Feil-tolerante datasystemer: Motivasjon & anvendelser

- **Hvorfor blir pålitelighet i datamaskinsystemer stadig viktigere?**
  - økt avhengighet
  - økt sårbarhet
- **Eks. på anvendelser hvor pålitelighet er spesielt viktig**
  - prosess-kontroll
  - produksjons-kontroll
  - transaksjons-prosessering
  - telefon-sentraler
  - styring av flytrafikk
  - pasient-overvåking
  - børs

Datamaskinsystemer med meget høy tilgjengelighet

- **Kilde**
  - *High-Availability Computer Systems*, Jim Gray, Daniel P. Siewiorek, IEEE Computer, sept 1991, side 39-48
- **"Pålitelighets-paradoks":**
  - Jo større et system er, desto viktigere er det, men desto mindre sannsynlig er det at det har stor grad av tilgjengelighet
- **Utvikling av store, komplekse datasystemer er et lite modent fagfelt**
- **Bruk, omgivelser og programvare er i dag langt oftere kilde til feil enn maskinvare**
- **99% tilgjengelighet høres bra ut men tilsvarer 100 min. nedetid i løpet av en uke**
- **Høy tilgjengelighet, feiltolerant, m.fl.**
  - "ulike grader av samme egenskap"
  - Tabell 1.

Ulike typer av feil

- **Maskinvare**
  - spesifikasjons og konstruksjonsfeil
  - aldring (fysiske spenninger, slitasje, korrosjon,...)
  - kosmisk stråling
- **Programvare (typisk 1 pr. 1000 linjer !)**
  - spesifikasjons og konstruksjonsfeil
  - verktøy-feil (kompilator)
- **Operatørfeil**
- **Ytre påvirkning**
  - feil i strømforsyning
  - temperatursvingninger
  - brann, sabotasje, m.m.
- **I tillegg (ikke type av feil) : nedetid p.g.a.**
  - oppgradering av SW og HW
  - Reorganisering av data (nye krav, bedre ytelse)

Terminologi, (1/2)

- **Utarbeidet av IFIP WG 10.4 og IEEE CS TC on FTC**
- **Feil-situasjon**
  - spesifisert oppførsel er forskjellig fra faktisk oppførsel
- **feil-årsak => feil => feil-situasjon**

(Eng.: fault => error => failure)

- **Feil-forsinkelse (Eng. error-latency)**
  - Tid mellom feil og feil-situasjon
- **Latent feil**
  - En feil kan legges inn i systemet ved konstruksjon og ligge der som en latent feil til den inntrer som en feil-situasjon. Da er feilen effektiv/aktiv.
- **Tjeneste-utførelse og tjeneste-avbrudd**
  - Fig.1
- **"Backward error-recovery" og "forward error-recovery"**

Terminologi, (2/2)

- **MTTF = mean time to failure**
  - måler gj.snittsverdi for tid til første feil fra oppstart for et antall "identiske" moduler
- **MTTR = mean time to repair**
- **Tilgjengelighet i et ikke-redundant system med reparasjon**

  = MTTF/(MTTF + MTTR)
- **Transiente feil**
  - feil som forsvinner ved ny-oppstarting (Eng. reset) av et system

Økt pålitelighet i komponenter og enheter

- **Komponenter med lang levetid (minst 20 år)**
- **Mindre effektforbruk, som gir**
  - lavere temperatur, som gir
    - » langsommere aldring
- **Økende grad av integrasjon, som gir**
  - færre eksterne tilkoplingspunkter, som gir
    - » større andel av tilkoplingspunkter er godt beskyttet inne på kretsen
- **Bedre produksjonsmetoder**
  - dagens disker blir gammeldagse før de feiler (!?)
  - MTTF = 12 år
- **Bedre (og færre) kabler og kontaker**

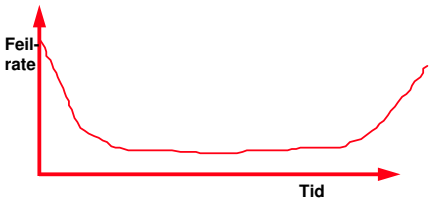
Teknikker for å øke pålitelighet/tilgjengelighet

- **Kvalitets-sikring (QA)**
  - validering (=/= verifikasjon)
  - bruk av metoder og verktøy
  - programverifikasjon
- **Testing**
  - f.eks. forsøk på å oppdage og reparere latente feil før de blir effektive
- **Preventivt vedlikehold**
  - f.eks. bytter ut gamle moduler
- **Behandling av oppståtte (effektive) feil**
  - feil-eliminering ("error masking")
    - » Eks. Feil-rettende kode, dvs. bruk av redundante data for å rekonstruere riktig svar/tilstand
  - feil-oppretting ("error-recovery")
    - » Eks. nekte forespørsel som gir feil, og rekonstruere en feilfri tilstand som setter systemet i stand til å ta i mot nye forespørsler

Konstruksjons-prinsipper

- **Modularitet**
  - avgrenser feilen og dermed effekten
  - lettere reparasjon og utbytting
  - lettere å konstruere med få feil
  - fordelen ved å bruke moduler m.h.p. pålitelighet avhenger av modulenes virkemåte (m.h.p "sikkerhet")
    - » f.eks. om en prosess (modul) har lov til å skrive i andre prosessers adresse-område
- **Feile hurtig**
  - enhver modul bør enten virke korrekt eller stoppe umiddelbart
- **Uavhengig feil-tilstand**
  - hvis en modul feiler, bør feilen ikke påvirke andre moduler
- **Redundans og reparasjon**
  - ekstra moduler bør om mulig være
    - » ferdig installert el.
    - » ferdig konfigurert el.
    - » ferdig produsert

Empiriske erfaringer



- **Feilrate for både maskinvare og programvare følger typisk en "badekar-kurve"**
  - synkende barne-dødelighet
  - rolig midt-periode
  - økende aldrings-dødelighet
- **Aldring i programvare**
  - skyldes i hovedsak vedlikehold, utvidelser og ytelsesforbedringer
    - » retting av en feil kan føre til mer enn en ny feil
    - » situasjoner der en ikke "tør røre systemet lenger"

Feil-tolerant maskinvare

- **Feile hurtig**
  - selvtest
  - sammenlikning (2 el. flere)
- **Ulike former for redundans**
  - løser ikke problemet med konstruksjonsfeil i maskinvaren
  - Fig.2
  - Fig.3

Redundans krever reparasjon

- **Redundans alene forbedrer ikke tilgjengelighet**
  - MTTF
    - » Simplex = 1 år
    - » Duplex = 0.5 år
    - » Triplex = 0.8 år
    - » Pair and spare = 0.7 år
- **Redundans m/reparasjon gir vesentlig bedre tilgjengelighet**
- **"on-line" modul reparasjon**
  - reparasjon og reinstallering (el. erstattning) mens systemet kjører
  - teknisk krevende

Feil i programvare (konstruksjonsfeil)

- **10 ganger vanligere enn maskinvarefeil**
- **Programvare moduler**
  - mer oversiktlig konstruksjon, færre feil
  - lettere å oppdage feil (lokal tilstand)
  - effekt av feil isoleres
  - flere og mer fleksible muligheter for “instrumentering”, testing og feilfinning
- **Defensiv programmering**
  - ekstra kode for å sjekke, I/O, parametre, tilstand og invarianter
  - ekstra overvåkingsprosesser o.l.
- ***N*-versjons programmering**
  - *N* uavhengige grupper konstruerer programvare for samme beregning
  - kostbart
  - selv disse kan gjøre samme feil
    - » feil i spesifikasjon
    - » felles bakgrunn (utdanning, “kultur”)
  - reparasjon er nødvendig og vanskelig

Programvarefeil

- **De fleste programvarefeil er transiente**
  - feilen “har forsvunnet” hvis en prøver igjen senere (i en noe annen sammenheng)
  - “Heisenbug”
- **Bohrbug**
  - “good solid bugs”
- **Ed Adams;**
  - godartede feil (Eng. benign bugs)
    - » rapportert bare en gang
  - ondartede feil (Eng. virulent bugs)
    - » rapportert flere ganger
    - » mindre enn 1 %
  - Konsekvens: loggfør feilen, vent med å reparere !
- **Programvare-pålitelighet er relativt dårlig i dag**
  - men vi har ingen alternativer !

Prosess-par

- **Generalisering for distribuerte systemer av sjekkpunkt-restart teknikken**
- **To prosesser kjører den samme beregningen**
  - primær-prosess
    - » utfører beregninger for klient
    - » sender jevnlig tilstands-info (sjekkpunkt) til backup-prosess
    - » stopper selv og varsler backup-prosess ved feil
  - backup-prosess
    - » overvåker meldingsflyt "passivt" (lagrer klient-meldinger siden siste sjekkpunkt) slik at den kan ta over ansvaret for beregningen
    - » ved feil i primær-prosess tar backup-prosess over og blir ny primær-prosess
- **Tåler maskinvarefeil hvis prosesser kjører på ulik maskinvare**

Transaksjoner

- **Applikasjonsprogrammet kan ved konstruksjon deles inn i transaksjoner**
- **Transaksjon**
  - en samling av handlinger (meldinger, database-oppdateringer og tilstands-endringer) som har følgende egenskaper (ACID)
- ***ATOMICITY* (atomiskhet)**
  - alt eller ingenting
  - commit el. abort
- ***CONSISTENCY* (konsistens)**
  - transaksjonen gir en ny korrekt tilstand
- ***ISOLATION* (isolasjon)**
  - hver transaksjon vil være isolert fra andre samtidige transaksjoner
- ***DURABILITY* (varighet)**
  - utført transaksjon vil overleve senere feil

Eksterne feil-kilder

- **98% av ikke-planlagte system-utfall skyldes ytre påvirkning !(?)**
- **System-par**
  - mulig p.g.a. billigere utstyr og mer automatisert drift
  - fig. 5
    - » uavhengig av strøm, jordskjelv, vær, politikk...!
  - samme prinsipp som disk-par og prosess-par
  - meget driftssikkert
  - oppgradering/reparasjon er enkelt
  - eksempler ar eksistert i flere 10-år
    - » Visa
    - » banker