BHG ORION TERTA

MŰSZAKI KÖZLEMÉNYEK

XXVIII. évfolyam 1982 8. szám

A hibatűrő rendszerek elvi kérdései

VERESS TIBOR
BHG

BEVEZETÉS

A mikroprocesszorok megjelenése és elterjedése korábban elképzelhetetlen intelligenciájú és teljesítményű vezérlő egységek építését tette lehetővé. A mikroprocesszorok nagy bonyolultságú, igen összetett rendszerek vezérlésére is alkalmasak, ezáltal mind technológiailag, mind gazdaságilag elérhetővé vált nagy megbízhatóságú, hibatűrő rendszerek építése, ami új távlatokat nyit a híradástechnikában. Egyes területeken már ma is elsődleges szempont a hibamentes működés.

Természetes, hogy a különböző felhasználási területeken a nagyobb megbízhatóság különböző igényeket támaszt a berendezésekkel szemben. Az alábbiakban azokról a szempontokról lesz szó, amelyek bármely felhasználási területen hasznosíthatók.

Ezzel elérkeztünk ahhoz a nagyon fontos kérdéshez, hogy lehet-e — s ha igen, hogyan — meghibásodható elemekből hibátlanul működő rendszert létrehozni. Elsőként Neumann János foglalkozott ezzel a problémával. Az élő szervezeteket vizsgálva arra a megállapításra jutott, hogy azok a jelentéktelen hibára utaló jelzést egyszerűen figyelmen kívül hagyják, lényeges hiba esetén pedig a hibás „részegységet tartalékra kapcsolják", s funkcióit az elegendően sokoldalú részeknek adják át. Amennyiben a kiiktatott rész regenerálása sikertelen, a rendszer nem foglalkozik tovább vele. Ily módon a rendszer élettartamát a kijavítatlan hibák száma és súlyossága határozza meg [1].

Hogy a hibatűrő számítás napjainkra realitássá vált, annak egy további oka a VLSI áramkörök árának rohamos csökkenése. Hibatűrő tulajdonságokat ugyanis csak a beépített elemek számának — következésképp a berendezés árának — növelésével érhetünk el.

1. A megbízhatóság növelésének lehetőségei

A megbízható számítás legáltalánosabb megfogalmazása: az algoritmus korrekt végrehajtása. Ez az — Avizienistől számazó [2] — tömör megfogalmazás azt jelenti, hogy a működés eredménye nem térhet el a gép szoftver és hardver által meghatározottól. Hogy ezt elérhessük, a következő elemekről kell gondoskodnunk:

- a szoftver-specifikáció korrektsége és komplettsége;
- a programok tesztelése és ellenőrzése;
- a hardver hibáinak kiküszöbölése;
- folyamatos, korrekt programvégrehajtás és adatvédelem esetleges hardver-hiba esetén;
- a rendszer védelme a hiba által okozott szétesséssel vagy szándékos behatással szemben.

A huzamosabb idejű megszakítás nélküli működés biztosítására — különösen karbantartó személyzet nélkül — alapvetően két lehetőség kínálkozik:

- hibatűrés (fault-tolerance),
- hiba-nemtűrés (fault-intolerance).

A hiba-nemtűrés alkalmazása eleve megköveteli a megbízhatatlan elemek kiküszöbölését. A rendszer megbízhatóságának növekedését nagy megbízhatóságú alkatrészek biztosítják. Ebben az esetben az üzemeltetési árat növeli a karbantartás és programmegszakadás ára. Hibatűrő tervezés esetén a számítás megbízhatóságát védő redundancia biztosítja.

Az ötvenes, hatvanas években a hiba-nemtűrést alkalmazták elsősorban, aminek a redundancia miatti árnövekedés volt az oka. Az utóbbi években a hardver-tervezésben előtérbe került a hibatűrés, amit a technológia fejlődése és a megbízhatósági követelmények növekedése tett elkerülhetetlenné.

Látható, hogy a két lehetőség kiegészíti egymást (komplemensek), és hogy a megbízhatósági követelményeket a kettő valamelyikével lehet kielégíteni. A gyakorlat és az analízis azt mutatja, hogy a legmegbízhatóbb számítás a kettő kiegyensúlyozott alkalmazásával érhető el [3, 4].

1.1. Megbízható rendszerek tervezése

Egy számítógép vagy vezérlő megbízhatatlanságát a logikai struktúra fizikai megépítésének tökéletlensége okozza. A megbízhatóság-elmélet a rendszer R(T) megbízhatóságát annak a valószínűségeként definiálja, hogy az a $t=T$ időpontig kifogástalanul mű-

ködik, ha az indulási idő $t=0$. Számítógépek esetén a hibátlan működés azonban más, mint egyéb berendezéseknél. Célszerűbb egy program korrekt végrehajtásáról beszélnünk, mint a rendszer komponenseinek hibátlan voltáról.

A program hibátlan végrehajtása a következő feltételek kielégítését jelenti:

- a programok és adataik nem változnak vagy akadoznak hiba esetén;
- a programok eredményei nem tartalmaznak zavar okozta hibát;
- a programok végrehajtási ideje nem lép túl egy specifikált határt;
- a programok számára rendelkezésre álló tárolókapacitás egy meghatározott minimum fölött marad.

Ezeket a specifikációkat (pl. végrehajtási idő, tárolókapacitás stb.) a rendszer architektúrája, ill. felhasználója határozza meg.

Az előzőekben láttuk a megbízható rendszerek megvalósításának két szélsőséges módját: a nem hibatűrő és a hibatűrő tervezést. Mivel az előbbit történelmileg is előbb alkalmazták, tekintsük előbb át ennek a jellemzőit. A nem hibatűrő tervezés elemei a következők:

- a legmegbízhatóbb elemeket kell kiválasztani;
- kipróbált eljárást kell alkalmazni a komponensek csatlakoztatására és az alrendszerek kiválasztására;
- a rendszer összeállításakor gondosan ki kell szűrni az előre látható külső zavarokat;
- a rendszer megbízhatósága az ismert vagy előírt komponens- és csatlakozóhibák alapján számítható.

A fentiek alapján könnyen belátható, hogy a nem hibatűrő rendszerek esetén a program korrekt végrehajtásának valószínűsége megegyezik a hardver hibamentességének valószínűségével. Nagy megbízhatóságú rendszerek építéséhez a nem hibatűrő tervezés emiatt igen korlátozott lehetőségeket nyújt, hiszen a megbízhatósági követelmények kielégítése továbbra is a technológia és nem az architektúra függvénye. Ez azonban nem jelentheti ennek az eljárásnak a mellőzését, ill. figyelmen kívül hagyását, hiszen — mint majd a későbbiekben még látni fogjuk — számos területen előnyösen alkalmazható.

Egy rendszer megbízhatóságának számottevő javulása redundancia alkalmazásával érhető el, amelynek megvalósítási módjai a következők:

- hardver eszközök további beépítése (hardver redundancia);
- további programok, programszegmensek használata (szoftver redundancia);
- a működés megismétlése (idő redundancia).

Ezek a hibatűrő tervezés eszközei. Az így tervezett rendszerek esetén egy program konkrét végrehajtásának valószínűsége nem egyezik meg a hardver hibamentességének valószínűségével, hanem annál — a redundancia mértékétől függően — nagyobb.

A hibatűrő rendszerek az alábbi három jellemzővel rendelkeznek:

- komponensek (hardver) és programok (szoftver) sorát tartalmazzák;
- kezdetben hibamentesek és a program végrehajtása alatt védettek a hibák megszakító hatása ellen;
- működési hiba esetén is korrektül végrehajtják a programot.

A két megoldás — hibatűrő és nem hibatűrő tervezés — tehát alapvetően különbözik egymástól. Az előbbinél a rendszer megbízhatóságát elsősorban az architektúra határozza meg, az utóbbinál a technológia.

1.2. A működési hibák osztályozása

A redundancia beépítésének célja az esetleges hibák maszkolása. Nyilvánvaló, hogy a megvalósítás módja a hiba jellegétől függően más és más. Célszerű ezért röviden áttekinteni az előforduló hibákat.

A működési hiba definíciója: egy vagy több logikai változó értékének eltérése a hardver által meghatározottól. A hibákat az alábbi módon osztályozhatjuk: időtartam szerint:

- tranziens
- állandó,

kiterjedés szerint:

- lokális (egy változó)
- elosztott (több változó),

érték szerint:

- meghatározott (fix érték)
- nem meghatározott (változó érték).

A hibák időtartam szerinti osztályozása alapvető fontosságú, hiszen a tranziens és állandó hibák merőben eltérő helyreállítási eljárást kívánnak. Tranziens hiba pl. a program újrafuttatásával javítható, míg állandó hiba esetén a korrekt működés a hibás egység kiváltásával biztosítható.

A kiterjedés és érték szerinti osztályozás hasonló okokból szintén fontos. A hibadetektálás, hibalokalizálás, a megfelelő elhárító eljárás tervezéséhez, ill. a rendszer architektúrájának tervezéséhez adnak támpontot.

2. A védő redundancia megvalósítási lehetőségei

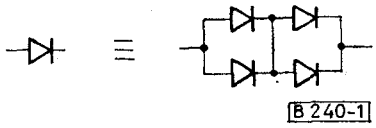
Hibatűrő tervezés esetén a megbízhatóság növekedését redundancia segítségével érjük el. Ennek sikere az alábbi három forma szisztematikus és kiegyensúlyozott alkalmazásával biztosítható:

- hardver redundancia: járulékos elemek (alrendszerek, modulok) beépítése,
- szoftver redundancia: speciális programok futtatása,
- idő redundancia: a működés megismétlése.

Az alkalmazott hardver és szoftver redundanciát a modulok időbeni működésének alapján további két csoportra oszthatjuk: sztatikus és dinamikus redundanciára.

2.1. Sztatikus hardver redundancia

A hardver redundanciának ez a fajtája „maszkoló" redundancia néven is ismert, mivel a járulékos komponens a hardver hibájának hatását maszkolja. (Pl. az *1. ábrán* látható kapcsolás bármely elem hibája — szakadás vagy rövidzár — esetén is dióda marad. Sőt, szerencsés esetben két vagy akár három elem is meghibásodhat.)



1. ábra. A sztatikus (maszkoló) hardver redundancia alkalmazása

Látható, hogy a sztatikus hardver redundancia alkalmazása esetén a komponens hibáját a rendszer nem észleli, az nem okoz késleltetést sem a program futása során. Emiatt azonban a hiba észrevétlen marad, ami a rendszer állagának fokozatos romlását eredményezi, megteremtve ezzel egy későbbi szétesés lehetőségét.

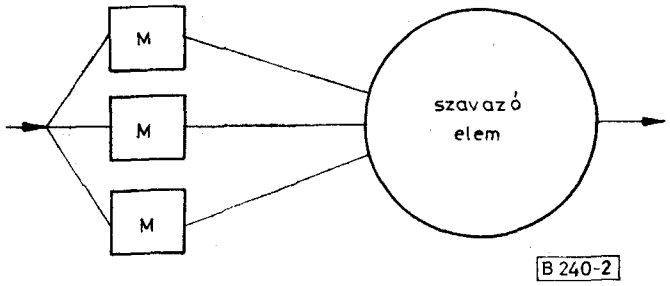
A sztatikus hardver redundancia — maszkoló tulajdonságánál fogva — egyaránt védelmet nyújt tranziens és permanens hibák ellen, alkalmazása azonban költséges. A gyakorlatban két eljárás terjedt el:

— egyedi alkatrészek duplikálása (l. fenti példa),
— triplikálás szavazással (TMR=Triple Modular Redundancy).

A szavazásos (TMR) eljárás szintén maszkoló tulajdonsággal rendelkezik. Ez az oka annak, hogy sztatikus redundanciaként ismert. Lényeges eltérés azonban a sztatikus duplikálással szemben, hogy a szavazó elem képes a hibás egység felismerésére. A triplikált modul ezáltal javíthatóvá válik, azaz a rendszer állagának fokozatos romlása megakadályozható. A szavazó elem megbízhatósága azonban döntően befolyásolja a működést, annak különösen megbízhatónak kell lennie. Hogy ezt hogyan oldják meg, azt a későbbiekben még látni fogjuk.

A *2. ábrán* a TMR elvi blokksémája látható. Az M-mel jelölt modulok azonos funkciót látnak el, szinkron működnek. Ha közöttük eltérés mutatkozik, a szavazó elem a két megegyező eredményt fogadja el az eltérő harmadikkal szemben. (Az eltérő eredményből állapíthatja meg a modul hibás voltát.) Feltűnő, hogy ha egyszerre két modul hibázik, akkor a kimenet is hibás lesz. Ennél is nagyobb gond azonban, hogy ilyenkor a jó modul minősül hibásnak. Ez a probléma a hibatűrő tervezés sarkalatos pontja, amiről a dinamikus redundancia ismertetésekor még bővebben szó lesz. Egy lehetséges megoldás, hogy öt, hét stb. (páratlan számú) elemet kapcsolnak párhuzamosan. Ebben az esetben azonban jelentős az árnövekedés.

Csupán három modul alkalmazása is elegendő lehet, ha az ún. egyszeres hiba feltételezésével élünk. Ez azt jelenti, hogy egyszerre csak egy elem meghibásodásával számolunk. Ez azzal a követelménnyel jár, hogy egy hiba bekövetkezése és észlelése között más egység nem romolhat el, ill. a helyreállításnak olyan gyorsnak kell lennie, hogy közben újabb hiba ne következhessen be. Ezen követelmények kielégítése speciális áramkörökkel és programokkal valósítható meg.



2. ábra. Triplikálás szavazással

2.2 Dinamikus hardver redundancia

Dinamikus hardver redundancia alkalmazása esetén a rendszer önellenőrző és önjavító képességgel rendelkezik (pl. hibajavító kódok). A dinamikusan redundáns rendszer tehát képes a hiba észlelésére, majd annak hatására helyreállító akció végrehajtására. Hibajelzés alkalmazása esetén karbantartó személyzet segítségével elérhető, hogy a rendszer állaga az idő folyamán elvileg nem romlik.

A dinamikus redundancia eszközei, amelyekkel minden dinamikusan redundáns rendszernek rendelkeznie kell:

a) *Modularitás*

A rendszert modulokra (alrendszerekre) bontjuk, amelyek összekapcsolása változtatható lehet. Ha a modulok egymás feladatait is képesek ellátni, akkor feladatátcsoportosítással megvalósítható a helyreállítás. Nagyon fontos a hibák modulok közti terjedésének megakadályozása az egyszeres hiba feltételezhetősége szempontjából. A modularitás a hibabehatárolásnál is nagy segítséget nyújt.

b) *Hibadetektálás*

A rendszernek képesnek kell lennie a hiba észlelésére. Ez a tulajdonság a következő eszközökkel érhető el: hibakód, státusjel használata, önellenőrző áramkörök, duplikáció, triplikáció stb. alkalmazása, valamint a kritikus egységek belső figyelése (monitoring). Hogy a sok lehetőség közül melyiket választjuk, azt az illető modul működése, ill. működésének kritikus volta határozza meg.

c) *Hibalokalizálás*

A helyreállítás sikere érdekében a hibát nemcsak észlelni kell tudnunk, hanem bekövetkezésének a helyét is ismernünk kell. Lényeges elvi különbség van tehát a hibadetektálás és -lokalizálás között. A helyreállító programmal vagy a karbantartó személyzettel közölnünk kell azt is, melyik modul szorul javításra.

d) *Helyreállítási akció*

Az akciónak a hiba észlelése, ill. bekövetkezési helyének meghatározása után kell indulnia. Ha a program nem tudja kijavítani a hibát, permanens hiba keletkezik. Permanens hibából valói helyreállítási eljárások:

- a hibás modul kicserélése egy tartalékkal,
- hibajavító modul alkalmazása,
- a hibás modul üzemen kívül helyezése, majd a működőképes modulok közti feladat-átcsoportosítás.

A hibajavító kódot általában memória-modulokban, ill. buszok figyelésére használják, a működőképes (aktív) modulok közötti feladat-átcsoportosítást pedig abban az esetben, ha nem áll rendelkezésre működőképes tartalék.

e) *„Hard core" védelem*

A dinamikus redundancia lényege, hogy a rendszer önellenőrző és önjavító tulajdonsággal rendelkezik. Az ezen feladatokat ellátó egységek megbízhatósága különösen fontos, azok meghibásodása nem engedhető meg. Ezeket az egységeket a rendszer „hard core" részének nevezzük. Védelmüket a drágább eljárások (pl. sztatikus redundancia) közül kell megválasztanunk:

- duplikálás kiegészítő felügyelettel,
- triplikálás és szavazás (TMR),
- hibrid redundancia, azaz TMR a hibás modulok cseréjének lehetőségével,
- komponens (esetleg alkatrész) szintű redundancia,
- permanens, huzalozott (wired-in) hibajavító kód.

f) *Modulok közötti kapcsolat*

A modulok közti kapcsolat megvalósításának kiválasztása döntő jelentőségű a dinamikus redundáns rendszerek esetén. A következő két lehetőség áll rendelkezésünkre:

- busz segítségével való kommunikáció,
- minden modul mindegyikkel külön-külön összekötve (point-to-point).

Az előbbi megoldás vonzóbb lehet, mivel kevésbé bonyolult, viszont a busz is hard core elem. Az utóbbi ezzel szemben költségesebb, és a rendkívül sok csatlakozás esetleg megoldhatatlan feladat elé állíthatja a tervezőket.

g) *A bemenet érvényesítése*

A bemenő jelek és adatok érvényesítése és/vagy hibakódos átkódolása minden redundáns rendszer számára fontos. Enélkül ugyanis hibás adatokat olvashatunk be, ami a belső redundancia ellenére is hibás működést okozhat.

2.3. A sztatikus és dinamikus redundancia összehasonlítása

Már az eddigiek alapján is látható, hogy egy hibatűrő rendszer tervezőjének rendkívül sok lehetőség áll rendelkezésére (eddig csupán a hardverről esett szó), ugyanakkor azonban nagyon nehéz feladat az optimális megoldás kiválasztása. Célszerű ezért röviden áttekinteni a sztatikus és dinamikus redundancia jellemző tulajdonságait.

A dinamikus redundancia előnyei a sztatikussal szemben:

- A modulok nagyobb elszigeteltsége, ami a független hiba feltételének teljesülése szempontjából fontos.
- A rendszer életben marad mindaddig, amíg egy modultípusból minden tartalék ki nem fogy.
- A tartalékelemek számának és típusának szabályozhatósága.
- A tápfeszültséggel el nem látott tartalékok potenciálisan kis hibavalószínűségének kihasználása.
- A sztatikus redundancia áramköri problémáinak elkerülése (ezek pl.: ki- és bemeneti terhelések, a tápfeszültség teljesítményének növekedése).
- A tartalékelemek ellenőrzésének könyebbsége az állandó diagnosztizáló program segítségével.
- A rendszer hibatűrő képességének elvileg állandó szinten tarthatósága.

A sztatikus redundancia előnyei a dinamikussal szemben:

- Egyszerűbb alkalmazás: a nem redundáns tervezés egyszerű, szisztematikus konverziója.
- Azonnali hibamaszkolás: nincs késleltetés, ill. működésmegszakítás a helyreállítás számára.
- A rendszer minden részének egyforma védelme: a hard core és a kapcsoló áramkörök védelmére nincs külön követelmény.

Ez utóbbi tulajdonság csak a tervezés szempontjából előnyös, egyes modulokkal, alkatelemekkel szemben viszont esetleg túl szigorú követelményeket támasztunk.

A felsorolt tulajdonságok alapján jól látható, hogy egy hibatűrő rendszer tervezésekor nem szorítkozhatunk csupán a dinamikus vagy a sztatikus redundancia által nyújtott lehetőségek kihasználására. A kettő kiegyensúlyozott alkalmazása biztosítja a hibatűrő tulajdonságok lehető legoptimálisabb megvalósítását.

2.4. Szoftver redundancia

A szoftver redundancia további programok, programszegmensek, utasítások és mikroprogramlépések alkalmazása, melyekre hibátlan hardver esetén nem lenne szükség.

A szoftver redundancia három fő formája:

a) *A kritikus programok és adatok többszörös tárolása*

A hiba utáni helyreállítás csak akkor lehet sikeres, ha a kritikus programok és adatok a hiba által okozott megszakítást, ill. zavart „túlélik". A legtöbb általános célú rendszer duplikált háttértárakat használ másodlagos memóriaként: diszk, mágnesdob, mágnesszalag. Ezek lehetővé teszik, hogy a hiba következtében károsodott programok és adatok az

újrafuttatáshoz eredeti állapotukban rendelkezésre álljanak. A háttértár adatbázisa periodikusan újratöltődik, és a kezdeti értéket szolgáltatja, ha az adat elvész a memóriából. A kritikus programok védelmére általában duplikált ROM-okat használnak, mivel a nem változtatható tárolás és a nemdestruktív kiolvasás jó lehetőség a kritikus programok és adatok tárolására [5, 6, 7].

b) *Tesztelő és diagnosztizáló programok program és mikroprogram szinten*

Ezeknek a programoknak vagy mikroprogramoknak a célja

- a hiba detektálása, lokalizálása, terjedelmének meghatározása;
- a hardver (tartalék is) periodikus tesztelése;
- a tárolók adatai állandóságának és teljességének ellenőrzése;
- a szoftver ellenőrzésének végrehajtása.

Kiváló példája a programok hierarchiája alkalmazásának az ESS No. 1. tárolt programvezérlésű telefonközpont [8]. Ez is és általában minden számítógépes rendszer tartalmazza a tesztelő, ellenőrző vagy diagnosztizáló programok valamelyik formáját.

A mikrodiagnosztizáló program a számítógép hardverének tesztelésére szolgáló speciális mikroprogram. Előnyei a következők:

- a rendszer egy kis részének kell csak hibamentesnek maradnia a mikrodiagnózis kezdeményezéséhez;
- a tesztek nagyobb felbontása érhető el a mikroműködés vizsgálatával, ezért redukálódnak a tároló- és időkövetelmények.

A mikrodiagnosztikák rezidens és nem rezidens részeket tartalmazhatnak. A nem rezidens rész a perifériális tárolóból olvasható be, miután a viszonylag kis rezidens rész leellenőrizte a működéshez szükséges egységeket. A mikroprogramozott vezérlés használata a modern rendszerekben lehetővé tette a diagnosztizáló programok mikrodiagnózissal való egyre nagyobb méretű pótlását.

c) *A programvégrehajtás hibatűrő képessége*

A hibatűrő működés legmagasabb szintű vezérlése az operációs rendszer felügyelő programjában (supervisor) található. Ez alól csak az olyan rendszerekben van kivétel, amelyek egyszerű sztatikus hibatűréssel rendelkeznek (TMR, komponens-redundancia stb.). A modern rendszerekben a hardver-hibadetektálást és -helyreállítást specális egységek vezérlik (Test and Repair Processor, Recovery Control Unit stb.). Ezek az egységek tájékoztatják és/vagy vezérlik a felügyelő programot a helyreállítási folyamat alatt, melynek megvalósítása nagyon változó; a helyreállítás hardver úton való végrehajtásától kezdve egészen a nagyon erős szoftver-függésig. A hibatűrő funkciók végrehajtásához a következők szükségesek:

- hibakollekció,
- a diagnózis végrehajtása,
- hibaanalízis és -jegyzék,
- hibastatisztika készítése,
- újjászervezés (rekonfiguráció),
- ellenőrzőpont- (checkpoint-) hálózat,
- csatorna- és I/O-műveletek helyreállítása,
- memóriamásolás,
- az állapotvektor tárolása,
- az újrakezdési (restart) pont vezérlése a felhasználói program számára stb.

2.5. Idő redundancia

A redundanciának ez a formája a különböző szintű utasítások (mikroutasítások, gépi utasítások, programszegmensek vagy egész programok) ismétlését vagy elfogadását tartalmazza. Általában a dinamikus hardver és szoftver redundanciával együtt alkalmazzák. Két külön célja van az idő redundanciának:

- hibadetektálás a végrehajtás megismétlésének vagy elfogadásának segítségével,
- helyreállítás programrestarttal vagy a művelet visszahívása hibadetektálás, ill. rekonfiguráció után.

a) *Hibadetektálás*

A program végrehajtásának megismétlése valószínűleg a legrégibb formája a hibadetektálásnak. Bár alkalmas a tranziens hibák detektálására, hatékonyságát korlátozza az a tény, hogy permanens hiba esetén következetesen hibát produkál, ezért összehasonlításkor elmarad a hibajelzés. Ezt a problémát az utasítások részben más sorrendjével vagy a hardver ismétlés alatti átszervezésével lehet áthidalni. Az adattovábbítás és -elfogadás különböző formáinak használatát (handshaking) kiterjedten alkalmazzák az általános célú rendszerekben, különösen a másodlagos (háttér-) tárolók, csatornák, I/O eszközök hibadetektálására.

b) Újraindítás

Az idő redundanciát egyaránt használják tranziens hiba okozta zavarok felismerésére és javítására, valamint a hardver-rekonfiguráció utáni programrestart esetén. Ez az utasítások, programszegmensek vagy teljes programok hibadetektálás utáni megismétlését jelenti. E három ismétlési eljárás tervezése a következő tényezők függvénye:

- Milyen messze lehet vagy kell a programot restartolni?
- Milyen költséges (idő, hardver, programozási kényszer) az ismétlés?
- Milyen valószínű a zavar sikeres javítása, ha azt tranziens hiba okozta?

Egyszeres utasítások ismétlése esetén minden utasítás eredményét védett tárolóban kell őrizni a következő utasítás végrehajtásáig. A másik két esetben járulékos programozási követelmények lépnek fel:

- a program szegmentálása,
- az ismétlési pont meghatározása,
- az ismétlési cím felülírása előtti állapotvektor tárolása stb.

A járulékos hardver tartalmazza a védett tárolókat az ismétlési cím és az állapotvektor számára.

3. A megbízhatóság modellezése és becslése

Hogy a tervezés során a redundáns módszerek között választani tudjunk, az egyes módszerek hibatűrő tulajdonságainak bizonyítása szükséges. E bizonyítás számszerű mennyiségek becslésével, számításával valósítható meg.

A megbízhatóság meghatározása kimutathatja az eredeti tervezés elégtelenségeit, ilyenkor további hibatűrő képességek hozzáadásával érhető el javulás. Az eljárást természetesen addig kell ismételni, míg teljesen kielégítő konstrukciót nem kapunk.

Két elvi mennyiségi jellemzőt szoktak meghatározni:

— a megbízhatóságot (az állandó hibák figyelembevételével) és
— a túlélést (tranziens hibák esetén).

A hibatűrő képességek becslésének két módja terjedt el. Az analitikus megközelítés során a rendszer hibatűrő képességének paramétereit annak matematikai modellezésével kapjuk. A kísérleti megközelítés esetén hibát okozunk a rendszer hardver-prototípusában vagy annak szimulált modelljében, és a hibatűrő képességeket a statisztikai adatokból határozzuk meg.

3.1. Az analízis alapmennyiségei

Egy rendszer hibatűrésének legáltalánosabban használt mértéke az MTBF (mean time between failures = = hibák közti átlagos idő), amely az alábbi módon származtatható:

$$\mathrm{MTBF} = \int_0^{\infty} R(t)\,dt.$$

A nem redundáns rendszerek vagy azok elemeinek megbízhatósága a jólismert $R(t)=e^{-\lambda t}$ függvénnyel írható le, ahol λ = meghibásodási tényező (failure rate) és $R(\varnothing)=1$.

Ha $R(t)=e^{-\lambda t}$, akkor $\mathrm{MTBF}=\frac{1}{\lambda}$.

Az MTBF-ek összehasonlítása tehát a rendszerek λ-inak összehasonlítását jelenti. Redundancia esetén R(t) polinomiális $e^{-\lambda t}$-re és a különböző rendszerek R(t) görbéi több helyen metszik egymást, ezért nem dönthető el egyértelműen, melyik a jobb. Emiatt további, speciális jellemzők bevezetése szükséges.

Ha adott egy állandó T_M (mission time = működési idő), akkor két vagy több rendszer összehasonlításához csak az $R(T_M)$ értékekre van szükség.

A megbízhatóság-javulási faktor (RIF = reliability improvement factor) egy új rendszer megbízhatóságának javulását mutatja a régiéhez képest.

$$\mathrm{RIF}=\frac{1-R_{\text{régi}}(T_M)}{1-R_{\text{új}}(T_M)}.$$

Abban az esetben, ha nem adható meg a T_M állandó, az MTIF (mission time improvement factor = = működési idő növekedési faktor) szolgál összehasonlításul:

$$\mathrm{MTIF}=\left.\frac{T_{\text{új}}}{T_{\text{régi}}}\right|_{R_{MIN}}, \quad \text{ahol}$$

R_{MIN} egy specifikált megbízhatóság, $T_{\text{új}}$ és $T_{\text{régi}}$ pedig azok az idők, mialatt $R_{\text{új}}$ és $R_{\text{régi}}$ (rendszermegbízhatóság) várhatóan R_{MIN}-re csökken.

3.2. Sztatikus megbízhatósági modellek

A megbízhatóság modellezésének ez az osztálya a sztatikus hardver-redundáns rendszerek megbízhatóságának becslésére szolgál. E redundancia jellegénél fogva az elemek állandó csatolásúak, saját meghibásodási tényezőjük van, azonnal végrehajtják a hiba maszkolását, amely sikerességének valószínűségét egységnyinek, a hibákat pedig függetleneknek feltételezzük. Ilyen feltételek mellett a sztatikusan redundáns rendszerek megbízhatósága a nem redundáns rendszerekéből számítható. Ha a szimplex rendszer megbízhatósága R, akkor

$$R_{\text{duplex}}=R^2+2R(1-R),$$

$$R_{TMR}=[R^3+3R^2(1-R)]R_V,$$

R_V = a szavazó elem megbízhatósága.

Ez utóbbi kifejezésből nagyon jól látható, hogy szavazásos eljárás esetén a rendszer megbízhatóságát döntően befolyásolja a szavazó elem megbízhatósága. Annak hibája nem engedhető meg, ezért hard core.

3.3. Dinamikus megbízhatósági modellek

A dinamikus redundanciához hibadetektálás és helyreállítás kell. A sztatikus megbízhatósági modellek dinamikus esetben való alkalmazása mindkettő sikerességét egységnyi valószínűségűnek feltételezi. Ennek megfelelően nagyon nagy megbízhatóságot lehet elérni a tartalékok számának növelésével. Általános követelmény, hogy a dinamikus modellnek reprezentálnia kell a teljes hibatűrő rendszert, azaz

— különböző meghibásodási tényezőket működő és tartalék modulokhoz,
— minden modul tartalékainak számát,
— a nem tökéletes hibadetektálást és helyreállítást,
— a hard core védelem megvalósítását,
— a modulokon belüli hibatűrést,
— a várható hibák kiterjedését és értékét,
— a várható tranziens hibák időtartamát és eloszlását.

Látható, hogy a dinamikusan redundáns rendszerek analitikus modellezése viszonylag bonyolult a különböző paraméterek nagy száma miatt, amelyek az optimális termék kiválasztásakor változhatnak. Nagyon jó eszköz a megbízhatóság becslésére egy interaktív számítógépprogram, amely on-line végrehajtja a dinamikusan redundáns rendszer fontosabb paramétereinek finomítását.

Az úttörő a REL program volt, melyet APL nyelven írtak. Ha egy rendszer paramétereit specifikáljuk, adott T_M-hez megadja a megbízhatóságot. A számítás során figyelembe vehető a tökéletlen hibadetektálás és -javítás hatása is [9].

Egy másik korai kísérlet a CARE program, amelyet a JPL—STAR komputer-tervezettel kapcsolatban fejlesztettek ki [10]. Újabban erőfeszítéseket tesz-

nek általános és számítható hatékony modellek elérésére további APL-programok segítségével [11, 12].

A dinamikusan redundáns rendszerek hibatűrő képességeinek kvalitatív vizsgálatához nagyon jól használható azok gráfokkal történő leírása. A különböző modulokat a gráfok csomópontjai, a modulok közti kapcsolatokat pedig ágai reprezentálják [13, 14]. Újabban az önellenőrző és helyreállító programokat is gráfos leírással elemzik, tervezik [15, 16].

3.4. A megbízhatóság kísérleti becslése

Az analitikus becslés bonyolultsága, ill. eredményeinek ellenőrzése szükségessé teszi a megbízhatóság kísérleti becslését, amelynek két módja:

- kísérlet a prototípussal,
- szimuláció.

Bár alkalmazása költségesebb és időigényesebb, a kísérlet mégis előnyösebb, ha az analitikus modell nem képes helyesen leírni a rendszer struktúrájának komplexitását vagy a várható hibák természetét. A prototípussal való kísérlet jó példája az ESS No. 1 rendszer, melyhez hibakatalógust készítettek a prototípus felhasználásával [8].

A kísérlet azonban időigényes és nem mindig áll rendelkezésre a prototípus. Erre az esetre fejlesztették ki a különböző szimulációs nyelveket (SIMULA 67, GPSS), amelyeket elterjedten használnak a hibatűrő képesség és megbízhatóság vizsgálatához.

IRODALOM

[1] *Neumann János élete és munkássága* (A különböző tudományterületeken elért eredményeinek összefoglaló áttekintése.) Szerk.: Szentiványi Tibor Bp. 1979.

[2] *A. Avizienis:* Architecture of Fault-Tolerant Computing Systems. FTC—5, International Symposium on Fault-Tolerant Computing, Paris, France, June 1975. pp. 3—16.

[3] *A. Avizienis:* Fault-Tolerance: The Survival Attribute of Digital Systems. Proc. of the IEEE, Vol. 66, No. 10, Oct. 1978, pp. 1109—1125.

[4] *A. Aviziens:* Fault-Tolerant Computing — Progress, Problems and Procpects. Proc. of IFIP Congress '77, Toronto, Aug. 1977, pp. 405—420.

[5] *S. B. Akers:* Partitioning for Testability. FTCS—6, Int. Symp. on FTC, Pittsburgh, Pennsylvania, June 1976, pp. 121—126.

[6] *C. J. Jenny:* Process Partitioning in Distributed Systems. National Telecommunications Conference, Vol. 2, New York, 1977, pp. 31:1-1—31:1-10.

[7] *R. D. Roger, F. E. Fromm:* Quantification and Measurement of Fault Recovery Performance FTCS—7, The 7th Ann. Int. Conf. on FTC, Los Angeles, California, June 1977, pp. 209—210.

[8] *R. W. Downing, J. S. Nowak, L. S. Tuomenoksa:* No. 1 ESS Maintenance Plan. The Bell System Technical Journal, Vol. 43, No. 5, Part 1, Sept 1964, pp. 1961—2019.

[9] *W. B. Bouricius, W. C. Carter, D. C. Jessep, P. R. Schneider, A. B. Wadia:* Reliability Modeling for Fault-Tolerant Computers. IEEE Transactions on Computers, Vol. C—20, No. 11, Nov. 1971, pp. 1306—1311.

[10] F. P. *Mathur:* Automation of Reliability Evaluation Procedures through CARE — The Computer-Aided Reliability-Estimation Program. AFIPS Conference Proceedings, Vol. 41, Anaheim, California, Dec. 1972, pp. 67—77.

[11] X. *W. Ng, A. Avizienis:* A Unifying Reliability Model for Closed Fault-Tolerant Systems. Digest of the 5th Int. Symp. on FTC, France, June 1975.

[12] D. *A. Rennels, A. Avizienis:* RMS: A Reliability Modeling System for Self-Repairing Computers Digest of the 3rd Int. Symp. on FTC, Palo Alto, California, June 1973, pp. 131—135.

[13] *J. P. Hayes:* A Graph Model for Fault-Tolerant Computing Systems. IEEE Transactions on Computers, Vol. C—25, No. 9, Sept. 1976.

[14] *J. P. Hayes, R. Yenney:* Fault Recovery in Multiprocessor Networks. FTCS—8, The 8th Ann. Int. Conf. on FTC, Toulouse, France, June 1978, pp. 123—128.

[15] *Harmat László:* Multi-mikroprocesszoros struktúrák önellenőrzése és öndiagnosztikája. Kandidátusi értekezés, 1980.

[16] *A. D. Friedman, L. Simoncini:* System-Level Fault Diagnosis Computer, March 1980, pp. 47—53.

[17] *Veress Tibor:* Hibatűrő irányító rendszerek a kapcsolástechnikában. Szakmérnöki diplomaterv, BME—HEI, 1981.