JEDNA REALIZACIJA DETEKTORA KVARA U UPRAVLJAČKIM SISTEMIMA ROBOTA
A REALISATION OF FAILURE DETECTORS IN ROBOTS CONTROL SYSTEMS

Milun Jevtić, Bojan Jovanović, Sandra Brankov, Marko Cvetković, *Elektronski fakultet Niš*

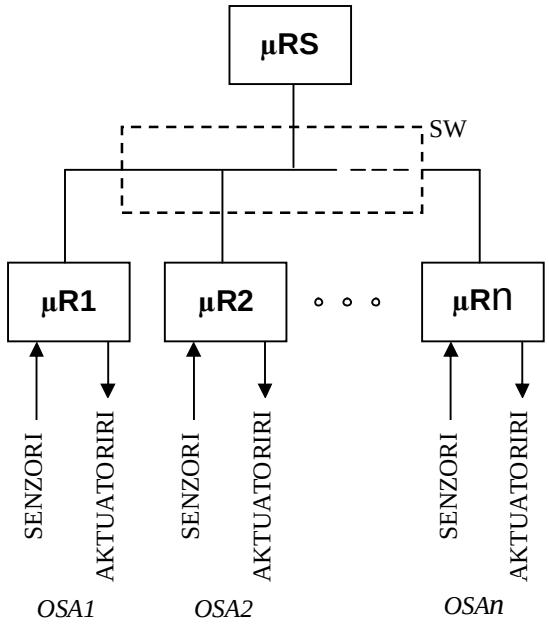
Sadržaj - *Za sigurno i bezbedno funkcionisanje robota predstavljena je jedna realizacija detekcije kvara. Kako su upravljački sistemi robota realizovani uglavnom kao višeprocesorski sa distribuiranom obradom zadataka, od interesa je realizovati u njima i savršene detektore otkaza. Savršeni detektori otkaza su detektori greške koji sa sigurnošću mogu da identifikuju računar (ili proces) koji je otkazao, i da nikada pogrešno ne posumnjaju. Takve detektore je moguće implementirati u sistemima gde računari imaju sopstvene hardverske satove i nadzorne tajmere, to jest i u takozvanim vremenski asinhronim sistemima.*

Abstract – *This paper presents one realization of failure detection for dependable and safe operation of a robots. Since the control systems of robots are implemented mainly as multiprocessing with distributed processing, it is important to realize in them a perfect failure detectors. Perfect failure detectors are failure detectors that can correctly identify crashed computer (or prcress) and never wrongly suspect a non-crashed computer. These detectors is possible to implement in asynchronous systems with computers having their own unsynchronized hardware clocks and hardware watchdogs, i.e. in so called timed asynchronous systems.*

1. UVOD

Za sigurno i bezbedno funkcionisanje robota neophodno je da njegova upravljačka jedinica ima ugrađene detektore otkaza. Kada se radi o projektovanju distribuiranih mikroračunarskih sistema visoke pouzdanosti [1][2], detekcija otkaza je ključni problem kojim projektant mora da se pozabavi.

Mada postoje roboti veoma različitih struktura i namena, generalno se može reći da su njihove upravljačke jedinice realizovane kao višeprocesorske sa distribuiranom obradom zadataka. Globalna arhitektura upravljačke jedinice robota prikazana je na slici 1.



Slika 1. *Arhitektura upravljačkog sistema robota*

Na višem hijerarhijskom nivou je mikroračunar **µRS** gde je realizovana interakcija između robota i operatera ili daljinskog nadzornog sitema. U njemu se obrađuju i aktiviraju globalni zadaci robota sa složenom trajektorijom kretanja, na osnovu kojih se definišu zadaci pojedinih mikroračunara na nižem hijerarhijskom nivou za upravljanje kretanjima po pojedinim osama robota. Bilo da je veza između računara u upravljačkom sistemu master-slave strukture ili zvezda strukture realizovane preko mrežnog switch-a (SW), od interesa je u njima realizovati i savršene detektore otkaza [3].

Detektovanje neispravnog mikroračunara u distribuiranom sistemu nije uvek jednostavno zbog činjenice da je izuzetno teško razlikovati proces koji je prestao da se izvršava, tj. otkazao, od veoma sporog procesa. Zbog toga su razvijene posebne tehnike za detekciju otkaza, kako bi se sa određenom sigurnošću moglo tvrditi da je dati proces zaista otkazao. Današnji distribuirani sistemi kao svoj sastavni deo imaju posebne module, tzv. detektore otkaza, koji imaju ulogu da detektuju otkaz nekog procesa ili računara. Posebnu grupu detektora otkaza čine savršeni detektori otkaza, koji će biti prikazani u ovom radu.

U radu će biti objašnjen pojam savršenih detektora otkaza, način njihove implementacije u vremenski sinhronim distribuiranim sistemima, i kako je moguća njihova primena u asinhronim sistemima. Posebno će biti razmotrena grupa asinhronih sistema gde računari imaju sopstvene nesinhronizovane hardverske satove i hardverske nadzorne (*watchdog*) tajmere u kojima je moguća implementacija savršenih detektora otkaza. Ovi, tzv. *vremenski asinhroni sistemi* su od posebnog interesa pri projektovanju i realizaciji industrijskih mikroračunarskih sistema za upravljanja procesa

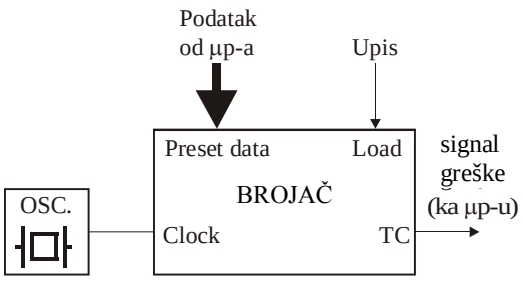
sa visokom pouzdanošću. U radu su ukratko opisane i najvažnije osobine protokola za detekciju otkaza, primenljivog kod ovih sistema

U procesu nadzora za detekciju toka kontrole koriste se nadzorni tajmeri (Watchdog) koji proveravaju vreme izvršenja programa. Nadzorni tajmeri se oddređenom sekvencom instrukcija mikroprocesora u aplikacionom programu periodično restaartuju (reinicijalizuju). Ako zbog otkaza restartovanje izostane unutar specificiranog vremena, nadzorni tajmeri generišu signale greške.

Pretpostavimo da nadzorni tajmer proverava donju (T-T_w) i gornju (T) granicu vremena izvršavanja (t_i) programskog segmenta (CS_i) a parametri T i Tw mogu biti fiksni za sve segmente. Može se proceniti verovatnoća p_{ij} (T_w) da će nadzorni tajmer otkriti pogrešnu granu segmenta CS_i u segmentu CS_j (sa vremenom izvršavanja t_j):

$$p_{ij}(T_w) = [(t_i + t_j - T)^2 + (T - T_w)^2] / (2 t_i t_j) .$$

Optimalna situacija je kada je $t_i=t_j=T-T_w/2$. Smanjivanjem T_w, možemo postići sasvim dobro pokrivanje greške. Međutim ovo zahteva znanje o vremenu izvršavanja programa (procenjeno specijalnim programima). Najpopularniji nadzorni tajmeri proveravaju samo gornju granicu ($T_w=T$). U ovom slučaju, maksimalana vrednost od p_{ij} ($T_w=T$) iznosi 0,5. Jedan takav nadzorni tajmer prikazan je



na slici 2.

Slika 2. *Nadzorni tajmer*

Signal greške pored signalizacije ka nadređenom nadzornom sistemu može i da u mikroračunarskom sistemu u kome je implementiran izazove određenu akciju. Naime delujući na nemaskirajući prekid (interrupt) ili reset logiku za restartovanje mikroračunara, može se inicirati ponovni pokušaj ili na određen način obraditi - odbaciti proces u cilju prevazilaženja nastale greške. Naraavno, strategija prevazilaženja greške u mnogome zavisi i od toga da li je nastali otkaz permanentan ili prolazan. Jedno konkretno rešenje modula sa većim brojem hardverski realizovanih nadzornih taajmera za *on-line* nadzor većeg broja procesa koji se simultano odvijaju u jednom *hard real-time* sistemu dato je u radu [4].

2. SAVRŠENI DETEKTORI OTKAZA

Savršeni detektori otkaza su komponente koje sa sigurnošću mogu da donesu odluku da li je neki od računara u distribuiranom sistema otkazao, i istovremeno obezbeđuju da se nikada ne posumnja da je otkazao računar koji funkcioniše. Kao takvi su izuzetno pogodni za korišćenje u distribuiranim sistemima visoke pouzdanosti.

Većina postojećih detektora otkaza spada u grupu takozvanih *heartbeat*-detektora koji ne štite od pogrešnih pretpostavki da je neki računar prestao s radom, što može da dovede do pojave problema nekonzistentosti podataka. Ovu situaciju je najlakše prikazati na primeru sistema sa dva računara od kojih je jedan primarni, a drugi ima ulogu *backup*-računara. U ovakvom sistemu pogrešna pretpostavka *backup*-računara da je primarni računar otkazao vodi do situacije gde i jedan i drugi računar misle da su primarni. [5].

Prilikom rešavanja problema pogrešne pretpostavke ključno je naći rešenje koje sigurno obezbeđuje da je dati računar otkazao, pre nego ostali računari posumnjaju da je otkazao. To znači da, ako se posumnja da je računar otkazao, taj računar bude ili isključen ili izolovan od ostalih računara. Poseban problem se javlja u slučaju da je došlo i do nekog otkaza u mreži, pošto onda nije moguće inicirati isključivanje tog računara od strane nekog od preostalih računara koji su u funkciji, zbog nemogućnosti komunikacije.

Savršeni detektori otkaza su prvenstveno bili namenjeni za detektovanje prestanka izvršavanja pojedinih procesa u sistemima gde nije bilo moguće da se procesi oporave posle prestanka rada. U ovom radu je od interesa detektovanje otkaza računara u sistemima gde računari mogu da se oporave posle otkaza, kao i detektovanje otkaza u specificiranom vremenu. Akcenat je stavljen na vremenske-savršene detektore otkaza definisane u [6]. Ovakav detektor otkaza donosi ispravnu odluku da li je dati računar otkazao ili ne, čak iako je taj računar trajno ili privremeno isključen sa mreže.

3. POTPUNO SINHRONI I ASINHRONI SISTEMI

U ovom poglavlju definisaćemo sinhrone i asinhrone distribuirane sisteme i razmotriti mogućnost implementacije savršenog detektora otkaza u ovim sistemima.

Sinhroni distribuirani sistem je distribuirani sistem kod koga je poznato maksimalno moguće kašnjenje u komunikaciji i obradi, i gde je jedina mogućnost otkaza da računar prestane s radom. Pretpostavićemo zbog jednostavnosti da po prestanku rada računar ne može da se oporavi. U ovakvom sistemu postoji unapred poznata vremenska konstanta *T* koja predstavlja vreme potrebno da jedan računar primi i obradi poruku od nekog drugog računara. To znači da je ukupno vreme potrebno da jedan računar pošalje poruku drugom računaru, pod pretpostavkom da nijedan od ova dva računara nije otkazao, i da primi i obradi odgovor od tog računara, iznosi 2*T*.

Detektor otkaza (DO) u ovakvom sistemu se sastoji od skupa modula (funkcija) za detektovanje otkaza, gde se svaki modul DO_i izvršava na jednom računaru. Proces sa nekog *i*-tog računara može da pozove svoj DO_i modul da bi utvrdio da li DO_i sumnja da je neki drugi računar prestao sa radom. Ovde ćemo ukratko objasniti *SynchPerfect* protokol za savršeni detektor otkaza u sinhronim sistemima.

SynchPerfect protokol prikazan na slici 3. (*Synch* zato što se radi o sinhronom sistemu, a *Perfect* pošto implementiramo savršeni detektor otkaza) radi na sledeći način: Kada proces sa datog računara želi da utvrdi da li je drugi računar prestao sa radom, on poziva funkciju *SynchPerfect*, koja šalje poruku AYA (*Are You Alive*) željenom računaru. Posle toga ovaj modul čeka najmanje 2*T* vremenskih jedinica na odgovor od drugog računara. Ukoliko u tom intervalu dobije odgovor IAA (*I Am Alive*) funkcija vraća vrednost UP koja ukazuje da upitani računar nije prestao sa radom. U suprotnom, SynchPerfect ispravno sumnja da je upitani računar prestao sa radom i vraća vrednost CRASHED. U sinhronim sistemima, ovakav protokol detektuje sve računare koji su prestali sa radom, i nikada ne sumnja ni na jedan računar koji nije prestao sa radom.

```
function SynchPerfect(computer c)
   send AYA to c;
   wait
      on receive IAA from c
           return UP;
      after 2*T
           return CRASHED;

main
   forever
      on receive AYA from sender
           send IAA to sender;
```

Slika 3. *Protokol savršenog detektora otkaza za potpuno sinhrone sisteme*

Međutim, većina distribuiranih sistema nije sinhrona već asinhrona, gde je situacija bitno drugačija. Tu ne možemo da budemo sigurni da će jedan računar dobiti odgovor od drugog računara u specificiranom vremenu, ma kako dug bio interval 2*T* tokom koga se čeka odgovor.

Asinhroni sistem se može definisati kao sistem koji nema gornju granicu kašnjenja u prenosu poruka, niti ima gornju granicu kašnjenja obrade. Osim toga ovi sistemi nemaju sat realnog vremena. S druge strane svaki savršeni detektor otkaza je *potpun* i *precizan*. *Potpun* znači da će detektovati sve računare koji su prestali s radom, a *precizan* znači da ni u kom slučaju neće pogrešno posumnjati da je neki računar prestao s radom. U [6] je pokazano da je nemoguće implementirati takav, savršeni detektor otkaza, u potpuno asinhronom sistemu. Međutim, postoji mogućnost implementacije savršenog detektora otkaza u tzv. vremenskim asinhronim sistemima, koje ćemo definisati u nastavku.

Kod vremenskih asinhronih distribuiranih sistema takođe ne postoji gornja granica kašnjenja u prenosu poruka niti u obradi. Osim toga računari mogu trajno ili privremeno biti izdvojeni iz mreže. Svaki računar ima svoj lokalni hardverski sat i ti satovi nisu globalno sinhronizovani. Hardverski sat omogućava merenje lokalnog vremena sa poznatom tačnošću. Ovakav sistem proširen je hardverskim nadzornim tajmerima (*watchdog*), da bi se omogućilo da se računar koji ne daje odziv resetuje čim hardverski sat dostigne specificiranu vrednost.

U ovakvom sistemu kod računara može da se javi otkaz u smislu prestanka rada ili otkaz u smislu narušenog rada. U svakom slučaju smatramo da računar ili radi ili je otkazao, ali da, uz to, još i može da se oporavi posle otkaza. Otkaz u smislu narušenog rada se definiše kao nemogućnost nekog računara da završi započeto izvršenje date procedure u okviru specificiranog vremena, gledano od trenutka koji je planer odredio za početak izvršenja. Pored toga računari mogu komunicirati međusobno razmenom poruka tako da i tu postoji mogućnost otkaza. Otkazi vezani za komunikaciju su: poruka uopšte nije prenesena, ili, poruka je prenesena, ali je njeno kašnjenje veće od neke unapred izabrane konstante.

Hardverski nadzorni tajmer je veoma bitan element u ovakvom sistemu. Interfejs hardverskog nadzornog tajmera se sastoji od jednog registra koji sadrži vrednost praga. Ovaj modul će resetovati računar u čijem je sastavu, u slučaju da vrednost hardverskog sata dostigne ovaj prag. Dakle, ukoliko se *watchdog*-prag ne obnavlja periodično, računar će prestati s radom. Ovde treba dodati da se računar može oporaviti posle prestanka rada, tj. hardverski nadzorni tajmer će se iskoristi za restartovanje računara. Prilikom ponovne inicijalizacije, vrednost praga nadzornog tajmera se postavlja na vrednost koja je za poznatu konstantu veća od vremena u kome se računar ponovo inicijalizovao, tako da računar ima tačno toliko vremena da osveži svoj nadzorni tajmer, tj. da postavi novu vrednost praga.

4. PROTOKOL ZA DETEKCIJU OTKAZA

Protokol za savršenu detekciju otkaza predložen u [6] sastoji se iz dva nivoa. Niži nivo obezbeđuje da se samo povremeno pogrešno posumnja da je računar otkazao, čime se minimizuje broj pogrešnih pretpostavki. Na višem nivou se obezbeđuje da detektor otkaza svojim klijentima daje informaciju da je određeni računar otkazao tek onda kad je taj računar stvarno otkazao. To se obezbeđuje pomoću *lease*-metode i nadzornih tajmera [6]. Protokol je potpuno simetričan tako da svi čvorovi u mreži izvršavaju isti kôd protokola. Osim toga ovaj protokol se može koristiti i za de-tekciju otkaza procesa, a ne samo otkaza računara. Jedino ograničenje ovog algoritma je da je ograničen na mali broj računara, tako da, ako neki računar hoće da detektuje otkaz više od dva računara, moras da pokrene više instanci istog protokola.

Osnovni aspekti predloženog protokola su *lease*-ekstenzije, distribuirani *snapshot* i *fail-aware* datogram. U nastavku ćemo ukratko objasniti smisao ovih pojmova.

Lease (eng. *lease* – najam, zakup) je informacija u smislu *dozvole*, koju jedan računar šalje drugom da bi mu dozvolio promenu vrednosti praga nadzornog tajmera. Na ovaj način dati računar, koji je u unapred određenom vremenu, tačnije vremenu pre nego što je njegov hardverski sat dostigao vrednost prethodnog postavljenog praga nadzornog tajmera, dobio *dozvolu* od makar jednog od ostala dva računara, može da osveži svoj nadzorni tajmer i samim tim nastavi s radom. Ukoliko računar ne dobije *dozvolu*, neće moći da osveži svoj nadzorni tajmer, a ovaj će zatim resetovati svoj *host*-računar. Sam postupak komunikacije je sledeći: Računar *a* želi da osveži prag svog nadzornog tajmera, tj. želi da produži *dozvolu*, tako da šalje zahtev računarima *b* i *c*. Ukoliko *b* (ili *c*) primi zahtev, on garantuje *dozvolu* računaru *a*, do unapred

definisanog trenutka u realnom vremenu. Tada *b* šalje povratnu poruku računaru *a*, a kad *a* primi tu poruku, onda i osvežava svoj nadzorni tajmer, tj. produžava period *dozvole* za neku unapred poznatu vrednost, tako da je po njegovom lokalnom vremenu eksplicitno definisan trenutak do kada ta *dozvola* važi. Računar *b*, koji je garantovao *dozvolu*, takođe lokalno definiše vreme *lt[a]* (prema svom hardverskom satu) do kada traje *dozvola* koju je upravo dao računaru *a*. Na taj način obezbeđeno je da kada vreme lokalnog hardverskog sata računara *b* dostigne vrednost *lt[a]*, sigurno je i vrednost lokalnog sata računara *a* dostigla vrednost praga nadzornog tajmera, što znači da je *dozvola* istekla. Nadzorni tajmer računara *a* garantuje da će *a* biti restartovan onog momenta kad mu istekne *dozvola*.

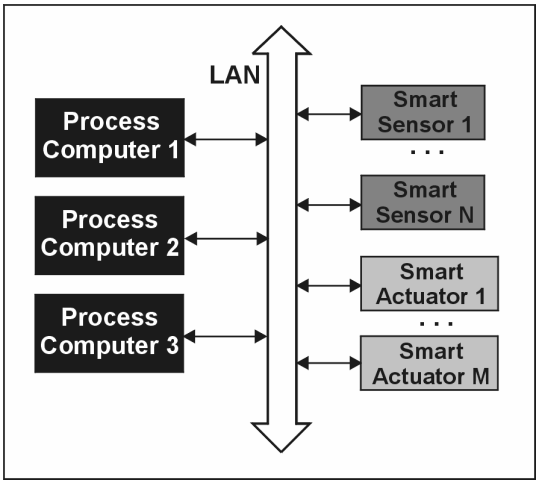
Distribuirani *snapshot* nam s druge strane omogućava da kad jedan računar otkaže, pošto mu je istekla *dozvola*, ostali računari saznaju da je dati računar otkazao. Jedan računar donosi odluku da je drugi računar otkazao samo ako je siguran da je istekla *dozvola* koju je dao tom računaru i da je istekla *dozvola* koju je treći računar dao tom istom računaru. Ovaj algoritam praktično utvrđuje da li je postojao trenutak kada dati računar nije imao ni jednu *dozvolu*. Računar *a* može da utvrdi da li je računar *b* otkazao iz komunikacije sa trećim računarom *c*. Tada *a* šalje zahtev računaru *c* u datom trenutku u realnom vremenu. Računar *c* šalje odgovor koji sadrži dve informacije koje se tiču vremena: jedna predstavlja trenutno vreme po njegovom lokalnom satu u trenutku slanja odgovora - T_R, a druga je vreme po njegovom lokalnom satu kada bi trebalo da istekne *dozvola* koju je dao računaru *b* - TL_b. Kada računar *a* primi odziv od *c*, upoređuje vremena T_R i TL_b i ako je T_R veće to znači da je *dozvola* koju je *c* dao računaru *b* istekla. Ukoliko je istekla i *dozvola* koju je *a* dao računaru *b*, što *a* može da utvrdi na osnovu vrednosti svog lokalnog sata, onda *a* ispravno donosi odluku da su sve *dozvole* date računaru *b* istekle i da je samim tim on otkazao.

Fail-aware datogram je veoma značajan za vremenski asinhrone sisteme pošto omogućava da se odredi vremensko kašnjenje u prenosu poruka. Detaljno je definisan u [5], a ovde ćemo samo ukratko objasniti njegovu ulogu. Osnovna uloga *fail-aware* datograma je da eliminiše hazard u smislu sigurnosti (eng. *safety hazard*) prouzrokovan sporim porukama i sporim procesima, koji nose informaciju koja je suviše stara da bi se smatrala validnom. Zbog toga je neophodno da se obezbedi da učesnici u komunikaciji koriste samo informacije koje su "sveže", tačnije informacije koje su stigle do odredišta sa kašnjenjem koje je manje od nekog unapred određenog broja vremenskih jedinica. Ukoliko do određenog čvora u mreži stigne poruka koja je zakasnila više nego što je dozvoljeno, onda taj čvor odbacuje tu poruku. Ovakav servis se može sam po sebi iskoristiti za brojne primene, od kojih je najčešća interna sinhronizacija satova u distribuiranim sistemima sa višestrukom redundansom, u cilju definisanja globalne vremenske rezolucije [7]. Međutim, kada je u pitanju savršena detekcija otkaza, ovaj servis se koristi na malo drugačiji način nego što je definisano u [7]. U suštini, *fail-aware* servis šalje zahtev nekom procesu periodično sve dok ne dobije potvrdu ili dok ne istekne neko unapred definisano vreme. Ako *time-out* istekne, on prestaje sa slanjem poruke. Na ovaj način se obezbeđuje da računar *a* može da odbaci zahtev drugog računara *b*, u kome *b* od računara *a* traži *dozvolu* za osvežavanje nadzornog tajmera, ukoliko je poruka previše dugo putovala od *b* do *a* (pošto to može da znači da je računar *b*, koji traži *dozvolu*, već otkazao).

U ovom poglavlju su ukratko prikazani suštinski aspekti protokola savršenog detektora otkaza. Sam prorokol je složeniji jer uzima u obzir i odstupanje i košenje takta, kao i zakasnele i neisporučene poruke [6].

5. PRIMENA U INDUSTRIJSKIM SISTEMIMA

Ovako definisan detektor otkaza može se iskoristiti u industrijskim upravljačkim sistemima kod kojih postoji zahtev za visokom pouzdanošću. Otpornost na otkaz ovde u svakom slučaju ne znači da se radi o sistemima kod kojih je eliminisana pojava otkaza, već da sistem, pri pojavi jednog ili većeg broja otkaza, i dalje obavlja svoju funkciju, a otkaz ili zanemaruje, ukoliko se radi o prolaznom otkazu, i nastavlja dalje s radom, ili se vraća u stanje pre pojave otkaza i ponavlja izračunavanja.



Slika 4. *Principijelna blok šema industrijskog sistema*

Princip moguće primene opisanog protokola prikazan je na slici 4. Prikazan je najjednostavniji slučaj sa tri procesna računara koja su međusobno povezana putem LAN mreže, a preko iste mreže komuniciraju sa senzorima i aktuatorima. Senzori očitavaju vrednosti pojedinih veličina nekog fizičkog procesa, a njih povremeno ili periodično očitavaju procesni računari. Procesni računari na osnovu očitanih stanja senzora i obrade tih informacija donose odluke o slanju odgovarajućih komandi aktuatorima. Ukoliko je fizički proces kritičan, neophodno je obezbediti da sistem potpuno bezbedno funkcioniše ukoliko je neki njegov deo otkazao, pošto bi u suprotnom postojala opasnost po okolinu. Ukoliko pretpostavimo da se radi o sistemu sa trostrukom redundansom, jasno je da je neophodno obezbediti da ukoliko neki od računara otkaže (otkaz u smislu prestanka rada) ili mu je rad ozbiljno poremećen (otkaz u smislu performansi), neophodno je da ostala dva računara to saznaju na vreme, i da se pri tom nikada ne desi da se pogrešno posumnja da je računar otkazao. Ukoliko se desi da neki računar pogrešno posumnja da je drugi (primarni) računar otkazao, celokupni sistem može preći u neregularno stanje, koje može biti opasno po okolinu (npr. dva računara istovremeno pokušaju da pošalju komandu istom aktuatoru). Izbegavanje ovakvog scenarija upravo obezbeđuje protokol opisan u prethodnom

poglavlju. Dakle, ako jedan računar otkaže, neki od preostala dva (ili oba) će privremeno preuzeti njegove funkcije, sistem neće prestati s radom, i istovremeno će i dalje biti bezbedan po okolinu, pri čemu može da pređe u rad sa redukovanim funkcijama sistema.

Za ovako definisan sistem može se iskoristiti još jedna mogućnost pomenuta u prikazu protokola, samo sada sa nešto drugačijom funkcijom. Naime, *fail-aware* servis omogućava svim uređajima u sistemu da utvrde da li je maksimalno izračunato kašnjenje poruke u komunikaciji manje od neke unapred zadate vrednosti. Na taj način moguće je i utvrditi da li je npr. informacija sa senzora "sveža", tj. da li u datom trenutku raspolažemo validnom informacijom o datoj veličini. Prijemnik će u tom slučaju moći da informaciju odbaci, ili uradi nešto drugo, zavisno od nivoa apstrakcije tog dela sistema.

Prikazani protokol je takođe moguće primeniti i za slučaj da nije potrebno detektovati otkaz čitavog računara, već otkaz procesa koji se izvršavaju na jednom ili više računara. U ovom slučaju isti princip se može primeniti u implementaciji savršenog detektora otkaza procesa. Osnovna razlika u odnosu na osnovnu primenu je da su hardverski nadzorni tajmeri zamenjeni procesnim (softverskim) nadzornim tajmerima, koji za razliku od hardverskih, ne "ubijaju" sve procese na računaru, već samo jedan, onaj koji se ne izvršava. U [6] je pokazano da u sistemu opšte namene kao što je PC sa Linux operativnim sistemom, i pored povremenog javljanja kašnjenja u planiranju, proces nikada neće da izvrši instrukciju posle isteka krajnjeg roka definisanog u procesnom nadzornom tajmeru. Drugim rečima, proces će biti "ubijen" na vreme.

6. ZAKLJUČAK

U ovom radu dat je prikaz savršenih detektora otkaza u distribuiranim mikroračunarskim sistemima i njihova mogućnost primene u sistemima za upravljanje robotom. Kako je moglo da se vidi, implementacija ovih detektora u sinhronim distribuiranim sistemima je relativno jednostavna. Međutim, kako je većina sistema asinhrona (za asinhroni karakter je dovoljno i postojanje mogućnosti prekida komunikacije tokom proizvoljno dugog perioda), primena savršenih detektora greške nije uvek laka, pošto ih nije uvek moguće implementirati u asinhronim sistemima. S druge strane, ukoliko pretpostavimo da računari u asinhronom sistemu imaju sopstvene hardverske satove, koji ne moraju da budu međusobno sinhronizovani, a uz to im je pridodat još i hardverski nadzorni tajmer, implementacija savršenih detektora otkaza postaje moguća. Ovakav pristup ima poseban značaj u sistemima za upravljanje od kojih se zahteva visoka pouzdanost, sigurnost i odgovornost u radu, kakvi su po pravilu svi sistemi upravljanja robotima. U takvim primenama upravljanja kritičnim fizičkim procesima ne retko je neophodno obezbediti da prilikom otkaza dela sistema, sistem nastavi dalje sa radom, i da pri tom nema opasnosti po okruženje.

LITERATURA

[1] M. Jevtić, M. Damnjanović, "An Approach to Design for Testability in Hard Real-Time Systems," *Proc. of 21st International Conference on Microelectronics - MIEL'97*, Vol. 2, pp. 849-852, September 1997, Niš.

[2] M. Jevtić, M. Damnjanović "Testing of Digital Systems in Real-Time Applications", *Proc. 20th International Conference on Microelectronics - MIEL'95*, Vol.2, pp. 835-840, September 1995, Niš.

[3] Marko Cvetković, Milun Jevtić, *"Savršeni detektori otkaza u distribuiranim upravljačkim sistemima"*, Zbornik XLVII konferencije za elektroniku, telekomunikacije, računarstvo, automatiku i nuklearnu tehniku - ETRAN 2003, Herceg Novi, 8 - 13. juna 2003. Sveska I, str. 84-87.

[4] Milun Jevtić, Volker Zerbe, Sandra Brankov, *"Multilevel Validation of On-Line Monitor for Hard Real Time Systems"*, Proc. 24th International Conference on Microelectronics - MIEL 2004, Vol. 2, pp. 755-758, Ni{, Serbia and Montenegro, 16-19. may, 2004.

[5] L. Sabel and K. Marzullo, "Simulating Fail-Stop in Asynchronous Distributed Systems," *Proc. 13th Symp. Reliable Distributed Systems,* pp. 138-147, 1994.

[6] C. Fetzer, "Perfect Failure Detection in Timed Asynchronous Systems," *IEEE Trans. Computers,* vol. 52, No. 2, pp. 99-112, Februar 2003.

[7] C. Fetzer and F. Cristian, "Fail-Awareness in Timed Asynchronous Systems," *Proc. 15th ACM Symp. Principles of Distributed Computing,* 1996., Philadelphia