Seminar: Pleiten, Pech und Pannen der Informatik
Prof. Bernd Becker und Dr. Jürgen Ruf
Sommersemester 2002

Betriebsfehler: Fehlertoleranz und Redundanz

Author: Dirk Spöri
E-Mail: spoeri@informatik.uni-freiburg.de

Inhaltsverzeichnis:

EINLEITUNG 2

Absturz eine Lauda Air 1991 2

Anforderungen an sicherheitskritische Komponenten (Flugsteuersystem) 2

Von Flugzeugen zur Theorie 3

GRUNDBEGRIFFE 4

Was sind Betriebsfehler? 4

Was ist Fehlertoleranz? 4

Was ist Fehlerredundanz? 5
Redundanzstufen 5
Redundanzmaßnahmen 6

FEHLERBEHANDLUNG MODULARER SYSTEME 7

Modulinterne Fehlerbehandlung 7
Redundanz in Speicherbausteinen 7
Error Correcting Codes 8
Hamming-Code 8
Beispiel an einem Speicherbaustein 8
Fehlerbehandlung in Speichersystemen 10
RAID 10

Modulexterne Fehlerbehandlung - Designs 11
Asynchrones Design 12

BEISPIEL AIRBUS A320 13

QUELLEN 15

Einleitung

Absturz eine Lauda Air 1991

Im Januar 1991 stürtzte eine Boeing 767 der Lauda Air ab. Die Katastrophe lief folgendermaßen ab:

- Sonntag, 26. Mai 1991-

23:05 Uhr Bangkok Zeit: Die Boeing 767-300ER, aus Hongkong kommend, hob in Bangkok ab. Alles läuft normal, das Flugzeug ist auf 7000 m gestiegen und stieg weiter.

23:16 Uhr: Eine *beratende* Warnleuchte (die niedrigste von drei Warnstufen) begann zu blinken. Der Copilot schaute auf die Checkliste und las vor: „ein anderer Fehler kann die Schubumkehr auslösen. Die Schubumkehr schaltet sich normalerweise nach der Landung ein. Keine sofortige Reaktion notwendig."

23:18 Uhr: wieder der Copilot Turner: „Sie ist eingeschaltet" (mit Sie war die Schubumkehr gemeint, die während des Steiflug, also bei hoher Motorleistung, ohne Zutun der Piloten aktiviert wurde).
Einige Momente später hört auf dem Voice-Rekorder eine akkustische Warnung, daß die Fluglage instabil sei. Zwei Sekunden später endet das Band, das Flugzeug stürtzte ab.

Zusammengefasst die für das Verständnis der Katastrophe benötigten Fakten:

- Die Schubumkehr der Boeing 767 schaltete sich ohne Zutun der Piloten automatisch ein, was zum Auseinanderbrechen und Absturz des Flugzeuges über Thailand und zu 223 Toten führte.
- Nach Boeing-Angaben hätte eine Boeing 767 trotz Schubumkehr weiterfliegen können...
- Die Einschaltung der Schubumkehr sollte während des Fluges unmöglich sein:
  - ein mechanisches System verhindert die Aktivierung der Schubumkehr während des Fluges
  Vermutlich ist dieses System ausgefallen, woraufhin ein Computerfehler die Schubumkehr auslöste.
- Kurz zur Aktivierung der Schubumkehr:
  Die Aktivierung erfolgt über einen 28 Volt-Ausgang aus einem AND-Gatter mit 4 Eingängen.
  Eine der Eingangsleitungen kommt vom „FADEC", einer Kombination aus „Maschinen im Leerlauf" und „Gewicht auf Rädern".
  Ein Input kommt direkt vom Schubumkehrsschalter.
  Zwei weitere Inputs sind jeweils Mikroschalter, die angeben, ob die Klappen ausgefahren sind.
  Diese Systeme sind nicht mehrfach ausgelegt, weder als ganzes, d.h. mehrere AND-Gatter, noch als Teile, also z.B. mehrere „FADECs".

Diese Katastrophe stellt ein Beispiel dar, wie das Versagen mehrerer Systeme zum Ausfall des Gesamtsystems führen kann. Das Versagen der einzelnen Systeme ließe sich sicher auch auf einzelne Fehler wie Materialverschleiß oder Fertigungsfehler zurückführen. Doch die Fehlerfreiheit aller Einzelsysteme läßt sich nie garantieren.

Anforderungen an sicherheitskritische Komponenten (Flugsteuersystem)

Im Flugzeugbereich wird die Ausfallwahrscheinlichkeit sicherheitskritischer Komponenten angegeben mit weniger als 10^{-9} / Flugstunde. Das bedeutet, daß eine sicherheitskritische Komponente ein Mal in einer Milliarde Flugstunden ausfallen kann.
Ein Milliarde Flugstunden sind schnell erreicht, wie folgende Rechnung zeigt:

Beispielrechnung:

- Bei 2000 Flugstunden im Jahr pro Flugzeug,
- 25 Jahre Dienstzeit pro Flugzeug,
- 1000 ausgelieferte Flugzeuge:

  Summe: 50 Millionen Stunden Flugzeit für die gesamte Flotte.

Die Wahrscheinlichkeit für den Ausfall einer sicherheitskritische Komponente beträgt weniger als 5%, was aber immer noch viel ist. Selbstverständlicherweise besteht deshalb der Anspruch, daß das Gesamtsystem bei Ausfall einer sicherheitskritischen Komponente nicht unvorhergesen ausfällt

Von Flugzeugen zur Theorie

Nicht nur in der Luftfahrt, sondern auch in anderen Bereichen, zum Beispiel in der Raumfahrt oder in der Kraftwerkstechnik, wird mit einer zunehmenden Computerisierung auch die Sicherheit der Systeme wichtiger.
Aufgrund der Komplexität, die bei den aufgezählten Bereichen offensichtich ist, lassen sich niemals alle Fehler im voraus identifizieren und verhindern. Somit ist es nötig, Systeme zu entwickeln, die auch mit unverhergesehenen Fehlern zurechtkommen, damit diese Systeme nicht in unvorhersehbare Zustände geraten.
Dabei spielen Methoden der Fehlertoleranz und Fehlerredundanz eine große Rolle, wobei bestimmte Konzepte z.B. der fehlerkorrigierenden Codes besonders häufig zur Anwendung kommen.
Diese Arbeit soll in die Grundbefriffe der Tolenranz und Redundanz einführen und anschließend konkrete Lösungen bei Speichern, Speichersystemen und modularen Systemen aufzeigen.

Um Lösungen für Probleme zu finden, ist es wichtig, nach den richtigen Dingen zu fragen. Die naheliegensten Fragen im Umgang mit Fehlern sind wohl: „Was sind Betriebsfeler?“, „Wo treten Fehler auf?“ und „Welche Auswirkungen haben Betriebsfehler?“.

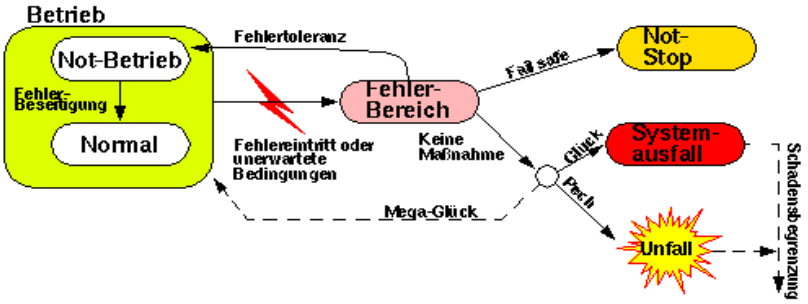
Grundbegriffe

Was sind Betriebsfehler?

Unter Betriebsfehlern versteht man ganz einfach Fehler, die im laufenden Betrieb eines Systems auftreten, egal ob sie vorhersehbar sind oder nicht, egal ob sie aufgrund äußerer Einwirkung entstehen oder aufgrund von Verschleiß.
Folgende Tabelle soll einen anschaulichen Überblick bieten:

Betriebsfehler
- zufällige, physikalische Fehler
- Verschleißfehler, z.B. durch Alterung elektrischer Bauteile
- Störungsbedingte Fehler aufgrund äußerer physikalischer Einflüsse
- Bedienungsfehler
- Wartungsfehler: fehlerhafte Systemeingriffe während Wartungsintervall
- Sabotage, Vandalismus

Der Anspruch an unsere Methoden imUmgang mit Fehlern ist es nun, mit allen diesen Fehlern irgendwie zurechtzukommen. Betrachten wir nun die anderen beiden oben genannten Fragen: Fehler können eine einzelne Komponente eines Systems betreffen, z.B. den Speicher, oder aber auch zum Ausfall des gesamten Computers oder Computersystems führen. Der Umgang mit Fehlern muß somit auch auf beiden Ebenen stattfinden, aber folgendes Schema zur Auswirkung von Fehlern läßt sich im kleinen wie im großen veranschaulichen:



Unter der Annahme des Auftretens eines Fehlers sind verschiedene Konsequenzen vom Zusammenbrechen über ein geregeltes Abschalten oder eingeschränktes Weiterlaufen bis zur Aufrechterhaltung des Normalbetriebes möglich. Hiermit sind wir schon bei einer Definition der Fehlertoleranz:

Was ist Fehlertoleranz?

Definition Fehlertoleranz:
Die Fähigkeit eines Systems, auch mit einer begrenzten Zahl fehlerhafter Subsysteme seine spezifizierte Funktion zu erfüllen.

Stufen der Fehlertoleranz	
Typ	**Verhalten des Systems**
(go)	System- und Anwendungsprogramm sicher und korrekt
(fail-operational)	System fehlertolerant ohne Leistungsverminderung
(fail-soft)	Systembetrieb sicher, aber Leistung vermindert
(fail-save)	Nur Systemsicherheit gewährleistet
(fail-unsave)	unverhersehbares Systemverhalten

Bei der Korrektur von Fehlern unterscheidet man die zwei Prinzipien der Vorwärts- und der Rückwärtsfehlerkorrektur.
Bei der **Vorwärtsfehlerkorrektur** versucht das System, so weiterzumachen als ob kein Fehler aufgetreten wäre, indem es z.B. fehlerhafte Inputs durch Erfahrungswerte aus der Vergangenheit oder durch korrekt funktionierende Inputs ausgleicht oder im Moment des Auftretens eines Fehlers sofort mit korrekt funktionierenden Ersatzsystemen weiterarbeitet. Fehler bleiben bei der Vorwärtsfehlerkorrektur normalerweise unsichtbar für Anwender.
Bei der Rückwärtsfehlerkorrektur versucht das System bei Auftreten eines Fehlers in einen Zustand vor diesem Auftreten zurückzukehren, z.B. in den Zustand direkt vor einer fehlerhaften Berechnung, um diese Berechnung erneut auszuführen. Genauso ist aber auch ein Zustandswechsel in einen Notbetrieb oder z.B. ein Reboot des Systems möglich. Kann eine fehlerhafte Berechnung erfolgreich wiederholt werden, bleibt auch bei der Rückwärtsfehlerkorrektur der Fehler für den Anwender unsichtbar. Oft ist aber nur ein Weiterbetrieb mit Leistungseinbußen oder eingeschränkter Funktionalität möglich und der Fehler somit sichtbar.

Was ist Fehlerredundanz?

Das häufigste Konzept, um Systeme gegenüber Betriebsfehler toleranz zu machen, ist die Fehlerredundanz.

> **Definition Redundanz:**
> Vorhandensein von mehr als für die Ausführung der vorgesehenen Aufgaben an sich notwendigen Mittel.
> *(nach DIN 40041, Teil 4)*

Man unterscheidet dabei verschiedene Stufen der Redundanz.

Redundanzstufen

Aktive-Redundanz (funktionsbeteiligte, heiße Redundanz): Mehrere Komponenten eines Systems führen dieselbe Funktion simultan aus. Fällt eine Komponente aus, wird dieser Fehler durch die verbleibenden Komponenten direkt kompensiert und führt daher nicht unmittelbar zu einer von außen erkennbaren Reaktion.
Standby-Redundanz (passive Redundanz): Zusätzliche Mittel sind eingeschaltet/bereitgestellt, werden aber erst bei Ausfall/Störung an der Ausführung der vorgesehenen Aufgabe beteiligt.
Kalte Redundanz: Zusätzliche Mittel werden erst bei Ausfall/Störung eingeschaltet/bereitgestellt.

Statt dieser Stufen trifft man auch häufig die Unterscheidung zwischen *statischer* und *dynamischer* Redundanz. Statisch: es gibt zwei redundante Systeme, fallen beide aus, läuft

nix mehr; Dynamisch: bei Auftreten eines Fehlers wird versucht zusätzliche Systeme bereitzustellen.

Redundanzmaßnahmen

Redundanz läßt sich auf verschiedenen Ebenen und mit verschiedenen Maßnahmen erreichen. **Strukturelle Redundanz** meint die Erweiterung des Systems um Objekte wie zusätzliche - gleichartige Rechner, - Baugruppen, - Speicher Komponenten. **Funktionelle Redundanz** ist ganz einfach die Erweiterung des Systems um zusätzliche Funktionen. Eine Anmerkung speziell zum Begriff **Software Redundanz**: Strukturelle und funktionelle Redundanz stehen in einem engen Zusammenhang. Eine strukturelle Redundanz auf einer tieferen Betrachtungsebene des Systems kann sich andererseits als funktionelle Redundanz in der nächsthöheren Betrachtungsebene äußern. Weiters wird immer einer redundanten Hardwarekomponente auch ein zusätzliches Managementmodul (d.h. Firmware/Software) vonnöten sein. Eine Einteilung in Hardware/Software Redundanz ist daher kaum sinnvoll. Hinter **Informationsredundanz** verbergen sich z.B. zusätzliche Bitpositionen wie Prüfbits, Polynombildung usw. und **Zeitredundanz** ist z.B. die Wiederholung der fehlgeschlagenen Operation.
Mit diesem Überblick über verschiedenen Maßnahmen sind wir schom wieder zu einer Frage des Anfangs zurückgekehrt: abhängig von dem Ort des Auftretens von Fehlern drängen sich unterschiedliche Redundanzmaßnahmen auf.

Fehlerbehandlung modularer Systeme

In einem modularen System, das heißt, einem zusammengesetzten System mit verschieden Komponenten wie z.B. Prozessoren, I/O-Geräten, Sensoren, ... können Fehler in den einzelnen Komponeten auftreten die zu einem fehlerhaften Verhalten des gesamten Systems führen. Dabei verbreiten sich Anomalien in aktiven Komponenten wie Prozessoren durch Buszugriffe, Schreibzugriffe auf Speicher usw. besonders schnell im System.
Leider sind Anomalien gerade in Prozessoren (neben mechanischen Ausfällen) aufgrund der Komplexität von Prozessoren besonders wahrscheinlich und leider sind dort auch die meisten systematischen Fehler (z.B. Entwicklungsfehler). Systematische, also in allen Komponenten vom selben oder ähnlichen Typ auftretende Fehler sind besonders schwer durch Redundanz zu korrigieren. Was bringt es, drei Mal die selbe CPU mit dem selben Fehler die selbe Berechnung machen zu lassen?

Im folgenden soll Fehlerbehandlung zu erst innerhalb von Modulen bestimmten Typs, danach im modularen System als ganzes dargestellt werden.

Modulinterne Fehlerbehandlung

Drei wichtige Komponenten, die eine modulinterne Fehlerbehandlung berücksichtigen muß sind der schon erwähnte Prozessor (meistens in Form eines Microcontrollers, der z.B. im Falle eines Absturtzes den Prozessor resettet), das Bussystem und der Speicher. In dieser Arbeit wird nur letzterer genauer beschrieben.

Redundanz in Speicherbausteinen

Das Stichwort in der Fehlertoleranz von Speicherbausteinen sind **Redundante Codes**. Die Redundanz ist hierbei Informationsredundanz, d.h. innerhalb von Codeworten wird die darin enthaltenen Information mehrfach codiert.

Das Konzept redundanter Codes basiert darauf, daß die Menge der gültigen Codeworte nur eine Teilmenge T der Menge M aller möglichen Bitkombinationen bildet.
Je größer die Redundanz des Codes ist, desto kleiner ist die Mächtigkeit der Teilmenge T im Verhältnis zur Mächtigkeit der Menge M.
Entsprechend steigt die Wahrscheinlichkeit dafür, dass ein fehlerhaftes Codewort nicht wieder Element der Teilmenge T ist, so dass bei einer Prüfung der Fehler entdeckt werden kann.
Konkret: Eine Folge von n Bits kann im Prinzip 2^n verschiedene Nachrichten übermitteln, denn so viele verschiedene Möglichkeiten gibt es, n Nullen oder Einsen hintereinander zu schreiben. Wenn man aber Fehler erkennen oder gar korrigieren will, kann nicht jede dieser Bitfolgen eine Nachricht sein.
Erklären wir eine gewisse Folge von Bits zum Codewort, das heißt zu einer bedeutungstragenden Nachricht. Dann ist jede Bitfolge, die sich von dieser nur an einer Stelle unterscheidet, als Codewort nicht mehr brauchbar, denn sie könnte ja, für den Empfänger unentdeckbar, durch Verfälschung dieses Bits aus dem ursprünglichen Codewort hervorgegangen sein. Nennen wir sie ein Fehlerwort zu diesem Codewort. Wenn man Fehler auch korrigieren will, dürfen zwei Codewörter kein Fehlerwort gemeinsam haben. Nur dann kann nämlich der Empfänger ein Fehlerwort nicht nur als solches erkennen, sondern daraus auch das – dann eindeutige – richtige Codewort wiederherstellen.
Zu jedem Codewort gibt es n Fehlerwörter, eins für jede Bitposition; die Fehlerwörter sind also n-fach in der Überzahl. Von dem Vorrat der insgesamt 2^n verschiedenen Bitfolgen der

Länge n verbraucht jedes Codewort n+1 Stück, eins für sich selbst und n Stück für seine Fehlerwörter.
Wenn n+1 selbst eine Zweierpotenz ist, können die Codewörter samt ihrem "Hofstaat" an Fehlerwörtern den Vorrat der 2^n Wörter restlos aufbrauchen; daher kommt es, dass die Fälle n=2^k–1 besonders günstig auskommen.

Error Correcting Codes

Das Konzept, mit dem diese Redundanz in der Praxis erreicht wird, ist das Hinzufügen von Prüfbits zu Speicherwörtern, die die Erkennung und Korrektur von Fehlern ermögliche. Die einfachste Form ist das Parity-Bit, das angibt, ob ein Wort eine gerade oder ungerade Anzahl an Einsen enthält. Mit dem Parity-Bit ist nur die Erkennung, nicht die Korrektur eines einzelnen Bitfehlers pro Wort möglich.
Fortgeschrittenere Methoden redundanter Codes sind der weit verbreitete Hamming-Code, zyklische Codes, arithmetische Codes oder Prüfsummen.

Hamming-Code

Die Bits des Codewortes werden mit 1 beginnend von links nach rechts durchnummeriert. Jene Bits, die Potenzen von 2 sind, also 1, 2, 4, 8 usw. sind Prüfbits. Die restlichen (3, 5, 6, 7, 9,... usw.) sind mit den Information tragenden Datenbits gefüllt.
Jedes Prüfbit ist ein Parity-Bit für eine bestimmte Menge von Bits. Ein Bit kann in Berechnungen verschiedener Prüfbits involviert sein. Um festzustellen, zu welchen Prüfbits ein bestimmtes Bit, nämlich das mit der Nummer k, einen Beitrag liefert, genügt es, k als Summe von Zweierpotenzen zu schreiben. Zum Beispiel 11 = 8 + 2 + 1 oder 29 = 16 + 8 + 4 + 1. (Mit anderen Worten: Die Zahl k wird in das Binärsystem umgerechnet).

Ein Bit liefert zu allen jenen Prüfbits einen Beitrag, deren Nummer in dieser Darstellung vorkommt.
So wird etwa Bit 11 in den Berechnungen der Prüfbits 1, 2 und 8 berücksichtigt. Um nun ein Prüfbit zu ermitteln, werden alle Datenbits, die bei diesem Prüfbit zu berücksichtigen sind, mittels Exklusiv-Oder verknüpft.

Beispiel: Hammingcode der aus 4 Datenbits und aus 3 Prüfbits besteht:

Gleichung für das Prüfbit 1: P1 = x3 + x5 + x7

Gleichung für das Prüfbit 2: P2 = x3 + x6 + x7

Gleichung für das Prüfbit 4: P4 = x5 + x6 + x7

Die kleinste Distanz zwischen zwei beliebigen (gültigen) Wörtern eines Codes wird **Hamming-Abstand d** genannt.
Diese kleinste Distanz ergibt sich aus der Anzahl der Binärstellen, die mindestens verfälscht werden müssen, um ein gültiges Codewort in ein anderes gültiges Codewort zu überführen.

Anzahl erkennbarer Fehler: d-1

Anzahl korrigierbarer Fehler:(d-1)/2 für ungerades d bzw (d-2)/2 für gerades d

Beispiel an einem Speicherbaustein

Wir nehmen im folgenden einen Hamming-Code mit Abstand d=3 an.

Ein Speicher habe W Worte mit höchstens N Bitfehlern. Der IC ist dann fehlerfrei, wenn die N Bitfehler alle auf unterschiedliche Worte fallen (für einen ECC der 1 Bitfehler pro Wort korrigiert), das hat die Wahrscheinlichkeit $P_{Korrigierbar:}$

$$P_K(N) = \prod_{i=0}^{N-1}\left(\frac{W-i}{W}\right) = \frac{W!}{W^N(W-N)!}$$

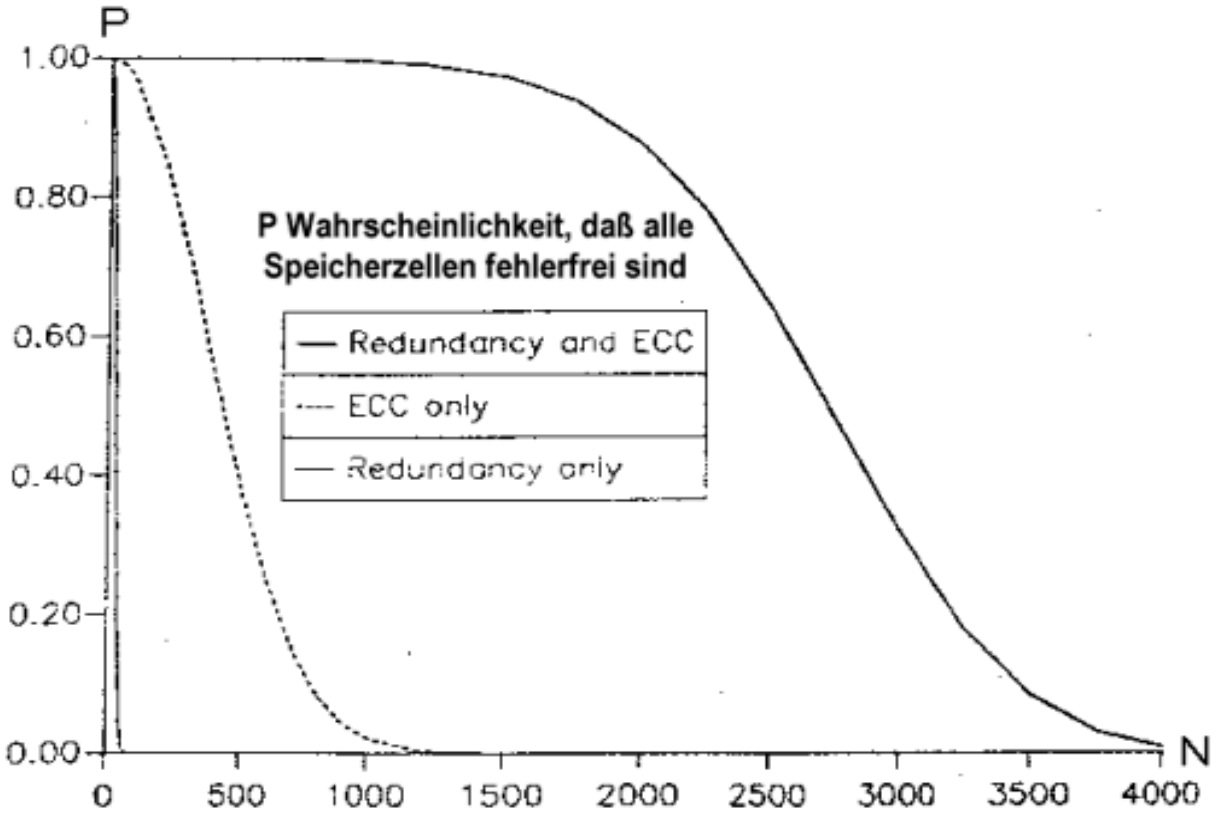
Die Fehlerwahrscheinlichkeit

$$\mathbf{P_{Error}\,(N) = 1 - P_K(N)}$$

steigt rasch mit zunehmender Anzahl der Fehler.

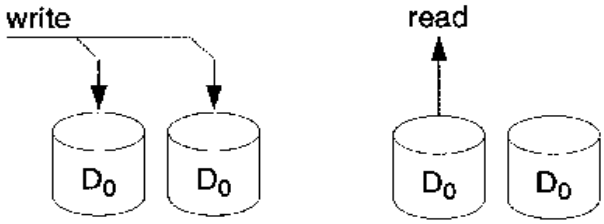
In unserem Beispiel nehmen wir einen 16 MBit DRAM mit Wörtern zu je 137 Bit (128 Bit, dazu 9(?) Prüfbits), also W = 131072; Weiterhin nehmen wir N = 200 Bitfehler an. Mit unserem Hamming-Code ist der Speicherbaustein genau dann fehlerkorrigierend, wenn in keinem Wort mehr als 1 Bitfehler auftritt.
Die Wahrscheinlichkeit dafür, daß das in zumindest einer Speicherzelle doch vorkommt beträgt $\mathbf{P_{Error} \approx 0{,}2}$.

Eine noch größere Verbesserung ist durch die zusätzliche redundante Auslegung von Bitleitungen erreichbar. Damit ist gemeint, daß jede Bitleitung (jede „Reihe“ an Bits) doppelt vorhanden ist. Eine Redundanz ohne fehlerkorrigierenden Code bringt nichts, da im Vergleich der beiden Bitleitungen nur erkennbar ist, wenn eine der beiden fehlerhaft ist, nicht jedoch, welche Information richtig ist.
In Kombination mit einem fehlerkorrigierenden Code steigt die Fehlertoleranz jedoch enorm, was folgende Grafik, wieder auf oben genannten Speicherbaustein bezogen, veranschaulicht:

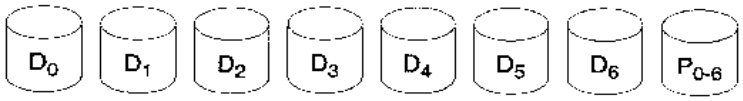


Fehlerbehandlung in Speichersystemen

In Speichersystemen, z.B. Festplatten oder Festplattencluster kommen vor allem zwei Konzepte, oftmals in Kombination, zur Anwendung. Beim **Mirroring** (= Spiegelung) werden einfach alle Daten bei jedem Schreibzugriff doppelt abgelegt, und zwar auf zwei verschiedenen Laufwerken.



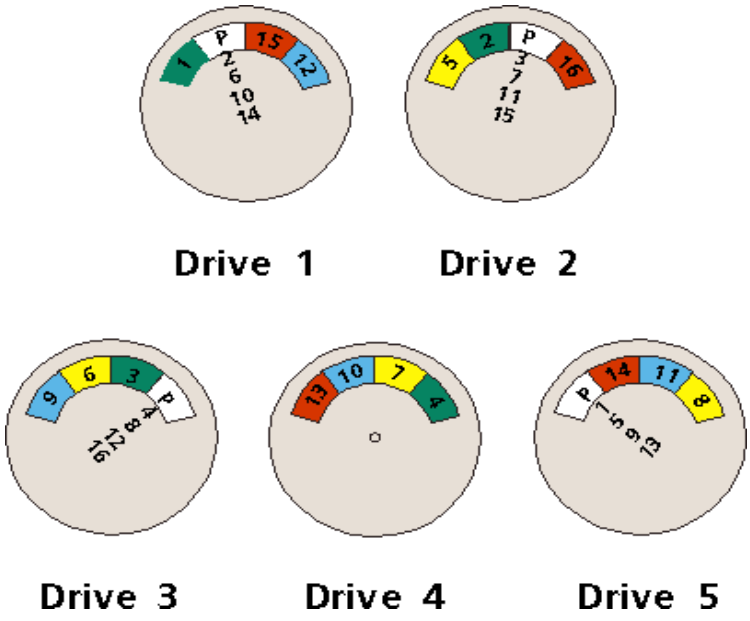
Bei Parity werden bei Schreibzugriffen in einem extra Parity-Bereich Prüfbits über alle anderen Laufwerke angelegt.



$$P_{0\text{-}6} = D_0 \oplus D_1 \oplus D_2 \oplus D_3 \oplus D_4 \oplus D_5 \oplus D_6$$

RAID

Das wohl bekannteste Design zur Implementierung von Mirroring und Parity bei Speichersystemen ist RAID. RAID steht für **Redundant Array of Inexpensive Disks**. Verschiedene "RAID-Level" beschreiben dabei Systeme mit mehreren Massenspeichern, die sich einerseits in **Geschwindigkeit**, andererseits in **Sicherheit**, oder in beidem auszeichnen.
Sicherheit wird durch die beiden Prinzipien **Mirroring** und Error Correcting Codes (**Parity)** erreicht.
Einer der verbreitesten RAIDs ist RAID Level 5, der im folgenden Schema dargestellt ist:



Der Ausfall einer Platte wird kompensiert aus dem Paritätsstreifen der anderen Platten.

Bei Ausfall z.B. von Platte 2 errechnen sich die fehlenden Daten durch die folgende Gleichung:

$$D_{i.2} = D_{i.1} \quad D_{i.3} \quad D_{i.4} \quad P_i$$

($D_{i.j}$ = Sektor i von Laufwerk j ; P = Paritätsstreifen)

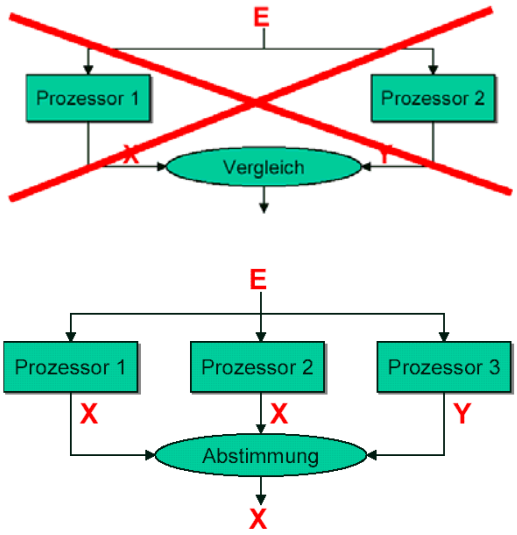
Rekonstruktion der Daten:
Die Rekonstruktion selber erfolgt **als Hintergrundprozeß**, wodurch die Abarbeitung normaler Anwenderzugriffe weiterhin möglich ist.
Die defekte Platte wieder entweder ausgetauscht (= Hot Plug, falls im laufenden Betrieb) oder statt dessen eine schon installierte freie Platte im Array genutzt (Hot Spare).

Aber:
Die hier vorgestellte Methode hilft nicht beim Ausfall von Kontrollern, deren Austausch das System weitaus mehr beinträchtigt.

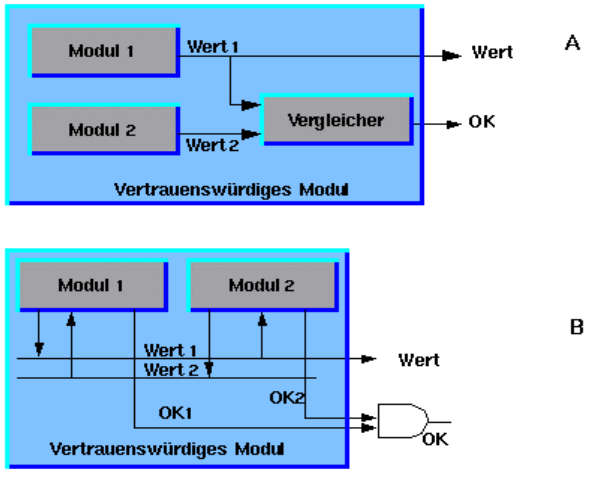
Modulexterne Fehlerbehandlung - Designs

Die modulexterne Fehlerbehandlung, das heißt die Behandlung von Modulausfällen in modularen Systemen, muß zwei Dinge berücksichtigt: als erstes die Möglichkeit, fehlerhaftes Verhalten eines Moduls zu erkennen und als zweites die Korrektur mit Hilfe von Modulredundanz.

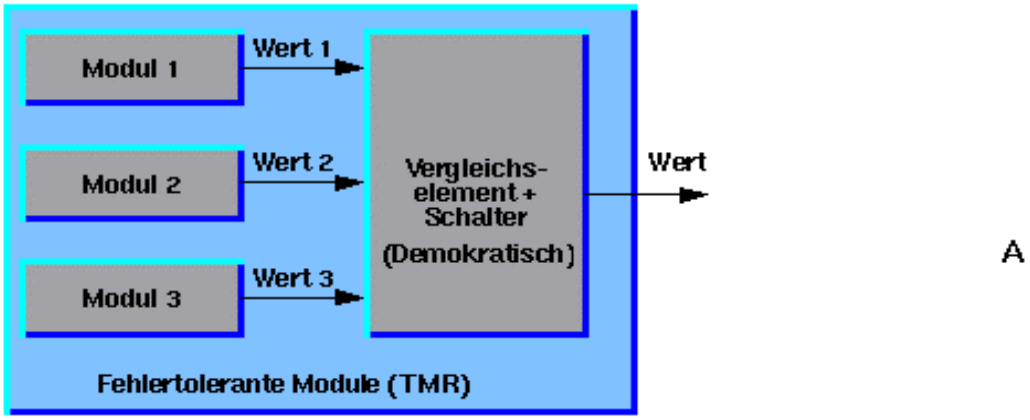
Wie man sich leicht überlegen kann, reicht eine zweifache Redundanz, also eine doppelte Auslegung, nicht aus, um Fehler korrigieren zu können, denn es läßt sich nur eine Abweichung der Ergebnisse der beiden Komponenten feststellen. Das System oder die Komponente gilt trotzdem als ausgefallen. Erst mit einer dreifachen Redundanz verbunden mit einer Mehrheitsabstimmung zwischen den Modulen oder einer Mittelung der Ergebnisse ist ein fehlertolerantes System möglich.



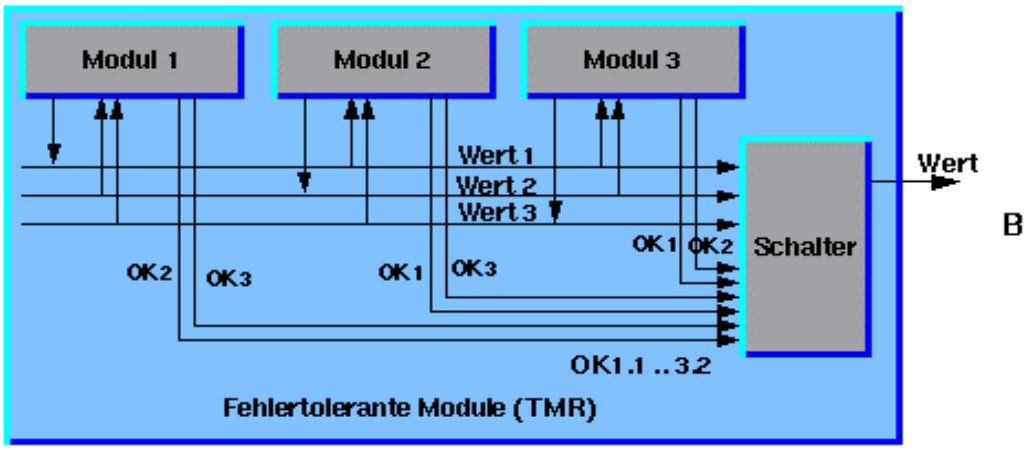
Diese Erkenntnis führte zum Konzept vertrauenswürdiger Module. Ein vertrauenswürdiges Modul ist in sich selber zweifach redundant ausgelegt, womit ein einzelnes Modul zumindest erkennen kann, ob es fehlerfrei arbeitet oder nicht. Die Fehlererkennung läßt sich über verschiedene Wege lösen: z.B. über einen Vergleicher, der die Ergebnisse der beiden Komponenten vergleicht und abhängig davon sein OK gibt, oder darüber, daß beide Komponenten jeweils die Ergebnisse der anderen Komponente mit dem eigenen Ergebnis vergleichen, abhängig davon ein OK geben und diese beiden Oks in ein UND-Gatter gehen. In beiden Fehlern wird das Ergebnis solch eines vertrauenswürdigen Moduls nur verwendet, wenn gleichzeitg ein OK am



OK-Ausgang liegt.
Damit ist nun schon eine fehlertolerante Konstruktion aus zwei vertrauenswürdigen Modulen möglich, die beide ihr OK und ihr Ergebnis an einen Schalter liefern. Damit erhalten wir ein System, das auch bei Ausfall eines vertrauenswürdigen Moduls weiterarbeiten kann.



A



B

Ähnliche wie bei der Konstruktion der vertrauenswürdigen Module mit zweifacher Redundanz läßt sich auch eine dreifache Redundanz oder dementsprechend eine N-Modulare Redundanz erreichen.
Diese dreifach Modulare Redundanz kommt im Gegensatz zur Konstruktion aus vertrauenswürdigen Modulen mit insgesamt nur drei statt vier Komponenten aus (jedes vertrauenswürdige Modul besteht ja in sich aus zwei redundanten Komponenten) um ein fehlertolerantes System zu erreichen.

Asynchrones Design

Was hier beschrieben wurde sind alles Beispiele für ein **asynchrones Design** redundanter modularer Systeme. Mit asynchron ist gemeint, das die einzelnen Module nicht synchron arbeiten, sondern
- jeder Sensor seine Inputs unabhängig sampelt
- alle Rahmenbedigungen und Regeln unabhängig evaluiert werden
- die Ergebnisse oder Befehle an eine Mittler- oder Auswahlkomponente (Voter) geschickt werden

Nun können Sensormessungen der einzelnen Module zu leicht unterschiedlichen Zeitpunkten zu deutlich unterschiedlichen Bewertungen führen, die durch Verarbeitungsregeln möglicherweise nochmals verstärkt werden. Die Auswahlkomponente (Voter) muß nun ein breites Intervall an gültigen Werten akzeptieren

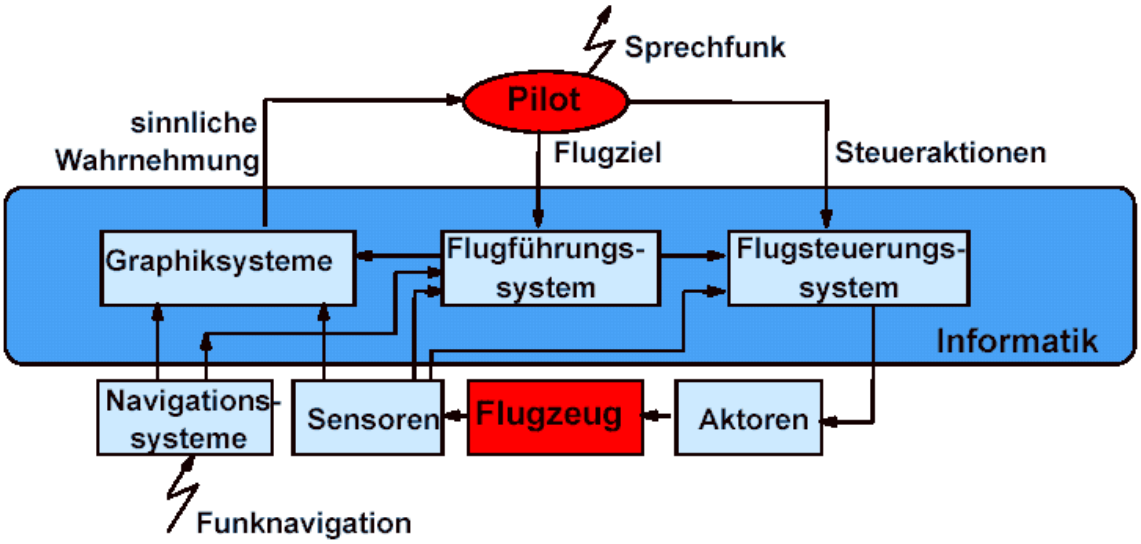
Die Erkennung einer fehlerhaften Komponenten wird erschwert und verzögert

sobald fehlerhafte Inputs ausgeschlossen werden, kann sich ein Mittelwert, sofern er berechnet wird, sprunghaft ändern und somit das System plötzlich von einem Zustand in einen anderen versetzen (z.B. ungünstig für Steuerung eines Flugzeuges)

Die Bewertungen der einzelnen Kanäle können noch stärker divergieren, wenn deren Regeln Entscheidungspunkte beinhalten. Damit ist gemeint, daß in der Verarbeitung von Inputs Zustände z_0, z_1, z_n durchschritten werden. An diesen Zuständen entscheidet sich das System jeweils für einen bestimmten weiteren Verarbeitungsweg, von dem es später nicht mehr abweicht.
Eine Lösung hierfür kann sein, daß an diesen z_0, z_1, z_n jeweils "Mehrheitsentscheidung" der Module über den weiteren Verarbeitungsweg getroffen werden, also eine Art Synchronisierung.

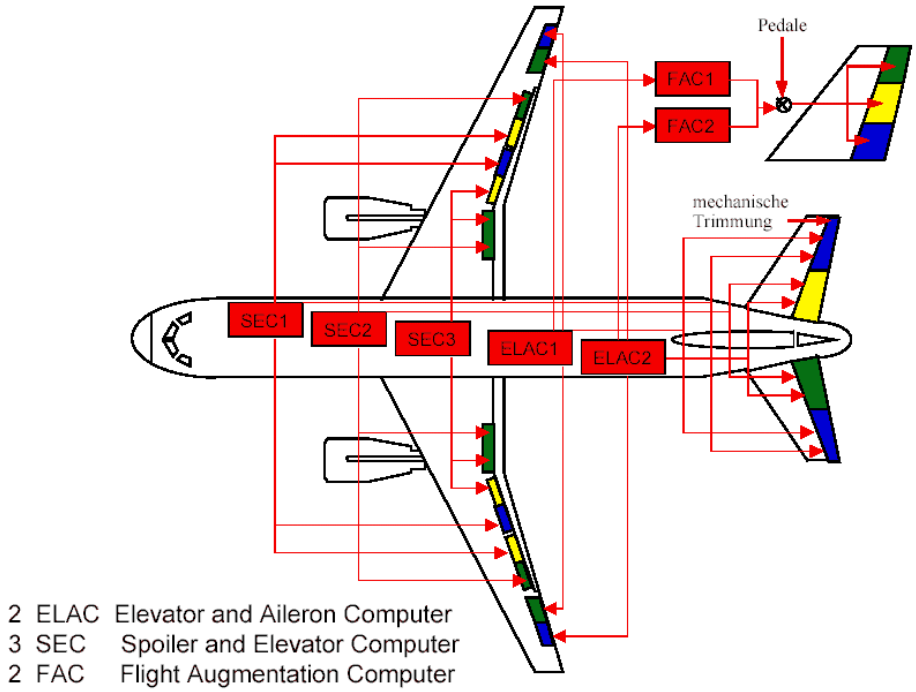
Beispiel Airbus A320

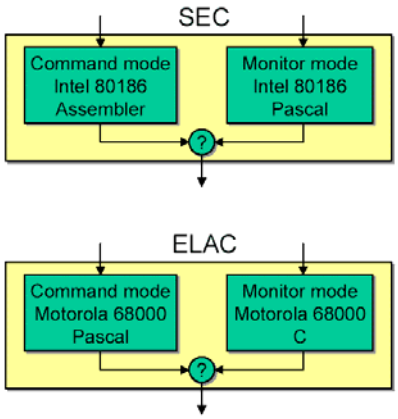
Der Airbus A320 war das erste Verkehrsflugzeug, daß vollkommen computergesteuert fliegt. Die Technologie nennt sich "Fly-by-wire". Alle Sensormessungen und Zustände des Flugzeuges werden in einem Computersystem verarbeitet und dann graphisch aufbereitet an die Piloten weitergegeben. Alle Steuerbefehle werden zusammen mit den Flugdaten vom Computersystem überprüft und nur ausgeführt, wenn sie keine Regeln verletzen und mit den Sensormessungen im Einklang sind.
Die Piloten "sehen" also nicht mehr die Realität, sondern eine Computerabbildung und alle Steuerbefehle der Piloten werden zuerst im Computer überprüft
Hier eine schematische Darstellung:



Ein wichtiger Nutzen der Fly-by-wire-Technologie soll die Vermeidung von Pilotenfehlern durch deren Entlastung sein. Das Computersystem übermittelt z.B. nur noch Probleme an Piloten, die kritisch sind: z.B. Feuer im Triebwerk, im Gepäckraum oder in den Toiletten.
Damit die Sicherheit wirklich erhöht wird, muß das Computersystem selber fehlererkennend und tolerant sein.

Die wichtigen Flugsteuerrungssystem des Airbus A320 sind deshalb redundant ausgelegt:





Jedes dieser einzelnen Systeme ELAC, SEC und FAC ist in sich Fehlererkennd. Dies ist durch eine 2-fach-Redundanz gelöst, wobei die doppelte Auslegung mit unterschiedlicher Hard- und Software gelöst ist, so ist beim SEC das Monitorung-System in Pascal und das ausführende System in Assembler geschrieben. Das SEC läuft auf Intel-Prozessoren, während das ELAC auf Motorola-CPUs läuft.

Leider ist jedoch das Fly-by-Wire-System als ganzes nur rudimentär redundant ausgelegt. Der Fall, daß das gesamte Computersystem ausfällt wurde wohl von den Konstrukteuren für äußerst unwahrscheinlich gehalten. Es gibt kein Ersatz-Computersystem, einzig ein rein mechanisches Backup-System, was sich nur zur Landung im Notfall eignet, aber schwierigen Flugsituationen nicht gewachsen ist.

Quellen

- Klaus Eppele: Erhöhung der Zuverlässigkeit von Rechnersystemen (Datacom 09/91)
  http://www.improve-mtc.de/Veroffentlichungen/Zuverlassigkeit1/zuverlassigkeit1.html
- Praktische Beispiele fehlertoleranter Systeme
  http://www.morawek.at/Arbeiten/Fehlertoleranz/Fehlertoleranz.html
- S. Montenegro: Prinzipien der Fehlertoleranz
  S. Montenegro: Fehlertoleranz und Industrie Computer
  beide über http://www.first.gmd.de
- Forum On Risks To The Public In Computers And Related Systems
  http://catless.ncl.ac.uk/Risks
- Informatik Informatik im Cockpit: Pilot contra Computer
  http://kbs.cs.tu-berlin.de/publications/presentations/He260399.pdf