

Verlässliche Echtzeitsysteme

Übungen zur Vorlesung

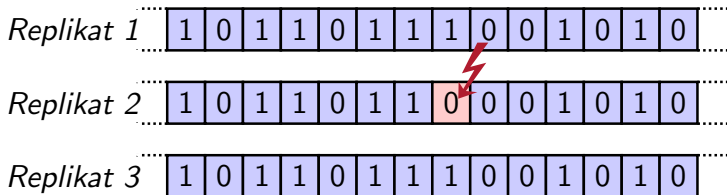
Aufgabe 3: TMR

Phillip Raffeck, Simon Schuster, Peter Wägemann

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

Wintersemester 2021

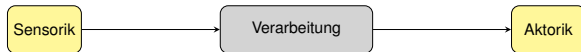


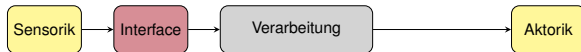


- **Wie viele Replikate** benötigt man zur Fehlermaskierung?
- Arten des Fehlverhaltens (von n Replikaten sind f fehlerhaft)
 1. fail-silent \rightarrow Anzahl Replikate: $n = f + 1$
 2. fail-consistent \rightarrow Anzahl Replikate: $n = 2f + 1$
 3. malicious \rightarrow Anzahl Replikate: $n = 3f + 1$, böartige verteilte Systeme

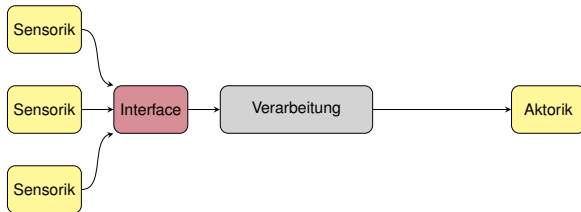


Triple Modular Redundancy

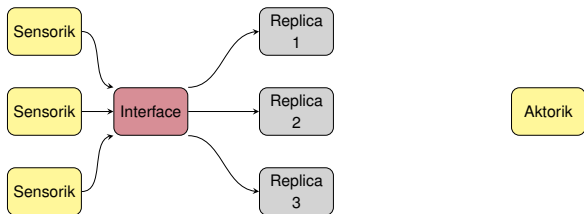




- Schnittstelle sammelt Eingangsdaten (Replikdeterminismus)

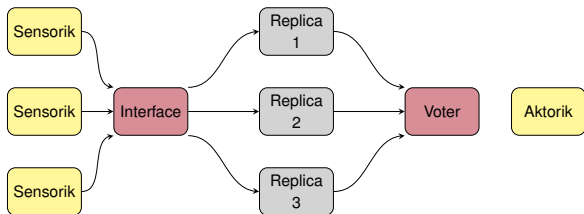


- Schnittstelle sammelt Eingangsdaten (Replikdeterminismus)
- Insbesondere: Mögl. Mehrheitsentscheid für redundante Sensordaten



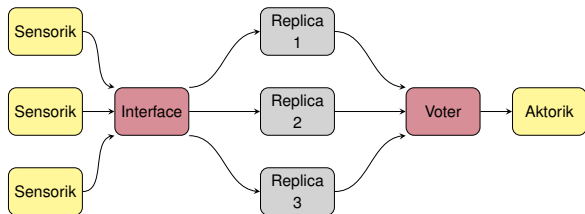
- Schnittstelle sammelt Eingangsdaten (Replikdeterminismus)
- Insbesondere: Mögl. Mehrheitsentscheid für redundante Sensordaten
- Verteilt Daten und aktiviert Replikate

Triple Modular Redundancy



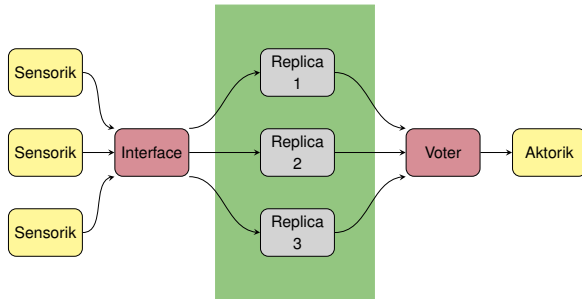
- Schnittstelle sammelt Eingangsdaten (Replikdeterminismus)
- Insbesondere: Mögl. Mehrheitsentscheid für redundante Sensordaten
- Verteilt Daten und aktiviert Replikate
- Mehrheitsentscheider (Voter) wählt Ergebnis





- Schnittstelle sammelt Eingangsdaten (Replikdeterminismus)
- Insbesondere: Mögl. Mehrheitsentscheid für redundante Sensordaten
- Verteilt Daten und aktiviert Replikate
- Mehrheitsentscheider (Voter) wählt Ergebnis
- Ergebnis wird an Aktuator versendet

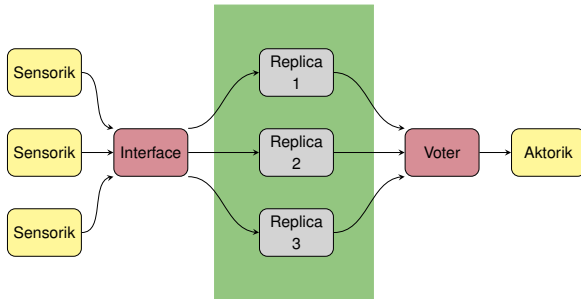
Triple Modular Redundancy



Redundanzbereich

Ausschließlich Replikatausführung





Redundanzbereich

Ausschließlich Replikatausführung

- Mehrheitsentscheid über Berechnungsergebnisse
- Erweiterung der Ausgangsseite mit Informationsredundanz



Replikdeterminismus

Replikat 1

```
void repl_1(void *p){  
    ticks_t time =  
        ezs_get_time();  
  
    ...  
}
```

Replikat 2

```
void repl_2(void *p){  
    ticks_t time =  
        ezs_get_time();  
  
    ...  
}
```

Replikat 3

```
void repl_3(void *p){  
    ticks_t time =  
        ezs_get_time();  
  
    ...  
}
```



Replikat 1

```
void repl_1(void *p){  
    ticks_t time =  
        ezs_get_time();  
  
    ...  
}
```

Replikat 2

```
void repl_2(void *p){  
    ticks_t time =  
        ezs_get_time();  
  
    ...  
}
```

Replikat 3

```
void repl_3(void *p){  
    ticks_t time =  
        ezs_get_time();  
  
    ...  
}
```

Sicherstellung Replikdeterminismus

- Globale diskrete Zeitbasis
- Einigung über Eingabewerte
- Statische Kontrollstruktur der Replikate
- Deterministische Algorithmen



Replikat 1

```
void repl_1(void *p){  
    ticks_t time =  
        ezs_get_time();  
  
    ...  
}
```

Replikat 2

```
void repl_2(void *p){  
    ticks_t time =  
        ezs_get_time();  
  
    ...  
}
```

Replikat 3

```
void repl_3(void *p){  
    ticks_t time =  
        ezs_get_time();  
  
    ...  
}
```

Sicherstellung Replikdeterminismus

Single Core

- Globale diskrete Zeitbasis
- Einigung über Eingabewerte
- Statische Kontrollstruktur der Replikate
- Deterministische Algorithmen



Replikdeterminismus

Replikat 1

```
void repl_1(void *p){  
    ticks_t time =  
        ezs_get_time();  
  
    ...  
}
```

Replikat 2

```
void repl_2(void *p){  
    ticks_t time =  
        ezs_get_time();  
  
    ...  
}
```

Replikat 3

```
void repl_3(void *p){  
    ticks_t time =  
        ezs_get_time();  
  
    ...  
}
```

Sicherstellung Replikdeterminismus

Single Core

- Globale diskrete Zeitbasis
- Einigung über Eingabewerte
- Statische Kontrollstruktur der Replikate
- Deterministische Algorithmen

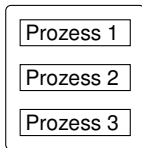
Sicherstellung Systemverhalten

☞ Replikate müssen *innerhalb bestimmter Zeitspanne* terminieren

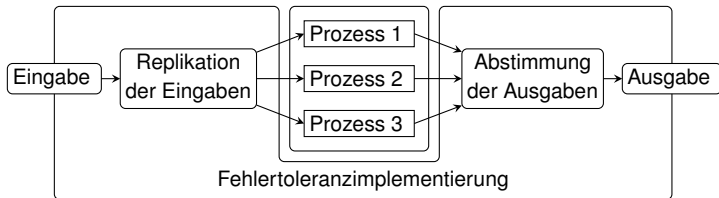




Process-Level Redundancy



Process-Level Redundancy



Process-Level Redundancy

