Toleranța la defecte

Litera A

Acceptability test = Test de acceptabilitate

Aproximații de cost scăzut pentru verificare stării unui sistem care încearcă să maximizeze numărul de erori găsite. Se bazează pe verificări de replicare, temporale, de rezonabilitate, de diagnosticare, de structură şi codificare.

Active replica approach = Metoda replicii active

La fel ca metoda sitului primar, şi această metodă asigură o rezilienţă de ordin *k* folosind pentru aceasta cel puţin *k*+1 noduri pentru replicarea datelor, însă algoritmul nu funcționează în cazul partiționării rețelei. O cerere de acces la date adresată replicii principale (active) este urmată de cereri către toate copiile, acțiunea având succes dacă toate copiile returnează acelaşi rezultat. Defectele la care rezistă metoda sunt cele de tip fail-stop (dacă există o modalitate sigură de difuzare) şi cele bizantine (dacă se folosesc algoritmi de consens).

Asynchronous distributed system = Sistem distribuit asincron

Acele sisteme distribuite în care acțiunile nu se realizează într-un timp finit. Într-un sistem asincron nu se poate face distincție între defectarea unui nod şi întârzierea în comunicație datorată performațelor rețelei de comunicație.

Atomic action = acțiune atomică

O cerință de bază în menținerea consistenței în asigurarea toleranței la defecte este existența unor acțiuni care fie se termină cu succes, fie nu au nici un efect asupra stării sistemului, ele apărând ca indivizibile. Se folosesc în special în sistemele bazate pe tranzacții, unde efectul unei tranzacții nu trebuie să fie vizibil decât dacă aceasta s-a terminat cu succes.

Atomic broadcast = difuzare atomică

Opearația cere ca mesaje diferite de la noduri diferite să fie livrate peste tot în aceeaşi ordine. Deşi această ordonare ajută în aplicațiile tolerante la defecte, uneori trebuie mai mult.

Availability = Disponibilitate

Atribut al gradului de încredere care exprimă disponibilitatea sistemului pentru folosire. Disponibilitatea $A(t)$ se defineşte ca fiind probabilitatea ca o componentă să funcționeze la momentul t, fiind o mărime *instantanee*. Disponibilitatea şi fiabilitatea se confundă dacă nu există reparații. După prima reparație se poate considera că viața este o succesiune de variabile aleatoare, reprezentând duratele de funcționare şi duratele de reparație.

Litera B

Backward error recovery = Recuperare înapoi

Mod de recuperare în care se încearcă restaurarea unei stări anterioare, care se speră a fi lipsită de erori. De aceea trebuie stabilite anumite *puncte de verificare*, care să fie memorate pe un suport extern. Ele vor fi folosite pentru a restaura o stare anterioară. Această metodă este utilizabilă în cazul multor defecte, însă are un overhead mare, exprimat prin spațiul de disc necesar şi prin operațiile de salvare a informațiilor de recuperare. Un alt dezavantaj este aşa numitul efect *domino*: restaurarea unei stări într-o componentă ar putea antrena o restaurare în altele, corecte, şi tot aşa până s-ar ajunge în starea inițială. Revenirea la o stare anterioară reprezintă principalul dezavantaj al metodei, deoarece o serie de calcule vor fi reluate. Metoda nu garantează că eroarea nu se va produce din nou, odată sistemul ajuns în starea anterioară. Totuşi, în cazul defectelor tranzitori, după faza de recuperare nu mai trebuie să se facă nimic, deoarece eroarea a dispărut. Există două abordări principale în ceea ce priveşte tipul informației salvate pe suport extern:

- Salvarea succesiunii de operații (operation-based approach). Metoda foloseşte memorarea acelor operații care modifică starea sistemului şi foloseşte jurnale pentru înregistrare. Exploatarea informațiilor din aceste jurnale se face cu ajutorul unor operații *REDO* / *UNDO*.
- Salvarea stării (state-based approach). Metoda salvează întreaga stare a sistemului, însă există şi variante care salvează doar o parte a stării, din motive de eficiență.

Vezi Backward Error Recovery

Broadcast = Difuzare

Operația de a trimite un mesaj către toate procesele care formează un grup. Operația este deosebit de utilă în implementarea anumitor algoritmi pentru toleranța la defecte (de exemplu pentru algoritmul generalilor bizantini).

Byzantine agreement = Înțelegerea bizantină

Un proces deține o valoare şi toate trebuie să se înțeleagă asupra acelei valori. Vezi si Byzantine generals problem

Byzantine consensus = Consensul bizantin

Fiecare proces are o valoare proprie şi toate trebuie să stabilească o valoare comună. Vezi si Byzantine generals problem

Byzantine failure = defect bizantin

Situație în care defectele sunt arbitrare şi nu respectă nici o regulă. O componentă defectă poate împiedica derularea corectă a calculelor nu doar prin absența unui răspuns, ci şi prin furnizarea unui răspuns eronat, cu şanse de a perturba rezultatul final. O subclasă a defectelor bizantine, dar diferită de celelalte, o constituie clasa defectelor de calcul incorect (CI), în care o componentă produce un rezultat greşit.

Byzantine generals problem = Problema generalilor bizantini

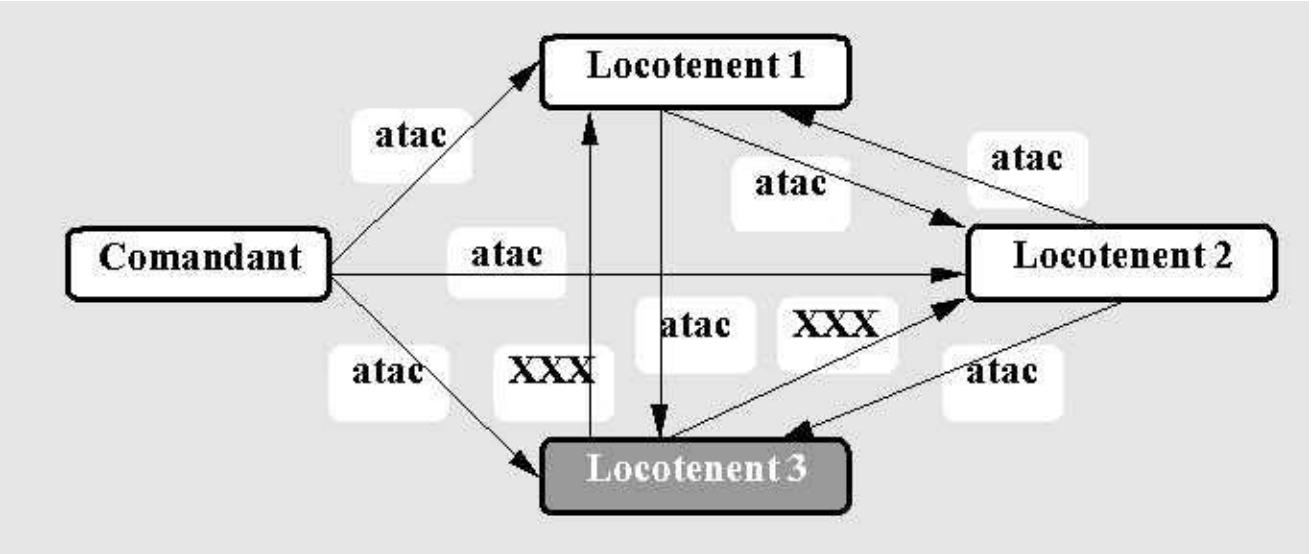
Se referă la defectele aleatoare, cele mai periculoase care pot apare într-un sistem, deoarece nu respectă nici o regulă. Problema se enunță în termenii unui conflict militar, însă similitudinile cu sistemele distribuite sunt mai mult decât evidente. Implicată în problemă este armata bizantină datorită faptului că Imperiul Bizantin a rămas cunoscut în istorie prin desele trădări care ajunseseră să constituie elementul caracteristic al "politicii de stat". Enunțul problemei este următorul: mai multe unitați ale armatei bizantine pregătesc o acțiune împotriva unui adversar. Fiecare unitate este comandată de un general, aceştia comunicând între ei direct, prin legături punct la punct (fiecare legătură având precizate cele două extremități, transmițătorul şi receptorul). Legăturile sunt ideale (fără erori: absența unui mesaj nu poate fi pusă pe seama defectării legăturii) şi nu interferă între ele (nici un general nu poate "asculta" ce comunică alți doi generali între ei).

Într-un mediu asincron această problemă nu are soluție. Dacă se folosește un mediu sincron, atunci algoritmii care rezolvă această problemă operează în mai multe runde, în care se schimbă mesaje. Dacă din cauza unui defect un nod nu mai trimite nici un mesaj, se va folosi o valoare implicită.

Problema consensului are trei variante, înțelefere bizantină, consens bizantin şi consistență interactivă, sintetizate în figura urmatoare

	Înțelegere	Consens	Consistența interactivă
Cine inițiază	**Un proces**	**Toate procesele**	**Toate procesele**
Înțelegere finală	**O valoare**	**O valoare**	**Un set de valori**

Problema înțelegerii bizantine este ilustrată în figura urmatoare



Litera C

Careful disc = disc de încredere

Sistemele care dețin un singur disc, pot să-l transforme într-un disc de încredere prin modificarea operațiilor asupra datelor. Fiecare dată se va afla în două (de regulă) sau mai multe pagini. Un astfel de disc dispune de următoarele operații:

- CarefulRead, care încearcă citirea până ce reuşeşte sau este imposibil;

- CarefulWrite, care face operații de scriere / citire repetat, până ce starea este bună.

Causal broadcast = difuzare cauzală

Anumite mesaje determină procesele din noduri să trimită alte mesaje, care vor determina procesele receptor să trimită alte mesaje. Este astfel anormal ca un mesaj din "ultima generație" să fie livrat (peste tot) înaintea mesajului care a provocat trimiterea sa. Este astfel necesară o păstrare a cauzalității mesajelor, care este asigurată de *difuzarea cauzală*.

Certification trails = Metodă bazată pe trasarea execuției

Metoda foloseşte cel puțin două programe, implementate diferit. Primul program la execuție lasă anumite *urme*, apoi se temină. Al doilea (şi următoarele) se rulează după ce primul s-a încheiat şi se bazează pe urmele găsite. Din această cauză ele pot fi implementate mult mai eficient, durata lor de execuție fiind mult redusă.

Checkpoint = Punct de verificare

Stare a sistemului în funcționaer normală, salvată pe disc pentru a ajuta revenirea sistemului după producerea unor defecte.

Communication medium failures = Anomalii de comunicație

Apar atunci când două calculatoare operaționale nu reuşesc să comunice între ele. Cauzele sunt legate în principal de legăturile de comunicație (întreruperi, zgomote, etc.), dar pot fi cauzate şi de anomalii de sistem sau de disc ale unor noduri intermediare, care se ocupă cu dirijarea mesajelor.

Concurrent processes = Procese competitive

Sunt procese care folosesc resurse comune, dar între care nu există nici un fel de schimb de informații.

Cooperant processes = Procese cooperante

Sunt procese care folosesc resurse comune şi fac schimb de informații.

Crash Failure = Cădere

Situație în care o componentă se opreşte sau îşi pierde starea internă, nemaiputând trece în altă stare.

Litera D

Design failure = defect de proiectare

Sunt acele defecte introduse în timpul proiectării sau modificării sistemului, sunt cel mai greu de tolerat.

Data replication = replicarea datelor

Modalitate de dispunere (alocare) a datelor în sistemele distribuite, astfel încât o unitate de adte are mai multe copii (replici) situate în noduri diferite.

Diagnosis checking = Verificări de diagnosticare

Aceste verificări se fac asupra componentelor sistemului şi sunt efectuate de către sist Constau în compararea valorilor de ieşire produse pentru anumite valori de intrare cu ce se consideră a fi valori de ieşire de referinţă. Aceste verificări nu se pot face decât dacă sistemul este oprit (nu furnizează serviciul), prin urmare se pretează doar pentru auto-teste la pornire.

Distributed Object Migration Environment

Este un mediu de dezvoltare a aplicaţiilor distribuite care suportă puncte de verificare şi repornirea proceselor. Se prezintă sub forma unei biblioteci de obiecte distribuite, peste o reţea de staţii de lucru (NOW - Network Of Workstations) şi un proces daemon pentru depistarea anomaliilor. DOME foloseşte PVM (Parallel Virtual Machine) pentru comunicaţie şi controlul proceselor. Paralelizarea folosită este de tip SPMD (Single Program Multiple Data). La instanţierea unui obiect, acesta este automat partajat şi distribuit pe toate nodurile existente.

DOME

Vezi Distributed Object Migration Environment

Dynamic voting = Votare dinamică

Metodele de vot dinamic obţin majoritatea de la copiile valide la un moment dat. Ele se bazează pe folosirea numerelor de versiune sau pe reasignarea dinamică a voturilor (la apariţia unei anomalii creşte numărul de voturi asignate replicilor, iar la revenirea componentelor defecte numărul de voturi scade). O variantă de vot dinamic este cea care defavorizează operaţiile de scriere şi creşte performanţele la citire (missing writes). Ea are două moduri de lucru: *normal*, când valorile de cvorum sunt r=1 şi w=v, şi de avarie, când fiecare operaţie necesită majoritate.

Litera E

Error = eroare

Este aceea parte a stării unui sistem care poate conduce la anomalii, printr-o secvenţă de acţiuni executate de sistem, dacă nu se iau măsuri corective. O eroare identifică deci acea parte a stării unui sistem care diferă de valoarea dorită. Cauza unei erori este un *defect*. Eroarea fiind o proprietate a unei stări ea poate fi observată şi evaluată. Pe baza erorilor se pot identifica şi anomaliile (ele nefiind proprietăţi sunt mai greu detectabile). Observarea erorilor presupune *monitorizarea* stării sistemului. Monitorizarea trebuie făctă *continuu*, altfel anumite erori pot să nu fie observate. De asemenea, monitorizarea trebuie să cuprindă *întreaga stare* a sistemului, pentru a nu scăpa unele anomali. În practică aceste cerinţe nu pot fi realizate din motive de cost, preferându-se o monitorizare "acceptabil" de deasă şi pentru porţiunea din starea sistemului cea mai susceptibilă de anomalii. O *eroare* este deci manifestarea unui *defect* într-un sistem, care poate conduce la o *anomalie* a sistemului.

Error detection phase = Faza de detecţie a erorilor

Fază a toleranței la defecte în care trebuie să depisteze defectul, după ce acesta s-a produs. Pentru aceasta se fac verificări asupra stării, deducându-se prezența unui defect. O bună detecție a erorilor se face doar pe baza informațiilor din specificație, fără să se țină cont de informațiile suplimentare furnizate de proiectarea internă a sistemului. De asemenea, detectarea erorilor trebuie să fie completă şi corectă: să depisteze toate erorile şi să nu semnaleze erori atunci când ele nu există. Foarte important este ca detecția erorilor să nu depindă de sistemul testat: defectarea sistemului să nu ducă la defectarea componentei care face testarea. În practică nu se fac verificări asupra întregii stări a sistemului, acestea fiind costisitoare şi susceptibile de a produce erori: se fac doar anumite teste de acceptabilitate.

Litera F

Failure = anomalie

Un comportament care deviază de la specificația sistemului, acesta nemaiputând furniza serviciul cerut.

Failure bounding phase = Faza de estimare şi izolare a defectelor

Între apariția defectului şi detectarea erorii se poate scurge un anumit interval de timp, în care eroarea se poate propaga. Scopul acestei faze este de a determina zona de extindere a erorii. Deoarece erorile se răspândesc prin comunicație, va trebui controlat fluxul de informații între componentele sistemului. Identificarea granițelor erorii poate fi făcută *dinamic*, pe baza controlului fluxului de informații sau *static*, dacă se folosesc *bariere* (fire-walls) care să separe anumite arii ale sistemului. Această fază nu apare întotdeauna explicit.

Failure recovery phase = Faza de recuperare a erorilor

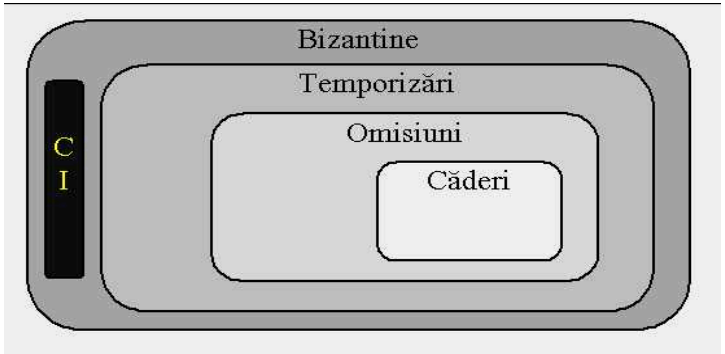
După ce fazele anterioare au fost parcurse (detrecție, respectiv izolare), trebuie ca eroarea să fie înlăturată. Acest lucru poate fi făcut în două moduri, prin recuperare înapoi şi prin recuperare înainte.

Failure types = tipuri de defecte

Defectele pot fi căderi, omisiuni, temporizări, sau bizantine.

Componentă	Tipuri de defecte
Procesor	Căderi sau defecte bizantine
Rețea de comunicație	Toate tipurile de defecte
Ceas	Temporizări, omisiuni, defecte bizantine
Disc	Temporizări, omisiuni, căderi
Software	Toate tipurile de defecte

Relația între aceste defecte este următoarea:



Fault = defect

Reprezintă o condiție fizică anormală. Cauzele producerii unui defect sunt următoarele:

- Erori de proiectare (de specificare sau de implementare);
- Probleme de fabricație;
- Deteriorări în timp;
- Influențe externe (condiții de mediu, interferențe electromagnetice, introducerea de date incorecte, folosiri necorespunzătoare).

Apariția unui defect crează o situație care are potențialul de a genera erori. Însă este posibil ca aceste erori să nu se manifeste pe perioada de observare. Prin urmare, apariția unui defect nu înseamnă automat apariția unei erori. Reciproca însă este adevărată, apariția unei erori este întotdeauna cauzată de existența unui defect.

Fault prevention = prevenirea defectelor

Modalități de asigurare a fiabilității unui sistem eliminarea a cât mai multor deficiențe, în faza de testare, înainte ca sistemul să fie folosit în mod real pentru scopul său. Un sistem proiectat doar după acest principiu are următoarele caracterisitici:

- Lipsa redundanței pentru mascarea defecțiunilor;
- Defectarea sistemului la defectarea oricărei componente a sa;
- Întreținere şi reparație manuală, costisitoare şi lentă.

Fiecare componentă a unui astfel de sistem *trebuie* să funcționeze corect tot timpul; defectarea oricărei componente duce la necesitatea opririi furnizării serviciului, pentru a asigura reparațiile necesare. În acest fel, pe durata reparației serviciul este *inaccesibil*.

Fault tolerance = Toleranța la defecte

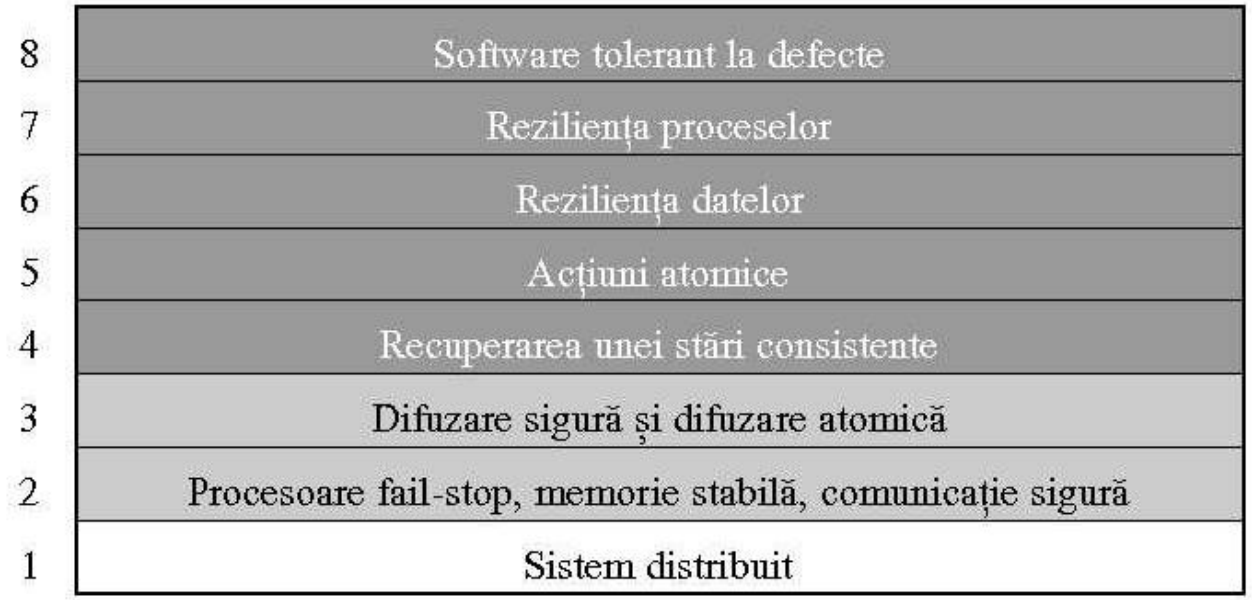
Disciplina care studiază modalitatea de a creşte fiabilitatea sistemelor de calcul, dincolo de ceea ce oferă metodele uzuale. Are drept scop să evite anomalii ale sistemului,

chiar în condiţiile în care anumite componente ale sale sunt defecte. Se foloseşte *redundanţa protectivă* pentru mascarea anomaliilor: există anumite componente suplimentare care nu sunt necesare în absenţa defectelor, ele înlocuiesc repararea manuală, crescând astfel atât fiabilitatea, cât şi disponibilitatea sistemului. Bazele toleranţei la defecte au fost puse chiar de von Neumann, care sugera folosirea *redundanţei* ca principal mijloc de a împiedica funcţionarea necorespunzătoare a unui sistem de calcul. Garantarea unei bune siguranţe în funcţionarea unui sistem de calcul urmează două direcţii de acţiune: minimizarea efectelor provocate de apariţia unor defecţiuni neintenţionate (defecte de material, erori de programare, etc.) şi evitarea efectelor produse de apariţia unor acţiuni maligne deliberate (acces neautorizat în sistem, viruşi, etc.). Prima direcţie este considerată ca fiind baza în toleranţa la defecte, însă şi a doua are importanţa ei: erorile care scapă de testele de validitate ale programelor (bug-urile) sunt cel mai adesea folosite pentru a provoca defecţiuni deliberate.

Toleranţa la defectele neintenţionate are două secţiuni, referitoare la partea de hardware şi la cea de software. Delimitarea între metodele din cele două secţiuni nu este foarte precisă, acceptându-se că o metodă software de toleranţă la defecte este una în care scopul se atinge în special prin software. O definiţie asemănătoare există şi pentru metodele hardware de toleranţă la defecte.

Fault tolerance levels = Niveluri de toleranţă la defecte

Un sistem distribuit cuprinde următoarele niveluri de toleranţă la defecte.



Primul nivel corespunde componentelor fizice uzuale care formează un sistem distribuit: memori, procesoare, discuri, leg ături de comunicaţie. Următoarele două niveluri constituie blocuri de bază pentru serviciile tolerante la defecte construite de nivelurile superioare. Ele asigur ă prin adăugarea unor module specifice, hardware sau software, servicii de bază cum ar fi consens între procese, ceasuri sincronizate, detectarea defecţiunilor, comunicaţie punct la punct. Aceste servicii de bază folosesc nivelurilor superioare, care asigură următoarele servicii tolerante la defecte: consistenţa în prezenţa defectării unor noduri (nivelurile 4 şi 5), servicii neîntrerupte de

defectările unor noduri (nivelurile 6 şi 7), servicii neîntrerupte în prezența unor defecte de proiectare (nivelul 8).

FER

Vezi Forward Error Recovery

Forward error recovery = Recuperare înainte

Mod de recuperare care nu se bazează pe folosirea unei stări anterioare salvate pe disc, eliminând astfel dezavantajele metodei BER, ci încearcă să corecteze starea eronată. Starea în care se va face revenirea este o stare viitoare, cea în care s-ar fi aflat sistemul dacă eroarea nu ar fi avut loc. Deoarece este necesară cunoaşterea cu exactitate a erorii, metoda nu este aplicabilă decât pentru anumite aplicații, fiind puternic dependentă de sistem.

Litera H

Hardware redundancy = Redundanța hardware

Presupune adăugarea unor componente hardware suplimentare pentru a realiza toleranța la defecte.

Highly dependable system = Sistem de încredere

Un sistem de calcul care controlează activități critice (controlul zborurilor, supravegherea pacienților, controlul unor operații financiare, etc.). Gradul de încredere se exprimă prin încrederea care se poate avea în serviciul oferit de sistem. Conceptul este general, şi are mai multe atribute:

- *Fiabilitatea*;
- *Disponibilitatea*;
- *Siguranța*;
- *Securitatea*.

Litera I

Independent processes = Procese independente

Sunt procese care accesează obiecte (date) din seturi disjuncte.

Interactive consistency = Consistență interactivă

Fiecare proces deține o valoare şi trebuie să se stabilească un set de valori comune.

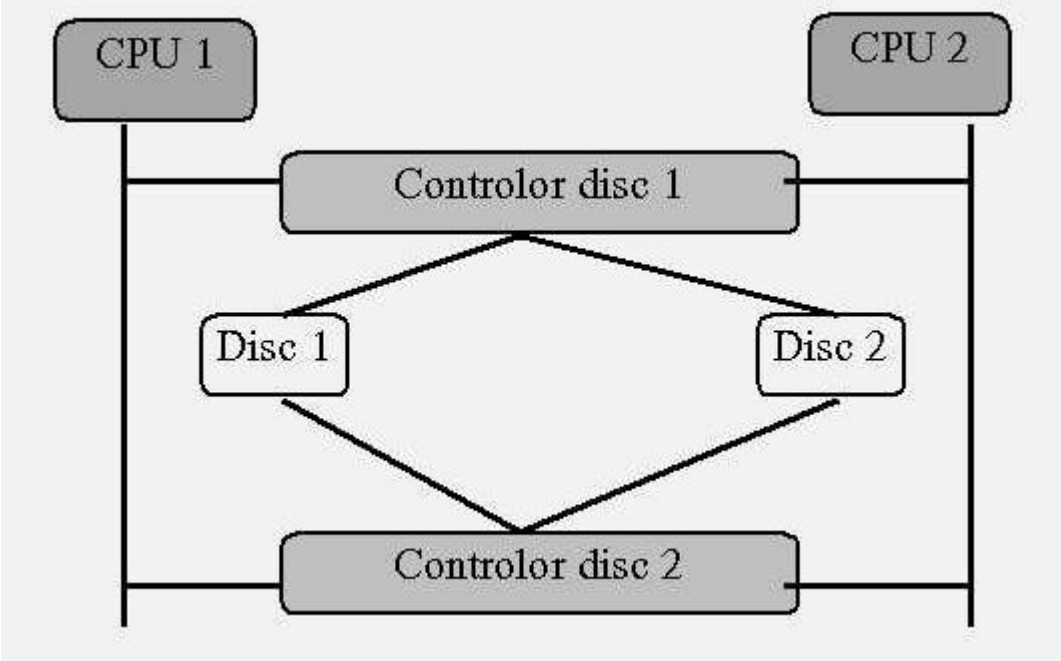
Litera M

Mean Time To Failure = Durata de viaţă estimată

Se obţine ca medie, prin integrare: $E[X] = \int_0^\infty tf(t)dt = \int_0^\infty R(t)dt = \frac{1}{\lambda}$. Un sistem este format de obicei din mai multe componente. Dacă acestea sunt legate în serie, atunci sunt valabile următoarele formule: $R(t)_{serie} = \prod_{i=1}^{n} R_i(t)$ şi $MTTF_{serie} = \frac{1}{\sum_{i=1}^{n} \lambda_i}$. În momentul în care una dintre componentele sistemului se defectează, întreg sistemul se va defecta. Pentru legarea în paralel, formulele de calcul sunt următoarele: $X = \max\{X_1, X_2, .., X_n\}$ şi $MTTF_{paralel} = \frac{1}{\lambda}\sum_{i=1}^{n}\frac{1}{i} \approx \frac{\ln n}{\lambda}$. Un astfel de sistem supravieţuieşte atât timp cât cel puţin o componentă este corectă. În cazul redundanţei bazate pe mai multe componente care aşteaptă defectarea, formulele sunt următoarele: $X = \sum_{i=1}^{n} X_i$ şi $MTTF_{redund} = \frac{n}{\lambda}$, obţinute prin adunarea formulelor pentru o componentă. În fine, dacă se foloseşte vor majoritar cu TMR, formulele devin: $R_{TMR}(t) = 3R^2(t) - 2R^3(t) = 3e^{-2\lambda t} - 2e^{-3\lambda t}$.

Mirroring = oglindire

Dacă un sistem deţine doar două discuri poate face *oglindirea* lor, scrierea făcându-se pe ambele discuri, iar citirea putându-se face de pe oricare. Metoda poate masca defectarea unui singur disc; dacă se defectează ambele simultan nu se poate face nimic. Dacă MTTF este timpul de viaţă al unui disc (Mean Time Till Failure) şi MTTR este durata unei reparaţii (Mean Time Till Repair), atunci timpul de viaţă al unui disc cu mirroring este dat de formula $MTTF_{mirroring} = \frac{MTTF}{2} * \frac{MTTF}{MTTR}$. Timpul de viaţă în acest caz este deci jumătate din timpul de viaţă al unui disc, înmulţit cu probabilitatea ca al doilea disc să fie defect când primul se repară.



MTTF

Vezi Mean Time To Failure.

Multicast = trimitere multidestinație

Operația de a trimite un mesaj către un subgrup dintre procesele care formează un grup. Operația este deosebit de utilă în implementarea anumitor algoritmi pentru toleranța la defecte.

Litera N

N-Version programming = programare cu N versiuni

Metodă care asigură toleranța software prin folosirea votului asupra rezultatelor produse de *N* (*N*>=3) versiuni ale aceluiaşi program. Există şi situații când votarea nu poate fi folosită, deoarece sunt posibile mai multe răspunsuri corecte (de exemplu rezolvarea unei ecuații de grad *n*).

Litera O

Omission Failure = Omisiune

Situație în care o componentă nu răspunde la anumite valori de intrare.

ORCHESTRA

Este un mediu de dezvoltare a aplicațiilor tolerante la defecte, bazat pe *injecția de defecte*. A fost dezvoltat inițial pentru Mach, un sistem de operare de timp real, dar a fost portat ulterior şi pe Solaris şi SunOS. Se foloseşte în special pentru verificarea şi evaluarea caracterisiticilor de toleranță la defecte şi constrângeri de timp ale protocoalelor distribuite de comunicație.

Prin găsirea unor script-uri corespunzătoare se pot detecta erori de proiectare sau implementare, se pot identifica încălcări ale specificației protocolului şi se pot obține detalii de implementare. Un protocol distribuit este văzut ca o abstractizare, în care o mulțime de participanți comunică prin schimb de mesaje. Înainte de protocolul de test se inserează un nivel suplimentar, PFI (Protocol Fault Injection), care se ocupă de execuția unor scripturi deterministe sau generate aleator, care au rolul de a testa sistemul prin introducerea anumitor defecte. În plus, acest nivel poate introduce diverse mesaje în sistem, cu scopul de a orchestra atingerea unei anumite stări.

ORCHESTRA cuprinde o parte independentă de protocol (motorul de injectare a defectelor cu structurile de date asociate şi interfața utilizator pentru generarea scripturilor). Partea dependentă de protocol constă în "adezivul" cu care motorul de injecție se ataşează stivei de protocoale. Injecția de defecte se face la nivel de mesaje, prin interceptarea traficului şi introducerea de mesaje noi. Limbajul folosit pentru scripturi este Tcl, însă se poate folosi şi o interfață grafică pentru acest lucru.

Litera P

Permanent failure = defect permanent

Au o durată permanentă, componenta defectă nu îşi mai revine, sau o face după o perioadă mare de timp.

Phantom voting = Votare cu fantome

Variantă de votare dinamică, care foloseşte *votul fantomelor*, şi creşte performanțele la scriere: când un nod se defectează, se crează automat un nod fantomă, având acelaşi număr de voturi, însă fără memorie externă. Fantomele nu participă decât la cvorumurile pentru scriere, trebuind totuşi ca orice astfel de cvorum să conțină cel puțin un nod adevărat.

Primary site approach = Metoda sitului primar

Scopul acestei metode este să permită accesul la date (sau la orice resursă în general), chiar dacă apar anomalii ale nodurilor sau legăturilor. Deoarece defectele legăturilor pot fi asimilate cu defecte ale unor noduri, se poate considera că există doar defecte ale nodurilor. Pentru a asigura date cu rezilienta de ordinul k (date accesibile când k noduri sunt defecte) trebuie ca cel puțin $k+1$ noduri să conțină replici ale datelor. Dintre acestea, unul este nod primar, celelalte sunt noduri de rezervă (pasive). Fiecare actualizare efectuată de nodul primar va fi precedată de o cerere de actualizare adresată către cel puțin k noduri de rezervă, în felul acesta datele fiind păstrate consistente. Calculele efectuate de nodurile de rezervă pot fi reduse, acestea primind de la nodul primar informații despre un punct de recuperare; această abordare presupune că nodurile de rezervă pot distinge un punct de recuperare eronat.

Dacă se defectează mai mult de k noduri nu se mai poate face nimic. Dacă însă se defectează mai puțin de k noduri de rezervă nu se observă nimic, sistemul funcționând normal. Defectarea nodului primar determină inițierea unui algoritm de alegere, pentru stabilirea unui nou primar dintre nodurile de rezervă (de obicei se alege nodul în funcțiune cu numărul de ordine cel mai mare). Performanțele algoritmului sunt reduse drastic în cazul partiționării rețelei, caz în care doar partiția conținând nodul primar va funcționa.

Process = proces

Reprezintă execuţia unui program secvenţial; în mediile multiprogramate se rulează mai multe astfel de procese în acelaşi timp, după relaţiile care se stabilesc între ele procesele putând fi independente, competitive sau cooperante.

Process failures = Anomalii ale proceselor

Se produc atunci când rezultatul furnizat de proces nu este cel corect, sau starea procesului nu mai corespunde specificaţiei sale, sau procesul nu mai evoluează. Cauzele acestui tip de anomalie sunt blocajele (deadlocks), timeout-urile, încălcarea protecţiilor, furnizarea de date greşite, încălcarea regulilor de consistenţă (în special pentru tehnicile optimiste de control al concurenţei). În această situaţie, un proces poate fi distrus şi reluat de la început sau reluat dintr-o stare anterioară.

Litera R

RAID

Vezi Redundant Array of Inexpensive Disc

Redundant Array of Inexpensive Discs = Matrice de discuri

Datele sunt împrăştiate pe mai multe discuri, fiind întreţesute la nivel de bit. Acest lucru face posibilă citirea lor în paralel. Discurile sunt grupate, fiecare grup având discuri proprii de verificare.

Reliability = Fiabilitate

Atribut al gradului de încredere care se referă la continuitatea serviciului şi este de cea mai mare importanţă pentru toleranţa la defecte. Dacă X este o variabilă aleatoare care modelează timpul de viaţă al unui sistem, se defineşte *fiabilitatea* sistemului ca fiind probabilitatea ca sistemul să fie în funcţiune la momentul *t*: $R(t) = P(X > t) = 1 - F(t) = e^{-\lambda t}$, unde λ este *rata de defectare*. Se observă că $R(t = 0) = 1$ (iniţial sistemul era în funcţiune) şi că $R(t = \infty) = 0$ (nici un sistem nu durează veşnic).

Reliable broadcast = difuzare sigură

Scopul unei astfel de operaţii de difuzare din punct de vedere al toleranţei la defecte este ca fie toate procesele să recepţioneze mesajul, fie nici un proces să nu o facă, împiedicând în felul acesta apariţia unor inconsistenţe. Nu se face nici un fel de presupunere asupra *ordinii* mesajelor.

Replication checking = Verificări de replicare

Ele sunt cele mai puternice şi complete, implementându-se fără informaţii despre structura internă a sistemului. Metoda se bazează pe tehnici de votare, fiind din această cauză destul de costisitoare. Cea mai cunoscută metodă bazată pe vot este TMR (Triple Modular Redundancy). Pentru ca metoda să aibă şanse mai mari, este bine ca modulele să fie implementări diferite ale aceleiaşi specificaţii.

Resonabilty checking = Verificări de "rezonabilitate"

Stabilesc dacă starea unei componente este rezonabilă, pe baza unor verificări de încadrare în domeni. Din aceast ă cauză ele nu pot stabili exact corectitudinea.

Run-time failure = defect operațional

Sunt acele defecte introduse în timpul funcționării sistemului, din motive fizice.

Litera S

Safety = Siguranța

Atribut al gradului de încredere care cere împiedicarea efectelor dezastruoase asupra mediului.

Secondary storage failures = Anomalie a dispozitivului auxiliar de stocare

Apare atunci când datele aflate pe disc nu mai pot fi accesate. Cauzele unei astfel de anomalii pot fi erori de paritate, erori ale capulu de citire / scriere, existen ța unor particule de praf pe suprafața discului. Revenirea se bazează pe menținerea unor arhive şi a unor jurnale unde se înregistrează toate operațiile efectuate.

Security = Securitate

Atribut al gradului de încredere care se referă la prevenirea accesului neautorizat la informați.

Software redundancy = Redundanța software

Presupune adăugarea unor programe sau instrucțiuni suplimentare pentru a asigura toleranța defecte.

Stabil disc = disc stabil

Discul poate fi făcut *stabil*, prin implementarea următoarelor operații, considerând că paginile sunt grupate în perechi:

- StableRead, care face CarefulRead pe o pagină; dacă nu reuşeşte încearcă pe a doua;
- StableWrite, care face CarefulWrite pe ambele pagini.

Static voting = Votarea statică

Această metodă se bazează pe acordarea unui număr *fix* de voturi pentru fiecare replică. Orice operație trebuie să încerce să-şi asigure un *cvorum*, un număr minim de voturi de acceptare din partea replicilor dispuse (libere) să execute operația. Dacă notăm cu r numărul de voruri necesare pentru operația de citire, cu w numărul de voturi necesare pentru operația de scriere şi cu v numărul total de replici (sau voturi), atunci trebuie să fie îndeplinite următoarele relații: $r+w>v$ şi $w>v/2$. Prima relație asigură că orice cvorum de citire şi orice cvorum de scriere au replici comune, iar a doua garantează majoritatea voturilor pentru o operație de scriere.

Deoarece numărul de voturi creşte odată cu creşterea numărului de replici, mesajele schimbate pentru obținerea unui cvorum devin importante din punct de vedere al costului. O metodă care încearcă reducerea comunicației este *votarea ierarhică*, care foloseşte o organizare arborescentă a replicilor. Fiecare nod are acelaşi număr de succesori, replicile fiind menținute de obicei în frunze. Pe fiecare nivel se defineşte recursiv un cvorum, în funcție de cvorumul nivelului anterior.

Structural checking = Verificări de structură şi de codificare

Se fac de obicei asupra datelor şi sunt favorizate de folosirea redundanței. Ele trebuie să asigure că valorile existente sunt consistente cu restul sistemului.

Synchronous distributed system = Sistem distribuit sincron

Acele sisteme distribuite în care fiecare acțiune se realizează într-un timp finit.

System failure = Anomalie a sistemului

Reprezintă un comportament al sistemului care nu este consistent cu specificația sa. Sistemul nu mai furnizează servicii conform specificației sale. Acest comportament anormal este cauzat de obicei de apariția unor defecte în componentele sistemului. Probabilitatea de anomalie a sistemului creşte odată cu creşterea numărului de componente şi cu creşterea complexității sistemului. Se produce atunci când procesorul nu mai funcționează, din motive hardware sau software (defectare CPU, defectare memorie principală, defectare a magistralei sistem, cădere de tensiune). Remediul este oprirea sistemului şi repornirea sa dintr-o stare bună. În funcție de modul de comportare la repornire se pot evidenția următoarele clase de anomalii sistem:

- amnezie, când se reporneşte într-o stare care nu depinde de cea cu defect;
- amnezie parțială, starea de revenire este parțial formată din valori corespunzătoare stării de dinainte de defect, iar parțial din valori predefinite (cazul serverelor de fişiere);
- pauză, când se reporneşte în starea de dinainte de eroare;
- oprire definitivă, când sistemul nu mai porneşte.

System life time = Timpul de viață al unui sistem

Durata de funcționare corectă a unui sistem (adică timpul până la defectare), este o mărime impredictibilă. De aceea poate fi modelată printr-o variabilă aleatoare X. Cel mai adesea distribuția acestei variabile se consideră a fi exponențială, adică având funcția de distribuție $F(X) = \begin{cases} 1 - e^{-\lambda X}, & X \geq 0 \\ 0, & X < 0 \end{cases}$ şi funcția de densitate de probabilitate $f(X) = \begin{cases} \lambda e^{-\lambda X}, & X > 0 \\ 0, & X \leq 0 \end{cases}$. Această distribuție are *proprietatea Markov*, adică este fără memorie: distribuția unei astfel de variabile după scurgerea unui interval de timp *t* este tot exponențială. Pentru cazul timpului de viață al unui sistem aceasta înseamnă că sistemul *uită* cât timp a fost în funcțiune.

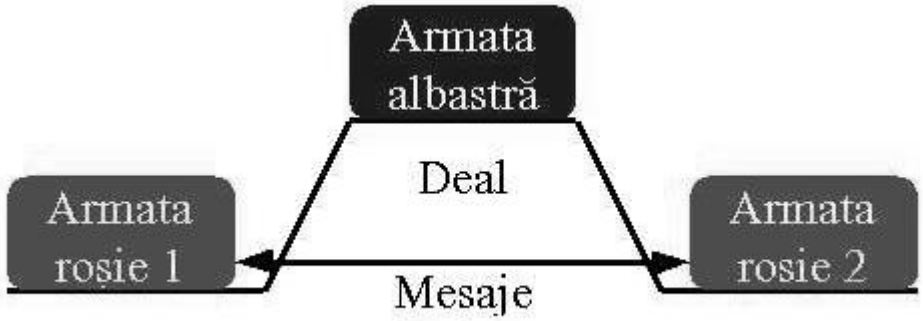
Litera T

Temporal checking = Verificări temporale

Sunt utile dacă specificaţia conţine constrângeri temporale. Detectarea unei erori se face prin pornirea unui ceas, care la expirarea unui interval de timp prestabilit anunţă producerea timeout-ului. În cazul unui sistem distribuit, absenţa răspunsului de la un anumit nod într-un timp dat poate însemna defectarea acelui nod.

The two army problem = problema celor două armate

Cazul în care legăturile de comunicaţie pot fi afectate de erori este analizat de altă problemă cu iz militar, *problema celor două armate*. Enunţul acestei probleme este următorul: două corpuri ale aceleiaşi armate (roşie de exemplu) se află de cele două părţi ale unui deal ocupat de inamic (armata albastră de exemplu), şi pe care trebuie să-l cucerească. Dacă cele două corpuri de armată nu atacă simultan, ele vor fi distruse. Comandanţii celor două corpuri de armată pot comunica între ei trimiţând mesaje prin curieri, însă aceştia se pot rătăci sau pot cădea prinonieri. Se arată relativ uşor că cei doi comandanţi nu pot cădea de acord: ultimul dintre ei care trimite un mesaj nu are garanţia că mesajul lui a fost recepţionat. Descrierea acestei probleme este prezentată în figura urmatoare



Time redundancy = Redundanţa de timp

Alocă intervale suplimentare de timp pentru reluarea execuţiei de mai multe ori.

Transient failure = defect temporar

Se caracterizează printr-o durată limitată, şi are drept cauză o funcţionare temporar eronată sau o interferenţă externă. Cauzează anomalii şi erori doar pe perioada cât există. Dacă apar rar, nu sunt grave şi chiar pot fi ignorate; cauzează probleme doar dacă sunt intermitente, adică dacă apar des.

Timing failure = Temporizare

Situaţie în care se răspunde prea repede sau prea devreme.

TMR = TMR

Vezi Triple Modular Redundancy

Transaction = tranzacție

Este o succesiune de operații care se execută *concurent* asupra variabilelor dintr-o bază de date distribuită. Pe lângă operațiile de bază (*READ* şi *WRITE*) există două tipuri de operații speciale, de pornire a unei tranzacții (*BEGIN*) şi de terminare a sa (*COMMIT* pentru terminarea cu succes şi *ABORT* pentru terminarea prin abandon). În sistemele tranzacționale două acțiuni sunt vitale pentru buna lor funcționare: controlul concurenței şi serializabilitatea. În timp ce prima are drept scop împiedicarea inconsistențelor şi a blocării sistemului, a doua doreşte să stabilească o ordonare a acțiunilor, asemănătoare cu o ordine serială a lor.

Triple Modular Redundancy

Una dintre schemele de asigurare a toleranței la defecte cel mai des folosite (TMR) în abordarea hardware a fost propusă de însuşi von Neumann. Ea se bazează pe folosirea a trei componente care îndeplinesc aceleaşi funcți, rezultatele oferite de ele fiind prelucrate de un circuit de vor majoritar. Schema este ilustrat ă în figura 2.8 şi rezistă la defectarea unei singure componente.

Din cauză că şi circuitul care realizează funcția de vot majoritar este supus aceloraşi posibilități de defectare, se folosesc trei astfel de circuite în paralel. Această variantă este deosebit de potrivită în cazul defectelor tranzitori, pentru c ă nu trebuie să se facă nimic pentru detecția erorii şi nici pentru recuperarea după eroare. Metoda poate avea şi unele deficiențe, anumite erori putând să se compenseze, dacă funcția de vot majoritar se bazează pe operația de SAU-EXCLUSIV