} (*apply*) zum gewählten Ergebnis
3. Konstante Signatur des durchlaufenen Zweiges identifiziert Gewinner (Rückgabewert: sig_{static})
 - Akteur wählt entsprechendes Replikatergebnis
 - Führt inverse Operation zu *apply* durch

Im Voter wurde die *dynamisch berechnete Signatur der Verzweigungsentscheidung* hinzu addiert. Im Akteur wird mit der entsprechenden *konstanten Signatur zurückgerechnet*.



Codierter Mehrheitsentscheid - Fehlerfall



1. Falsche Verzweigungsentscheidung: ($Y' \neq Z'$)

- Y' wird als korrekt angenommen, sig_{dyn} wird berechnet
- allerdings ist sig_{dyn} tatsächlich $\neq sig_{static}$
- Fehler wird bei der inversen Operation zu $apply$ erkannt

2. Falscher (plötzlicher) Sprung

- X' wird als korrekt erkannt, sig_{dyn} wird erneut berechnet

- 1 Fehlerinjektion mit FAIL*
- 2 Wiederholung Software-TMR
- 3 Eliminierung von Bruchstellen in TMR
- 4 Aufgabenstellung**



Aufgabe

- Absichern des Voters per EAN mittels CoRed-Ansatz
 - Absichern des Akzeptanzmaskierers am Eingang
-
- Jedes Replikat hat genau einen Ausgabewert (integer \mapsto enc_t): eine codierte Prüfsumme des Ergebnisses
 - ↪ Festlegung für jeden der drei Ausgabewerte (X', Y', Z') jeweils *unterschiedliche* aber *konstante* Signatur (SIG_X, SIG_Y, SIG_Z)
 - Nutzung des nächstgrößeren Datentyps X für den ursprünglichen Wert X'
 - ↪ Wahl einer Zahl A mit möglichst großem Hamming-Abstand unter *Vermeidung* möglicher *Überläufe bei der Codierung*



Für jede Operation zwischen zwei codierten Werten

eigene Operation mit konstanten Signaturwerten notwendig!

- Bestenliste wird automatisch auf Webseite veröffentlicht
- VEZS-Shootout Wettbewerb
 - Wer hat den robustesten Code?
 - Vergleich der Repositories
- Wie werden Resultate generiert?



Repository Zugriff

The screenshot shows a web browser window displaying the GitLab interface for a repository named 'Flansch / awesome-vezs'. The browser's address bar shows the URL 'https://gitlab.cs.fau.de/flansch/awesome-vezs'. On the left, a dark sidebar contains navigation options like 'Project', 'Activity', 'Files', 'Commits', 'Graphs', 'Milestones', 'Issues', 'Merge Requests', 'Members', 'Labels', 'Wiki', 'Forks', 'Settings', and the user profile 'Flansch'. The main content area shows the repository name 'awesome-vezs' with a lock icon, indicating it is private. Below the name, the SSH URL 'git@gitlab.cs.fau.de:Flansch/awesome-vezs.g' is displayed and highlighted with a red circle. Other repository statistics include '13 commits', '1 branch', '0 tags', and '0.64 MB'. A message at the bottom states 'This project does not have a README yet'.

Repository URL mit Gruppe per Mail an Übungsleiter

The screenshot shows the 'Members' tab within the 'Settings' section of a GitLab project. The breadcrumb navigation is 'Project > Repository > Issues > Merge Requests > Pipelines > Wiki > Snippets > Settings'. Under 'Settings', the sub-tabs are 'General', 'Members', 'Integrations', 'Repository', and 'CI/CD Pipelines'. The 'Members' section is active, showing the heading 'Members' and the instruction 'Add a new member to vezs17-vorgabe'. A search input field contains 'i4EZS'. Below the search field is the text 'Search for members by name, username, or email, or invite new ones using their email address.' A dropdown menu is set to 'Reporter'. Below this is a link 'Read more about role permissions'. An 'Expiration date' field is present with the text 'On this date, the member(s) will automatically lose access to this project.' At the bottom of the form are two buttons: 'Add to project' (green) and 'Import' (white). Below the buttons is the text 'Existing members and groups'.

■ i4ezs als Benutzer hinzufügen

- Lesezugriff: Reporter
- Settings → Members





Forin.

Vital coded microprocessor principles and application for various transit systems.

IFA-GCCT, pages 79–84, 1989.

