UNIVERZITET CRNE GORE
PRIRODNO-MATEMATIČKI FAKULTET
odsjek za matematiku i računarske nauke

Mr Stevan Šćepanović

ANALIZA JEDNE KLASE DISTRIBUIRANIH RAČUNARSKIH SISTEMA POSREDSTVOM IMITACIONOG MODELIRANJA

DOKTORSKA DISERTACIJA

PODGORICA, 2002.

Ovaj rad je realizovan pod rukovodstvom mentora
Prof. dr Miroslava Bojovića.

Analysis of a class of distributed computer systems by means of imitated modeling

Abstract

Intensive use of distributed computer systems and computer nets nowadays, actualises problems of their analysis, projecting and constructing. With the subject of this dissertation appears analysis and modelling of distributed computer systems built on the basis of personal computers. These are heterogeneous distributed systems in which computers don't share neither memory nor tact generator among themselves. The local computer net is used as a communicative environment, in respect to widespread use of such computer systems.

A mathematical model that describes functioning of distributed computer system is worked out in the dissertation and it principally differs from familiar models. In distinction from models known from literature, this work clasps not only algorithmic aspect of conducting of the applicative programme, but also characteristics of the hardware structure of the distributed computer system and programme, and time as a parameter. Within this mathematical model mutual relations of different aspects of parallelism inside the distributed computer system are being studied. There was an attempt to describe functioning of the applicative programme in a new way and, according the analysis of that conducting, to estimate the performances of the distributed systems. Mathematical apparatus of the theory of graphs and theory of automats were used while constructing the mathematical model.

The imitated model was developed together with the mathematical one in the work. Imitated model with messages was used to describe the conduction of the distributed systems. The conception of complex approach to the imitated modelling of distributed computer system was formulated. That kind of approach allows algorithm, as well as time characteristics of modelling to be checked within one unique system. Aiming at constructing the working load of imitated model, a method of analysis and estimation of the performances of distributed computer system was worked out, based on conducting of the applicative programme and autonomy of that conducting from the hardware structure.

The results of this research work can serve as starting base for development of a new perspective scientific method in researching the characteristics of conducting of distributed systems. An important characteristic of the proposed model is capability of estimated efficiency of distributed computer systems in the phase of their projecting, by which the costs and time for their constructing are reduced. It increases the quality of the project, allows checking relations of asked aims of the project and its abilities.

The practical importance of the work is seen in the constructing the mathematically correct method which:

- Allows estimation of system performances of the distributed computer system, without making the prototype or commands emulator of the system that is being analysed.
- Allows the prognosis of the applicative programme conduction in a new hardware environment during the projecting.
- Allows that the accuracy of the estimation stays among permitted borders.

Found theoretical results are checked on the distributed computer system based on the architecture client / server.

SADRŽAJ :

1. UVOD

1.1 Aktuelnost teme

1.2 Cilj rada

1.3 Doprinos rada

1.4 Pregled rada

1.1 Aktuelnost teme

Intenzivna primjena informacionih tehnologija i masovna upotreba računara, u današnje vrijeme, posebno aktualizira probleme izgradnje velikih računarskih sistema i računarskih mreža. Ove sisteme karakteriše teritorijalna rasprostranjenost uzajamno povezanih računara, koji uz pomoć odgovarajućeg softvera, međusobno sarađuju u procesu obrade podataka formirajući *distribuirani računarski sistem*, ili računarski sistema distribuirane arhitekture.

Brz razvoj distribuiranih računarskih sistema doveo je do saznanja da je ograničavajući faktor povišenja njihove efikasnosti upravo optimalan izbor njihove elementarne strukture i sastavnih komponenti. Ovo je nametnulo potrebu intenzivnijeg istraživanja u oblasti arhitekture distribuiranih računarskih sistema [1], [2], [3], [11], [12], [13], [14], [20]. Kao rezultat istraživanja javljaju se lokalne računarske mreže. Troškovi izgradnje lokalnih mreža, u čijim se čvorovima nalaze personalni računari, znatno su niži, nego ako se računarska mreža sličnih karakteristika i namjene izgrađuje uz pomoć neke druge klase računara. Prednosti lokalnih mreža personalnih računara kao što su: visoka sigurnost, relativno niska cijena, brza i laka izgradnja, mogućnost korišćenja već postojećeg softvera prirodno su nametnuli ideju izgradnje visokoproduktivnih računarskih sistema na bazi lokalnih računarskih mreža. U ovim sistemima se pretpostavlja postizanje visoke efikasnosti korišćenjem većeg broja računara koji rade paralelno na izvršavanju nekog zadatka.

Izgradnjom distribuiranih sistema postiže se bolje iskorišćenje resursa, veća raspoloživost, pouzdanost i fleksibilnost u radu, a dobijaju se i nove funkcionalne mogućnosti [4], [5], [6], [9], [11]. Međutim, zbog izuzetne složenosti kako hardverskih tako i softverskih komponenti sistema, pojavljuju se dodatni problemi koji nijesu bili prisutni kod računara klasične arhitekture kao što su: problemi kontrole i upravljanja distribuiranim sistemom, optimalnost distribucije računarske opreme, programa i podataka, problemi konkurentnog izvršavanje paralelnih programa i sl.

Dinamičko ponašanje distribuiranog računarskog sistema karakterišu različiti nelinearni fenomeni. Uzroci nelinearnog ponašanja sistema nalaze se u algoritmima korišćenim za upravljanje resursima sistema, raspoređivanje procesa, otkrivanje i ispravljanje grešaka, u načinu i stepenu distribucije programa i podataka a isto tako i u činjenici da tok saobraćaja kroz sistem varira na veoma različite i često nepredvidive načine. Direktno eksperimentisanje nad realnim distribuiranom sistemom zahtijeva velike finansijske troškove, a može biti sprovedeno tek u završnoj fazi projektovanja i instalacije distribuiranog sistema. U literaturi postoji tek nekoliko objavljenih radova, koji iznose eksperimentalne podatke o ponašanju aplikativnih programa u distribuiranoj računarskoj sredini. Svi ovi radovi daju algoritamsku analizu aplikativnih programa, bez analize njegovog ponašanja u vremenu. Koliko je poznato, još nije dato potpuno i detaljno teorijsko obrazloženje ponašanja aplikativnog programa u distribuiranoj sredini, niti zavisnost tog ponašanja od izmijene fizičke strukture sistema. Nije napravljen odgovarajući matematički, odnosno numerički model simulacije ponašanja aplikativnog

programa u distribuiranoj računarskoj sredini. Takav model bi trebao da uključi, kako razradu mnogobrojnih hardverskih i softverskih aspekata distribuiranog računarskog sistema, tako i uticaj opterećenja mreže na efikasnost rada sistema. Model bi pored teorijskog imao i praktični značaj, jer bi između ostalog, odredio opsege promjena parametara sistema, koje bi trebalo da proizvedu željene performanse sistema.

Postojeće metode i sredstva analize i projektovanja distribuiranih sistema obično daju grube aproksimacije kvalitativne ocjene karakteristika sistema. Povišenjem tačnosti dodatno se značajno povećavaju gubici i troškovi istraživanja i projektovanja. Izgradnja sredstava za tačnu procjenu, po pravilu, je usko specijalizovana i mora se izgrađivati svaki put ponovo. Osnovni uzroci toga leže u činjenici da:

- za distribuirani računarski sistem ne postoji teorija sličnosti, koja bi dozvolila određivanje koeficijenata proporcionalnosti između malog i velikog distribuiranog računarskog sistema, između prototipa i realnog sistema, između homogenog i heterogenog sistema,
- efikasnost distribuiranog sistema nelinearno zavisi od njegovih karakteristika takvih kao što su parametri hardverskih i softverskih komponenti sistema, struktura i tip korišćene računarske mreže, i sl.,
- intenzitet i tok saobraćaja kroz mrežu varira na veoma različite i često nepredvidive načine,
- zakoni ponašanja aplikativnih programa u distribuiranoj računarskoj sredini još uvijek nijesu dovoljno poznati.

Za modeliranje računarskih sistema tradicionalno su se koristile analitičke matematičke metode, kao što su: teorija masovnog opsluživanja, teorija grafova, lanci Markova itd. Bez obzira na nisku efikasnost i nesigurnost analitičkih metoda, njihovo korišćenje je i pri izgradnji distribuiranih računarskih sistema nezamenljivo zbog praktičnosti i relativno niske cijene ovih metoda s jedne strane a visoke cijene i složenosti direktnog eksperimentisanja sa druge strane. Praktično se, međutim, pokazalo da je primjena ovih matematičkih sredstava povezana sa nizom teškoća i ima bitnih nedostataka, a osnovni je netačnost, i bitno uopštavanje opisa složenih pojava koje se dešavaju u računarskim sistemima.

Distribuirani računarski sistemi dosad su se istraživali uglavnom empirijski, putem proba i grešaka kroz izgradnju maketa i eksperimentalnih sistema. Izgradnja makete i eksperimentisanje nad realnim distribuiranim sistemom je dug i skup proces. On uključuje u sebi kako izgradnju hardverske strukture sistema tako i veoma složenog i skupog softvera.

Naprijed rečeno objašnjava značaj problema razrade novih metoda i sredstava analize efikasnosti distribuiranih računarskih sistema na etapi njihovog projektovanja, koji ne bi zahtijevali izgradnju makete sistema i koji bi dozvolili procjenu efikasnosti posmatranog sistema u što kraćem roku, sa manjim gubicima i sa većom tačnošću nego danas. Osim toga, danas odsustvuju sredstva koja bi dozvolila da se na osnovu teksta aplikativnog programa predskažu i efekti njegovog djelovanja u distribuiranoj računarskoj sredini.

1.2 Cilj rada

Prevazilaženje nedostataka dosadašnjih metoda i izgradnja nove metode za određivanje performansi distribuiranog računarskog sistema i njegovog programskog okruženja, osnovni je cilj ovog rada. Značaj postavljenog cilja ogleda se u tome što njegovo ostvarenje obezbeđuje relativno lako i komforno projektovanje i optimalan izbor hardverske i softverske strukture distribuiranog računarskog sistema.

S obzirom na postavljeni cilj u radu se morao riješiti čitav niz zadataka među kojima su najvažniji:

- proučavanje dosadašnjih metoda analize performansi distribuiranog računarskog sistema radi prevazilaženja njihovih nedostataka,
- istraživanja uzajamnih veza strukture distribuiranog sistema, načina organizacije podataka u sistemu i distribuiranih baza podataka kao i izbora optimalnog načina distribucije programa i podataka,
- posmatranje ponašanja aplikativnih programa prije svega programa tipa klijent/server u distribuiranom okruženju i pokušaj da se na osnovu ovog ponašanja odrede performanse čitavog sistema,
- razrada nove metode procjene i analize performansi distribuiranih računarskih sistema na nivou projekta a bez potrebe izgradnje prototipa sistema koji se projektuje,
- postavljanje principa izgradnje kompletnog instrumentarija za istraživanje posmatrane klase distribuiranih računarskih sistema.

1.3 Doprinos rada

Rezultati opisani u ovom istraživačkom radu trebali bi da posluže kao polazna osnova za razvitak novog perspektivnog naučnog metoda u istraživanju osobina distribuiranih sistema. Ovdje predložena metoda analize performansi distribuiranog računarskog sistema se razlikuje od dosadašnjih. Nju karakteriše:

- istraživanje dinamičkog ponašanja aplikativnih programa ili dijelova tih programa u distribuiranoj računarskoj sredini,
- izgradnja instrumentalnih sredstava za mjerenje i opis ponašanja ovih programa u različitim distribuiranim računarskim sredinama,
- pokušaj da se na osnovu ponašanja aplikativnih programa definiše jedinstvena tehnika mjerenja i analize distribuiranog računarskog sistema bez obzira na hardversku strukturu ili operativni sistem individualnih računara koji čine distribuirani sistem.

Osnovni napor u istraživanju urađen je na traženju parametara ponašanja aplikativnih programa, koji su nezavisni od fizičke odnosno sistemsko-softverske strukture sistema. i iskorišćenje tih svojstava za analizu sistemskih performansi distribuiranih sistema. Taj problem je, bez obzira na veliki praktični značaj, dosad istraživan nedovoljno.

1.4 Pregled rada

Uvodno poglavlje objašnjava aktualnost potrebe istraživanja distribuiranih računarskih sistema posmatrane klase. Opisani su ciljevi i doprinos rada sa stanovišta projektovanja ovakvih sistema, a takođe je dat i kratak pregled rada.

U skladu sa navedenim zahtjevima u *drugom* poglavlju dati su osnovni pojmovi i terminologija, cilj djelovanja i osnovne karakteristike distribuiranih računarskih sistema. Opisani su zadaci i funkcije, područje primjene, kao i potrebna programska podrška posmatrane klase ovih sistema. Navedene su bitne prednosti ovih sistema u odnosu na centralizovane, kao i razlike između njih i multiprocesorskih sistema sa djeljivom memorijom. Opisani su razlozi uvođenja i eksploatacije distribuiranih sistema. kao i arhitekture ovih sistema sa svim svojim osobenostima.

U okviru ovog poglavlja dat je kratak pregled lokalnih računarskih mreža, kao osnove za izgradnju distribuiranih računarskih sistema posmatrane klase. Nabrojane su topologije i metodi pristupa, standardi i protokoli, kao i najvažnije mrežne arhitekture lokalnih mreža.

U nastavku je opisan sistemski softver i najvažniji operativni sistemi personalnih računara. Objašnjeno je zašto su operativni sistemi osnova cjelokupne aktivnosti hardvera i softvera računarskog sistema. Definisan je pojam mrežnog operativnog sistema i njegova uloga u radu distribuiranih sistema posmatrane klase.

U drugom poglavlju opisane su takođe osnovne karakteristike distribuiranih baza podataka, kao jedne od tipičnih primjena distribuiranih sistema. Osnovni problem distribuiranih baza podataka je način distribucije podataka, odnosno dijelova baze, između računara u mreži. U radu su objašnjena sljedeća dva rješenja ovog problema:

- *duplirana* distribucija, kada se neki podaci češće korišteni na udaljenim računarima dupliraju,
- *razdijeljena* distribucija, kada se na svakom računaru čuva i koristi sopstveni skup podataka, dostupan za udaljene upite.

Kod razdijeljene distribucije, neke datoteke ili relacije mogu biti rastavljene horizontalno, kada se dijeli skup slogova ili vertikalno kada se dijeli skup polja ili atributa. Pojedini podskupovi, koji se čuvaju na različitim računarima, mogu biti duplirani djelimično ili potpuno. Razvoj metodologija distribucije i razmještaja podataka predstavlja jedan od osnovnih zadataka pri projektovanju distribuiranih sistema. Ovaj zadatak se dobrim dijelom može riješiti metodom imitacionog modeliranja.

Na kraju ovog poglavlja date su osnovne karakteristike distribuiranih aplikativnih programa kao i tehnologije klijent/server.

Treće poglavlje formuliše zadatak rada u opštem obliku. Izvršena je neformalna analiza postavljenog zadatka i njegovo pojašnjenje. U ovom poglavlju opisan je način dekompozicije distribuiranog računarskog sistema na njegove osnovne komponente: fizičku sredinu, izvršnu sredinu i radno opterećenje. Dat je pregled glavnih logičkih komponenti distribuiranog sistema i koncepcijski pogled na sistem. Pravilna dekompozicija posmatranog distribuiranog sistema i dobro poznavanje osobenosti njegove hardverske i softverske strukture su od posebnog značaja i neophodni su za izgradnju relevantnog matematičkog i imitacionog modela.

U daljem tekstu ovog poglavlja, navedeni su problemi izgradnje opisa ponašanja programa u distribuiranoj sredini. Opisani su primarni problemi analize i matematički postupci njihovog rješavanja. Na kraju su, na osnovu analize postavljenog zadatka, formulisani načini, postupci i perspektive njegovog rješavanja.

Četvrto poglavlje se odnosi na pregled dosada korišćenih metoda i sredstava procjene performansi individualnih računara tj. računarskih sistema sa tradicionalnom arhitekturom. U okviru ovog poglavlja pokušava se odgovoriti na pitanja: koji se dodatni problemi javljaju kod procjene performansi distribuiranih sistema, a koji nijesu bili svojstveni tradicionalnim računarskim sistemima i da li se za analizu i procjenu distribuiranih sistema može primijeniti već postojeća metodologija i ustaljeni postupci i sredstva.

Ovdje se posebna pažnja obraća modeliranju radnog opterećenja. Načinima izgradnje i primjene modela radnog opterećenja kao i uzrocima njihove sistemske zavisnosti. Definišu se metode izgradnje matematičkog modela i metode imitacionog modeliranja korišćene u ovom radu. Time se gradi koncepcija koja dozvoljava efikasno upoređenje različitih prilaza ka izgradnji sistema procjene performansi distribuiranog računarskog sistema.

U *petom* poglavlju opisane su specifičnosti distribuiranih aplikativnih programa, načini njihovog opisa i predstavljanja itd. Podrobno se analizira ponašanje distribuiranih aplikativnih programa, kao i pojave koje se s tim u vezi dešavaju u distribuiranim sistemima. Data je klasifikacija ponašanja distribuiranih aplikativnih programa, kao i razni načini prikaza njihovog ponašanja.

Osnovni cilj ove glave je opis pojma logičke ili tz. funkcionalne karakteristike distribuiranog aplikativnog programa, koja karakteriše njegovo dinamičko ponašanje. Za njeno razumijevanje potrebo je proanalizirati uzajamno dejstvo aplikativnog programa i hardverske odnosno sistemsko-softverske strukture sistema. Ovdje se uvodi pojam funkcionalnosti aplikativnog programa kao osnove uzajamnog dejstva aplikativnog programa i distribuiranog sistema. Ova funkcionalnost treba i mora ostati nepromenjena bez obzira na promjene hardverske i sistemsko-softverske strukture sistema. Čak se može govoriti i o uvođenju pojma svojevrsnog logičkog resursa, kome se aplikativni programa obraća za zadovoljenje svojih resursa. Definiše se potreba za nepromenljivošću njegove funkcionalnosti. Određeni su osnovni oblici jednog ovakvog logičkog resursa i njihove karakteristike.

Posebna pažnja u ovoj glavi posvjećena je uzajamnoj vezi programa i vremena. Odnosno ponašanju programa u vremenu. Pokazana je teza o uslovnoj nezavisnosti ponašanja programa od vremena. Sa ciljem povezivanja dejstva programa i vremena uvedi je pojam svojevrsne mjere, koja je označena kao: mjera težine ostvarivanja neke funkcije aplikativnog programa. Ta funkcija može biti izračunavanje aritmetičkih operacija nad nekim skupom operanada, pretraživanje baze ili tome slično, adekvatno kontekstu razmatranja. Dati su praktični primjeri primjene te mjere i tačnost rezultata prognoziranja pomoći nje.

U zaključku ove glave formulisani su problemi, nužni za formalizaciju, osnovnog zadatka ovog rada i njegovog strogo matematičkog rješenja:

- uzajamno dejstvo različitih oblika paralelizma i njihov uticaj na ponašanje aplikativnih programa u distribuiranoj sredini,

- načini opisa funkcionalnosti aplikativnih programa i ponašanja programskih elemenata u distribuiranoj računarskoj sredini.

U okviru ovog poglavlja posebna pažnja je posvjećena definisanju pojma sistemski nezavisne mjere ili invarijante programa, koji daje osnovu za dokazivanje ispravnosti postupka modeliranja i analize distribuiranih računarskih sistema, uz korišćenje predložene metodologije.

Šesto poglavlje, definiše osnovne pretpostavke za izgradnju matematičkog modela distribuiranog sistema, razmatra načine predstavljanja paralelnih i distribuiranih programa, daje prednosti izabrane metode modeliranja i na kraju, algoritam procesa formiranja matematičkog modela. osnovne komponente ovog matematičkog modela su:

- aplikativni procesi,
- logički resursi,
- izvršioci i
- posmatrač.

Za istraživanje svojstava hardverske strukture sistema primijenjen je algebarski prilaz i aparat iz teorije grafova. Pokazano je da struktura ma kog sistema iz razmatrane klase može biti izražena u terminima predložene algebre. Dobijeni rezultati kasnije se koriste za izgradnju postupaka i sredstava imitacionog modela opisanog distribuiranog sistema kao i za dokazivanje adekvatnosti istih.

U *sedmom* poglavlju izložen je postupak *imitacionog* modeliranja distribuiranog računarskog sistema. Date su osnovne pretpostavke za izgradnju koncepcijskog, odnosno idejnog, modela sistema i definisani osnovni objekti imitacionog modela sa porukama, pomoću kojeg se simulira dinamičko ponašanje aplikativnih programa u posmatranom distribuiranom sistemu. Opisani model:

- odražava kako algoritamske tako i količinske karakteristike ponašanja programskih elemenata sistema,
- uključuje vrijeme kao količinsku karakteristiku,
- obuhvata strukturu i kapacitet hardverskih elemenata sistema,
- odražava mogući hijerarhijski karakter strukture distribuiranog računarskog sistema.

Osnovu, ovdje predloženog, modela sa porukama čine tri vrste objekata:

1. *Dinamički* objekti sistema, kojima je predstavljena funkcionalnost aplikativnih programa. Oni odražavaju dinamičko ponašanje aplikativnog programa ili pojedinih programskih elemenata i logički opisuju aplikativne programe tokom njihovog izvršavanja.
2. *Pasivni* objekti sistema predstavljaju hardverske i sistemsko-softverske elemente ili resurse sistema tj. izvršioce na kojima se izvršava dati program. Oni obezbjeđuju izvršavanje funkcija koje su definisane datim aplikativnim programima.
3. *Posmatrački* (monitorski, instrumentalni) objekti.

Dinamički objekti su matematički objekti, koji sadrži podatke o funkcionalnosti programskih elemenata sistema. Takvi objekti se sreću u literaturnim izvorima pod različitim nazivima: istorija programa, operaciono-logička istorija, termalna istorija, termalno-logička istorija programa i slično. Osnovna razlika predložene matematičke apstrakcije aplikativnog procesa od onih koji su već prisutni u literaturi sastoji se u sljedećem:

- ✓ razvoj procesa se opisuje u vidu djelimično-uređenog niza koraka, a ne prosto događaja;
- ✓ na svakom koraku razlikuju se događaji koji se sastoje u prenosu dejstva na odgovarajući proces (npr. prenos podataka tom procesu), od događaja, kada sam proces pokušava djelovati na nekog;
- ✓ svaki korak ima atribut, označen kao složenost, koji dozvoljava razmatranje i razvoj procesa u vremenu;
- ✓ opis uzima u obzir kako unutrašnji tako i spoljašnji nedeterminizam u ponašanju programa;

Izuzetno važna prednost ovdje predloženog načina opisa ponašanja programskih elemenata, sastoji se u tome, što on dozvoljava da se lociraju tačke nedeterminizma u ponašanju programa.

Pomoću pasivnih objekata sistema izgrađuje se model hardverske i sistemsko-softverske strukture. Ove objekte karakterišu sljedeći parametri:

- ✓ brzinske karakteristike koje određuje vremenski interval koji je neophodan hardversko-softverskom uređaju za izvršavanje određene aktivnosti,
- ✓ struktura veza između autonomnih računara,
- ✓ tehničke performanse autonomnih računara,
- ✓ sistem upravljanja distribuiranog sistema,
- ✓ način komunikacije (sinhroni ili asinhroni) među računarima i sl.

Posmatrač dijagnosticira ponašanja aplikativnog programa pri ograničenjima, određenim izvršiocem tj. hardverskim i sistemsko-softverskim elementima sistema.

Imitacioni model sa porukama, koji je opisan u ovom poglavlju, je objektno-orjentisan opis sistema i procesa. To je bitno iz razloga intenzivnog razvitka objektno-orjentisanog metoda programiranja. Ta činjenica je pokazala značajan uticaj na arhitekturu kontrolnog distribuiranog sistema razrađenog u okviru datog istraživanja. Može se lako pokazati da ovaj model, po klasifikaciji, zauzima srednji položaj između aparata Petrijevih mreža i P/V sistema (sistema sa semaforima). U okviru predloženog imitacionog modela ostvaruje se formalna postavka zadatka analize performansi.

Na kraju ovog poglavlja opisan je način dokazivanja adekvatnosti, pravilnosti i korektnost imitacionog modela kao i kalibracija istog.

Sljedeće, *osmo* poglavlje opisuje metode i sredstva mjerenja performansi distribuiranog računarskog sistema. Opisana je koncepcija i tipovi posmatrača kao instrumenata mjerenja. Date su glavne karakteristike programskih, mikroprogramskih i hardverskih posmatrača, kao i njihova uporedna analiza.

Glavne prednosti mjerenja performansi na stvarnom sistemu su eliminisanje grešaka prouzrokovanih zanemarivanjem nekih relevantnih sistemskih karakteristika koje imaju efekat na performanse sistema. Kombinovanjem mjerenja na realnom distribuiranom sistemu ili nekom kontrolnom sistemu sa imitacionim modeliranjem dobijaju najbolji rezultati analize posmatranog sistema.

U *devetom* poglavlje dat je primjer primjene imitacionog modeliranja za analizu jednog distribuiranog sistema posmatrane klase, kao i neki eksperimentalni rezultati mjerenja performansi na realnom sistema. Opisan je način formiranja radnog opterećenja kako kontrolnog sistema tako i samog imitacionog modela. Kod determinističkog načina formiranja radnog opterećenja, postupak se sastoji u tome da se namjenski izgrađen

distribuirani aplikativni program izvršava u specijalizovanoj distribuiranoj sredini tz. kontrolnom distribuiranom sistemu. U toku izvršavanja mjere se i snimaju parametri koji karakterišu ponašanje programa. Rezultati mjerenja skupljaju se u formi trase i služe kao ulazni podaci za imitacioni model. Kod stohastičkog prilaza radno opterećenje ili parametri distribuiranog programa se generišu uz pomoć raspodjele slučajnih brojeva.

U toku postupka analize upoređuju se vrijednosti pokazatelja performansi dobijeni mjerenjem i simulacijom na modelu. Slaganje eksperimentalnih rezultata sa onim dobijenim uz pomoć imitacionog modela potvrđuje da je model pogodan za analizu ponašanja aplikativnih programa u distribuiranoj sredini.

Primjenom ove metode analize distribuiranih sistema dobijene su nove informacije o uticaju hardverske strukture i ponašanja programa na performanse distribuiranog sistema. Zbog toga bi izgrađeni model i rezultati, dobijeni njegovom primjenom, mogli da budu osnova za optimizaciju i projektovanje distribuiranog računarskog sistema, kao i imitacioni eksperiment u teorijskom istraživanju u ovoj oblasti.

Kao što se vidi osim čisto naučno-istraživačkih ciljeva ovaj sistem modeliranja se može koristiti i za rješenje važnih praktičnih zadataka. Tako npr.: treba izvršiti dekompoziciju zadatog programskog sistema tako, da ona funkcioniše efektivno sa tačke gledišta neke ciljne funkcije, ili neka je data klasa distribuiranih aplikativnih programa, a treba odrediti takvu hardversku strukturu distribuiranog računarskog sistema, da bi softver tj. distribuirani aplikativni program zadate klase funkcionisao efikasno sa tačke gledišta neke tražene funkcije. Jasno, da su ta dva zadatka samo krajnji zadaci jednog neprekidnog niza zadataka.

Rezultati ovog istraživačkog rada dosada su objavljeni i izlagani na:

1. IX Kongresu matematičara i fizičara Jugoslavije, Petrovac 1995.
2. Simpozijumu o računarskim naukama i informatici, Brezovica, 1995.
3. XL Konferenciji ETRAN, Budva, 1996.
4. Katedri ASVK fakulteta VMiK MGU, Moskva 2000.
5. U zborniku: "Programski sistemi i instrumenti" u izdanju fakulteta VMiK MGU, Moskva 2000.

⁂

Eksperimentalni dio ovog rada urađen je u laboratoriji ASVK na fakultetu VMiK Moskovskog Državnog Univerziteta pod rukovodstvom ***akademika Prof. dr Ruslana Leonidoviča Smeljanskog.*** *Ovom prilikom bih želeo da se zahvalim saradnicima i šefu laboratorija Profesoru R.L.Smeljanskom, na svestranoj pomoći i podršci.*

GLAVA

2. KARAKTERISTIKE POSMATRANE KLASE DISTRIBUIRANIH RAČUNARSKIH SISTEMA

2.1 Uvod

Uopšteno govoreći distribuirani računarski sistem je vrlo širok pojam i nema jedinstveno prihvaćenu interpretaciju. U literaturi se može sresti više definicija ovog pojma, ali se obično pod distribuiranim računarskim sistemom podrazumijeva skup nezavisnih računara koje korisnik vidi kao jedinstven računar. Njegovo korišćenje, po pravilu, nije mnogo složenije od korišćenja običnog personalnog računara. Oni obuhvataju široku lepezu računarskih sistema od multiprocesora sa distribuiranom memorijom pa sve do nekih tipova LAN i WAN računarskih mreža, odnosno i računarske mreže se mogu, pod izvesnim uslovima posmatrati kao specijalan slučaj distribuiranih sistema.

U okviru distribuiranog računarskog sistema mogu biti uključeni personalni računari, radne stanice, miniračunari i veliki tz. *host* računarski sistemi opšte namjene. Računari odnosno procesori, u okviru distribuiranog sistema, označavaju se različitim skupovima imena kao što su ***lokacije*** (engl. *site*), ***računari*** (engl. *computer*), ***čvorovi*** (engl. *node*) i slično zavisno od sadržaja koji se nagovještava u istraživanju. Oni su povezani pomoću računarske mreže i imaju mogućnost međusobne komunikacije i razmjene podataka.

Klasa distribuiranih računarskih sistema, posmatrana u ovom radu, sastavljena je od više autonomnih personalnih računara. Karakteristike savremenih personalnih računara, takve kao što su: relativno niska cijena, visoke performanse, sigurnost u radu, elastičnost pri proširenju sistema itd., čine ih veoma pogodnim za izgradnju distribuiranih sistema, te su ovakvi sistemi danas veoma rasprostranjeni. Autonomni personalni računari rade samostalno, tj. nezavisno jedan od drugog, pod individualnim operativnim sistemom, pod kojim se mogu restartovati proizvoljan broj puta. Međusobno su povezani pomoću lokalne računarske mreže, mogu imati različite performanse i funkcije u sistemu, ali im je osnovno to da međusobno ne dijele operativnu memoriju niti generator takta (engl. *clock*).

2.2 Razlozi izgradnje distribuiranih sistema

Distribuirani, kao uopšte i svi ostali paralelni računarski sistemi, prvenstveno su nastali u težnji da se stvori računarski sistem što boljih performansi. Distribucijom programa i podataka na više računara postiže se bolje iskorišćenje računarskih resursa, veća fleksibilnost u radu sistema a dobijaju se i nove funkcionalne mogućnosti. Pored ovih, sljedeći razlozi su bitni za nastajanje, izgradnju i instalaciju distribuiranih sistema.

1. *Skraćivanje vremena izračunanja* glomaznih matematičkih izraza. Ono nastaje ako je komunikaciono kašnjenje, uneto u aplikaciju, manje od vremena izvršavanja i ako postoji stvarni paralelizam u aplikaciji. Primjenjuje se u slučajevima kada se manja količina podataka prenosi između procesora, odnosno krajnjih čvorova u mreži, a prisutna su glomazna izračunavanja. To je tz. veliki ili prosti (engl. *coarse grain*

level) nivo paralelizma. U ovom slučaju distribucijom dijelova programa, odnosno izgradnjom distribuiranih aplikacija, mogu se ubrzati izračunavanja. Da bi se sličan efekat postigao i za srednji (engl. *medium grain level*) nivo paralelizma, koji je tipičan kod petlji, neophodno je da paralelni računarski sistem bude čvrsto spregnut ili da brzina prenosa mrežom bude reda 100 Mbps ili viša, što je i ostvarivo kod klase distribuiranih sistema koja se ovdje razmatra.

2. *Smanjenje osjetljivosti na otkaze*, a samim tim i bolja raspoloživost (engl. *Availability*) i pouzdanost (engl. *reliability*) sistema. Distribucijom aplikacije i resursa postiže se, da se u slučaju otkaza jednog dijela sistema, koriste alternativna rješenja. Prolazni čvor se može zamijeniti rezervnim tako da u trenutku otkaza ovaj preuzima njegovu ulogu. Da bi ovo bilo ostvarljivo teži se da sistem nema kritičnih resursa i da je u što većoj mjeri simetričan.
3. *Specifična funkcionalnost djelova distribuiranog sistema*. Pojedini čvorovi, odnosno računari ili samo pojedini procesori u sistemu mogu imati specifičnu ulogu. Tako se mogu izdvojiti npr.: fajl serveri, komunikacioni serveri, numerički serveri, print serveri, serveri baza podataka i slično.
4. *Prirodna (inherentna) distribuiranost aplikacije*. Kod nekih firmi poželjno je izvršiti prostornu dislokaciju radnih mjesta, pa samim tim i aplikativna rješenja zahtijevaju dislokaciju podataka, programa i/ili hardverskih resursa.
5. *Povećanje efikasnosti* distribuiranog sistema, u odnosu na nedistribuirani, se ogleda u smanjenju vremena odgovora sistema (engl. *response time*) i ukupne cijene potrebne za dobijanje tražene informacije.
6. *Mogućnost lokalne autonomije.* Distribucijom sistema omogućeno je individualnim korisnicima da sami kontrolišu svoje lokalne podataka i da budu manje zavisni od udaljenog centra za procesiranje. U isto vrijeme ovi korisnici mogu pristupati podacima na drugim lokacijama kada je to neophodno. Ovo se postiže zahvaljujući većem procentu pristupa lokalnog karaktera uz istovremeno obezbjeđenje da dislokacija programa i podataka, kao i njihove eventualne kopije, bude neprimjetna (engl. *transparent*) za korisnika kako bi isti imao utisak korišćenja jedinstvenog sistema.
7. *Lako proširenje sistema.* Jedanput instaliran distribuirani sistem može lako biti proširen dodavanjem novih čvorova ako prostor za smještaj ili kapacitet procesiranja podataka postane nedovoljan.
8. *Ekonomski razlozi* ogledaju se u boljem iskorišćenju resursa, manjoj početnoj investiciji izgradnje sistema, manjoj cijeni proširenja sistema i sl.

Međutim treba naglasiti da se, prilikom izgradnje distribuiranih sistema javljaju dodatni problemi, koji nijesu svojstveni klasičnim računarskim sistemima sa jednim procesorom. Među problemima sa kojima se susreće projektant distribuiranog sistema mogu se izdvojiti sljedeći:

1. *Problemi softvera.* U koje spadaju: izgradnja kompleksnih operativnih sistema i specijalizovanih jezika programiranja, kao i dodatni programerski rad potreban za izradu, koordinacija i izvršavanje distribuiranih aplikacija.
2. *Problemi distribucije.* Što podrazumijeva određivanje lokacije programa i podataka distribuiranih kroz mrežu i distribuciju aktivnosti među računarima distribuiranog

sistema, kao i sinhronizaciju kopija programa i podataka i njihovo održavanje u konzistentnom stanju.

3. *Problemi komunikacije.* Povezivanje i rukovanje komunikacijom između programa u mreži, mogućnost gubitka informacija pri prenosu podataka, zasićenost prenosnih puteva, problemi razvoja komunikacione opreme i njeno servisiranje i sl.
4. *Problemi zaštite* podrazumijevaju: Detekciju grešaka i restauraciju sistema na nehaotičan i siguran način, sigurnost i zaštitu resursa kod ograničenog udaljenog pristupa od strane neautorizovanih korisnika, obezbjeđenje zaštite od uništenja ili gubitka podataka i sl.

2.3 Oblasti primjene distribuiranih računarskih sistema

Posmatrana klasa distribuiranih računarskih sistema, koja je zasnovana na personalnim računarima ima široku i raznovrsnu primjenu. Pomoću ovakvih sistema realizuju se najraznovrsnije, kako autonomne, tako i integrisane procedure obrade podataka. Koriste se u javnim službama, bankama, poštama, preduzećima, fabrikama, trgovinama, kao kontrolni mjerni sistemi, sistemi upravljanja eksperimentima, tehnološki kompleksi, kao instrumentalna podloga distribuiranih baza podataka itd.

Distribuirani sistemi posmatrane klase se *primjenjuju u nauci i tehnici* za automatizaciju i ubrzavanje glomaznih izračunavanja. Koriste se za izradu složenih projekata u građevinarstvu i arhitekturi, mašinstvu, elektrotehnici i sl. na čijoj je izradi potrebno angažovanje većeg broja ljudi. Ovi sistemi se često mogu sresti u mašinskoj industriji za upravljanje mašinama alatljikama, za komandovanje robotima pri sastavljanju automobila i aviona i sl.

Poslovna primjena distribuiranih računarskih sistema ogleda se u čuvanju, obradi i izdavanju informacija za potrebe poslovnih sistema takvih kao što su: banke, privredne organizacije, pošte i javne ustanove. Ovdje se distribuirani sistemi koriste za razne evidencije, finansijsko poslovanje, obračun plata, za dobijanje izvještaja, statističkih i drugih pokazatelja poslovanja i sl. koji se teško mogu obaviti bez distribuiranog sistema odnosno distribuiranih baza podataka.

U proizvodnji, saobraćaju, dispečerskim centrima na željeznici i u elektroprivredi itd. javljaju se složeni problemi *upravljanja proizvodnim sistemima*. Ovdje se računari uz pomoć analogno-digitalnih pretvarača povezuju na davače koji daju informacije o stanju objekata na terenu i izdaju komande za izmijenu stanja tih objekata ili upravljanje. Time se izgrađuje distribuirani računarski sistem, koji može uspješno rješavati pomenute probleme.

Distribuirani računarski sistemi, zasnovani na lokalnim mrežama sa personalnim računarima, našli su svoju široku i raznovrsnu primjenu i u *komunikacijama*. I to kako u klasičnim telekomunikacionim objektima za prenos govornih informacija tako i za prenos svih ostalih multimedijalnih informacija. Naravno, povezivanjem na svijetsku računarsku mrežu Internet, korisnik personalnog računara može ostvariti komunikaciju na globalnom nivou.

2.4 Paralelni i distribuirani računarski sistemi

Distribuirani sistemi pripadaju široj klasi paralelnih računarskih sistema. Jedan od osnovnih ciljeva izgradnje paralelnih i distribuiranih sistema je dijeljenje procesorskog rada na više procesora, odnosno, ostvarenje većeg ili manjeg stepena *paralelizama* prilikom izvršavanja programa ili dijelova programa. Ovakvi sistemi označeni su kao multiprogramski ili multitasking sistemi. Paralelno, odnosno *konkurentno*, izvršavanje programa se može ostvariti i u klasičnim sistemima sa jednim procesorom, međutim ovakvi sistemi ne omogućavaju stvarni paralelizam, već se radi o pseudoparalelizmu. Kod pseudoparalelizma se koriste mehanizmi kojima se simulira komunikacija unutar jedinstvenog sistema, korišćenjem principa za djeljivu memoriju. U tom slučaju se koriste djeljive promjenljive za interakciju između lokalnih procesa uz jasnu specifikaciju interakcija sa nelokalnim resursima kroz komunikacioni interfejs. Pojedini programski jezici i savremeni operativni sistemi dozvoljavaju eksplicitno raspoređivanje *procesa*, tako da se više njih mogu konkurentno izvršavati na jednom procesoru. Pod procesom se podrazumijeva sekvencijalno izvršavanje dijela programa i on će u daljem tekstu biti detaljnije opisan.

Računarski sistemi sa više procesora, u literaturi često označeni kao paralelni računarski sistemi, mogu se podijeliti na *multiprocesorske* sisteme tj. sisteme sa djeljivom memorijom i *distribuirane* tj. multiračunarske sisteme izgrađene uz pomoć računarskih mreža, kod kojih svaki procesor ima svoju lokalnu memoriju. Može se čak reći da su multiračunarski sistemi, multiprocesorski sistemi sa distribuiranom memorijom. Strogo uzevši multiprocesorski sistemi sa djeljivom memorijom ne pripadaju klasi distribuiranih računarskih sistema. U svakom slučaju, između sistema sa djeljivom memorijom i distribuiranih računarskih sistema ne može se baš sa sigurnošću povući oštra granica. Javlja se potreba da se integrišu dobre strane multiprocesorskih sistema sa djeljivom memorijom i prednosti komunikacionog mehanizma multiračunara.

2.5 Arhitekture distribuiranih sistema

S obzirom na arhitekturu razlikuju se dvije vrste distribuiranih računarskih sistema i to: čvrsto (engl. *closely coupled*) i labavo spregnuti (engl. *loosely coupled*) sistemi.

Čvrsto spregnuti distribuirani sistemi imaju veliki broj čvorova koji se u principu prave kao identični čvorovi, formirajući relativno pravilan graf veza između njih. Kod čvrsto spregnutih distribuiranih računarskih sistema u upotrebi su komunikacione mreže tipa hiperkocke ili rešetke. Zahvaljujući topologiji pravilnog grafa koja se ogleda u malom dijametru (najduži minimalni put), malom stepenu čvora i malom rastojanju između susednih čvorova, prosljeđivanje poruka je vrlo brzo. Zbog malog rastojanja između procesora veze su kratke tj. lokalne, pouzdane i imaju malu osjetljivost na otkaze. Ukupno vrijeme prenosa poruke između dva procesora je ispod jedne milisekunde. Ovo vrijeme najviše zavisi od tz. "bitske" brzine prenosnog kanala. Ova brzina se mjeri brojem bita koje kanal može propustiti u jedinici vremena. Od bitne važnosti je i minimalno kašnjenje prilikom određivanja optimalnog puta za

prosljeđivanje poruka do odredišta. Ovo vrijeme "odlučivanja", odnosno, traženja najoptimalnijeg puta ne smije biti veće od jedan do dva bitska intervala. U cilju smanjenja vremena "odlučivanja" potrebno je obezbediti adaptivno prosljeđivanje poruka kroz mrežu tj. rutiranje sa korišćenjem tabela preopterećenih ili otkazanih puteva.

Čvrsto spregnuti distribuirani računarski sistemi koriste se za izgradnju sistema specijalne namjene. Najčešće tamo gdje su potrebna velika izračunanja za što kraće vrijeme ili čak u realnom vremenu takva su npr.: istraživanja u oblasti kvantne i statističke fizike, kosmička istraživanja, istraživanja u meteorologiji, biologiji, ekologiji, farmakologiji, balistici, u automobilskoj industriji, kod proizvodnje nafte i gasa, za projektovanje elektronskih sklopova, za kontrolu i upravljanje železničkim saobraćajem i sl.

Labavo spregnuti distribuirani sistemi se izgrađuju uz pomoć klasičnih računarskih mreža. Čvorovi, odnosno računari, mogu biti povezani fizički na različite načine. Kod ovih distribuiranih sistema čvorovi su fizički locirani na malom ili velikom geografskom prostoru. S obzirom na prostor lociranja razlikujemo LAN (engl. *Local Area Network*), MAN (engl. *Metropolitan Area Network*) i WAN (engl. *Wide Area Network*) mreže.

Lokalne računarske mreže - LAN organizovane su na manjem prostoru, podaci se prenose velikim brzinama sa malom vjerovatnoćom grešaka. Globalne računarske mreže, označene kao WAN, su vezane na svetsku mrežu ili su to privatne mreže rasprostranjene na nekom većem geografskom prostoru. Mogu obuhvatati više država ili kontinenata. Čvorovi ili računari su fizički međusobno veoma udaljeni a komunikacioni kanali, između njih, relativno spori i manje pouzdani u poređenju sa lokalnim mrežama. Tipično se za izgradnju ovih mreža koriste telefonske linije, mikrotalasni linkovi ili satelitske veze. Prenos podataka se ostvaruje najčešće uz pomoć modema. Gradske mreže ili MAN obuhvataju gradsko mrežno područje i predstavljaju srednje rješenje između lokalnih i globalnih mreža uz težnju da se iskoriste dobre strane i jednih i drugih..

Zahvaljujući napretku tehnologije, brzine prenosa podataka u lokalnim računarskim mrežama stalno rastu i već je u upotrebi lokalna računarska mreža koja omogućava i brzine prenosa od jednog gigabita u sekundi. Time se vrijeme prenosa poruke između procesora kod labavo spregnutih sistema približava ili je čak i manje od vremena prenosa poruke kod nekih čvrsto spregnutih distribuiranih sistema. S druge strane, za podršku čvrsto spregnutim sistemima potreban je specijalizovani softver, koji znatno utiče na povećanje njihove cijene. Ovo dovodi do toga da je broj instalacija čvrsto spregnutih distribuiranih sistema relativno mali u poređenju sa labavo spregnutim distribuiranim sistema.

Neka aplikativna rješenja zahtijevaju povezivanje LAN i WAN mreža, odnosno obuhvataju više mreža tipa LAN-WAN-LAN. Tako formirana računarska mreža na vrlo velikom prostoru, takoreći na prostoru čitave zemljine kugle, predstavlja globalnu svetsku računarsku mrežu poznatu pod imenom *Internet*. Izgradnja globalne računarske mreže (engl. *internetworking*) obuhvata kako probleme fizičkog povezivanja sličnih ili različitih mreža, uključujući i WAN mreže, tako i način komuniciranje između sličnih aplikacija i njima podržanih protokola. Ovakvim povezivanjem javnih računarskih

mreža omogućeno je efikasno dijeljenje resursa kroz mrežu i ostvarivanje funkcija potrebnih krajnjim korisnicima mreže. Zahvaljujući velikim brzinama prenosa podataka, kod savremenih računarskih mreža, kao poseban oblik labavo spregnutih distribuiranih računarskih sistema može se posmatrati i globalna svjetska računarska mreža *Internet*. Povezivanje računara na Internet, odnosno izgradnja jednog *globalnog distribuiranog sistema*, ima niz pogodnosti, među kojima spadaju: efikasnije komuniciranje među udaljenim korisnicima, blagovremeno dobijanje podataka i informacija, transfer informacija, koji se obezbeđuje uslugama Interneta, među kojima su tipične elektronska pošta i prenos datoteka, saradnja na obradi podataka, između udaljenih računara.

U ovom radu, razmatrani su distribuirani sistemi izgrađeni uz pomoć brzih lokalnih računarskih mreža, što ne znači da se dobijeni rezultati pod izvesnim uslovima ne mogu primijeniti i na druge pomenute klase distribuiranih sistema.

2.6 Lokalne računarske mreže - LAN

Povezivanje personalnih računara, odnosno njihovo *umrežavanje* u lokalne mreže i izgradnja distribuiranog sistema, ima niz prednosti u odnosu na tradicionalni "udaljenom pristup" host računaru uz korišćenje jedinstvene mrežne arhitekture. Umrežavanje računara u lokalne mreže omogućuje:

- dijeljenje resursa, odnosno pristup dodatnim perifernim uređajima kao što su datoteke, štampači, skeneri, grafički uređaji itd,
- standardizaciju aplikacija,
- upotrebu interaktivnih aplikacija, pri čemu podaci mogu biti dislocirani,
- izgradnju klijent/server sistema, kod kojih aplikativni programi "vide" mrežu kao jedan programski interfejs koji može biti korišćen za dijeljenje funkcija i procedura kroz mrežu,
- smanjenje troškova zahvaljujući diobi programa, podataka i perifernih uređaja.

Danas, uglavnom, postoje dva principa izgradnje lokalnih mreža na bazi personalnih računara: mreže računara istog prioriteta (engl. *peer-to-peer networks*) i serverske mreže (engl. *server based networks*). Ove mreže se izgrađuju na relativno malom, ograničenom području, po pravilu povezuju računare i periferne uređaje smještene u okviru jedne zgrade ili grupe zgrada. Za prenos podataka koristi se jedinstven komunikacioni kanal, koji režimom multipleksiranja, opslužuje sve računare u mreži. Jedinstveni kanal se realizuje uz pomoć medijuma sa višestrukim pristupom, visoke propusne moći od 0,5 Mbps do 1 Gbps. S obzirom na način pristupa prenosnom medijumu i topologiju, ima više tipova lokalnih mreža.

Lokalne mreže predstavljaju osnovu za izgradnju distribuiranih računarskih sistema na bazi personalnih računara. Istraživanje ove klase distribuiranih sistema nije moguće bez poznavanja pravila i zakonitosti koja važe kod lokalnih računarskih mreža. Ovdje će one biti detaljnije razmotrene.

2.6.1 Topologije i metode pristupa

Pod pojmom tipologije podrazumijeva se fizički raspored računara, kablova i drugih komponenti mreže. Tri su osnovne *fizičke topologije* lokalnih mreža: magistralna, u obliku zvijezde i u obliku prstena. Pored ove tri osnovne primjenjuju se i kombinacije ovih topologija, npr.: objedinjenje više magistralnih ili prstenastih podmreža, kombinacija zvijezde i magistrale ili kombinacija zvijezde i prstena.

Magistrala (engl. *bus*) je najjednostavniji i najčešći način povezivanja računara u mrežu ili umrežavanja. Njena konfiguracija je linearna sa računarima povezanim samo jednim kablom koji se zove stablo, kičma ili segment (engl. *trunk*, *backbone* ili *segment*). On sve umrežene računare povezuje pravolinijski. U topologiji zvijezde, računari su povezani segmentima kablova sa centralnim čvorom koji se označava kao hab (engl. *hub*) ili svič (engl. *switch*), dok se kod topologije prstena računari kružno povezuju kablom.

Koja će topologija biti primijenjena, pri projektovanju i izgradnji lokalne mreže zavisi od niza faktora među kojima su najvažniji:

- *cijena instalacije* ili cijena fizičkih veza između čvorova u sistemu,
- *komunikaciona cijena*, odnosno vrijeme i novac koji su potrebni za slanje poruke između čvorova u mreži,
- *stabilnost performansi* u odnosu na promjene opterećenja mreže,
- *komforan način upravljanja* i nadgledanja mreže,
- *pouzdanost*, odnosno učestanost kvar ova,
- *raspoloživost*, tj. stepen do koga se podacima može pristupati uprkos ispada nekog od segmenata između čvorova ili samih čvorova,
- *proširenje*, koje se ogleda u dodavanju novih računara odnosno čvorova.

Ovi faktori igraju važnu ulogu pri izboru kablovske strukture i ostalih mrežnih komponenti lokalnih mreža. Njihova optimizacija i usklađivanje moguće je uz pomoć metoda imitacionog modeliranja.

Računari povezani u lokalnu mrežu rukuju porukama na više načina, pa se pored fizičke, može govoriti i o njihovoj *logičkoj topologiji*. S obzirom na mehanizme za rukovanja porukama i metode pristupa prenosnom medijumu razlikuju se više vrsta logičke tipologije lokalnih mreža.

Jedna vrsta mreža povezuju računare u logičke krugove. Poruke se šalju od računara do računara sekvencijalno u obliku jednog logičkog prstena. Kod druge vrste mreža računar šalje poruku svim ostalim računarima istovremeno u obliku emisije. To je difuzni sistem emitovanja. Postoji i deterministički pristup sa prioritetom zahtjeva, kod koga računari pristupaju prenosnom medijumu pod centralističkom kontrolom haba ili sviča. Takva mreža ima oblik logičke zvijezde. Pri izradi modela distribuiranog računarskog sistema mora se posebno voditi računa da model odražava stvarnu logičku topologiju projektovanog sistema.

Metode pristupa medijumu su skupovi pravila koji definišu načine predaje i prijema podataka na ili sa prenosnog medijuma. Metodi pristupa sprečavaju da više računara pristupa medijumu u isto vrijeme. Obezbeđujući da samo jedan računar u jednom trenutku predaje podatke prenosnom medijumu, metodi pristupa održavaju red u slanju i primanju podataka preko mreže. Osnovni metodi pristupa su:

- višestruki pristup medijumu sa praćenjem njegovog zauzeća;
  - sa otkrivanjem kolizije, (engl. *Carrier-Sense Multiple Access with Collision Detection*, CSMA/CD);
  - sa izbjegavanjem kolizije, (engl. *Carrier-Sense Multiple Access with Collision Avoidance*, CSMA/CA);
- metod predavanja tokena koji slanje podataka omogućava samo jednom računaru koji ga posjeduje;
- deterministički pristup medijumu sa prioritetom zahtjeva, (engl. *Demand Priority Access Medium*, DPAM).

Kod višestrukog pristupa sa nadgledanjem nosioca i otkrivanjem kolizije (CSMA/CD) svaki računar iz mreže, uključujući klijente i servere, provjerava saobraćaj u kablu. Ako je kabl slobodan, računar simultano šalje podatke u mrežu, dok svi ostali računari osluškuju. Ukoliko su podaci prisutni u kablu, ni jedan računar ne može da emituje podatke.

Kada dva računara istovremeno počnu da šalju podatke, dolazi do kolizije. Tada oba računara privremeno zaustavljaju prenos, ponavljajući ga poslije nekog slučajnog vremenskog intervala, kada je mogućnost da ponovo dođe do kolizije vrlo mala. Sa povećanjem mrežnog saobraćaja mogućnost za koliziju i potreba za izbjegavanjem iste se povećava, što usporava mrežu.

Za razliku od CSMA/CD kod višestrukog pristupa sa nadgledanjem nosioca i izbjegavanjem kolizije svaki računar signalizira svoju namjeru da će da prenosi podatke, čime se može izbjeći kolizija. Ova metoda, međutim, nema mnogo pristalica.

Kod metoda predavanja *token-a*, posebna vrsta paketa nazvana token kruži od računara do računara. Računari tako obrazuju logički prsten. Kada neki računar iz prstena želi da pošalje podatke kroz mrežu, on mora da čeka da dobije slobodan token. Tek tada može da prenosi podatke. Dok jedan računar koristi token, drugi računari ne mogu da šalju podatke. Kod ovog metoda nema kolizije, niti gubljenja vremena radi oporavka prenosa.

Prioritet zahtjeva je relativno nov metod pristupa napravljen za standard Ethernet brzine 100 Mbps. Kao i kod CSMA/CD, dva računara mogu da počnu da prenose podatke u isto vrijeme. U tom slučaju prvo se opslužuje zahtjev višeg prioriteta. Ako se radi o zahtjevima istog prioriteta, oba zahtjeva se opslužuju naizmjenično.

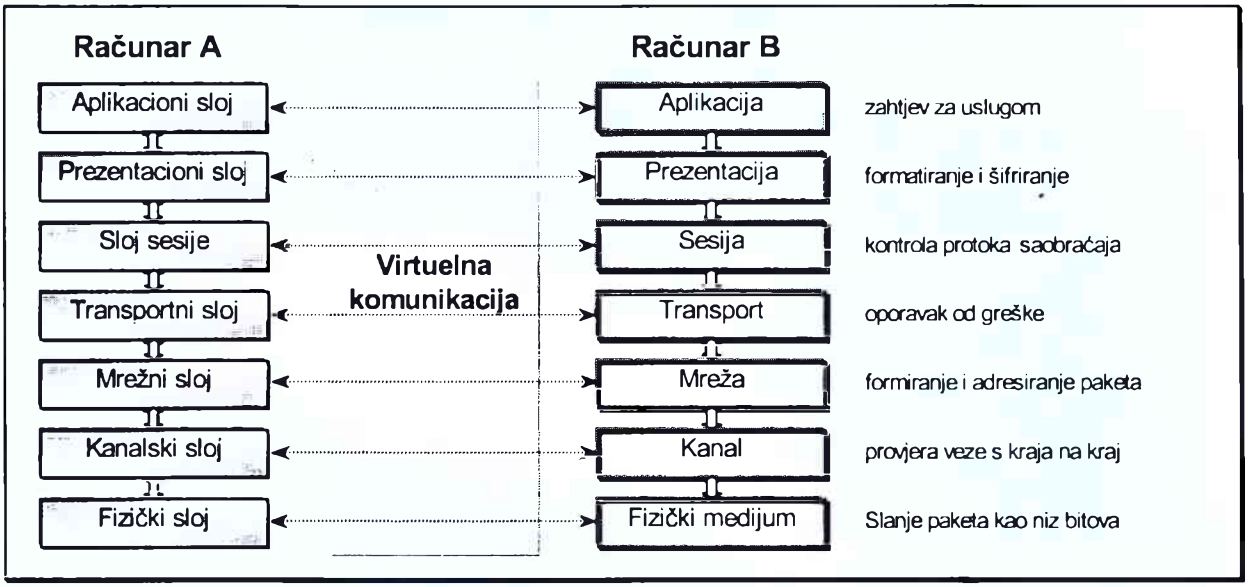
Kod prioriteta zahtjeva, postoji komunikacija samo između računara pošiljaoca, razvodnog čvora i računara primaoca. To je efikasnije nego CSMA/CD, gdje računar emituje prenos cijeloj mreži. Kada nastoji da pristupi medijumu, računari nisu u konkurenciji već su pod centralizovanom kontrolom komutacionog čvora tj. haba ili sviča. Osim toga u upotrebi su UTP kablovi kategorije 5 sa četiri para provodnika, čime je omogućen istovremeni prijem i predaja.

2.6.2 Standardi i protokoli

Distribuirani računarski sistemi zasnovani na lokalnim mrežama mogu biti veoma heterogeni i to kako u hardverskom tako i softverskom smislu. Heterogeni distribuirani sistemi mogu uključiti u svoj sastav računare različitih arhitektura i operativnih sistema. Mogućnost njihovog funkcionisanja u sastavu jedinstvenog distribuiranog sistema može

biti ostvarena samo u slučaju ako pri bitnim razlikama u arhitekturi, softveru i tehničkim realizacijama svi ti računari odgovaraju nekom jedinstvenom sistemu standarda.

Međunarodna organizacija za standardizaciju (engl. *International standards Organization*, ISO) sa sjedištem u Ženevi propisala je tzv. OSI (*Open Systems Interconnection*) specifikacije za razmjenu informacija i povezivanje otvorenih računarskih sistema. One pružaju opis kako mrežni hardver i softver rade zajedno po sistemu modela, i na taj način omogućuju komunikaciju. Ovi standardi reprezentovani su sedmoslojnim referentnim modelom [4], [5], [6], [7], [8]. Slojevi ili nivoi OSI modela su: fizički, kanalski, mrežni, transportni, sloj sesije, prezentacioni i aplikacioni, kao što je to prikazano na slici 2.1. Svaki sloj sadrži druge mrežne aktivnosti, opremu i protokole i daje svoj doprinos ukupnoj usluzi, tako da najviši sloj pruža skup usluga potreban da se mogu izvoditi distribuirane obrade. Slojevitost omogućuje nezavisnost funkcija svakog pojedinog sloja, s tim da postoji jedino zavisnost u obavljanju usluga jednog sloja za drugi.



Slika 2.1 Sedmoslojni OSI model

Postoje dva načina komunikacije između slojeva. Sa jedne strane svaki sloj razmjenjuje podatke sa susjednim nižim i višim slojevima, to je direktna veza među slojevima (vertikalna komunikacija). Drugi način komunikacije ogleda se u razmjeni informacija dva istoimena sloja abonenta koji međusobno komuniciraju. To je logički nivo komunikacije ili virtuelna veza između slojeva. Na slici 2.1 virtuelne veze su predstavljene isprekidanim linijama (horizontalna komunikacija).

Pored OSI modela, koji predstavlja uopštene preporuke za sve vrste računarskih mreža, organizacija IEEE (engl. *Institute of Electrical and Electronics Engineers, Inc.*) je početkom 80-ih definisala LAN standarde poznate pod nazivom specifikacije 802. Ova dva modela su se razvijala u približno isto vrijeme i međusobno su kompatibilni.

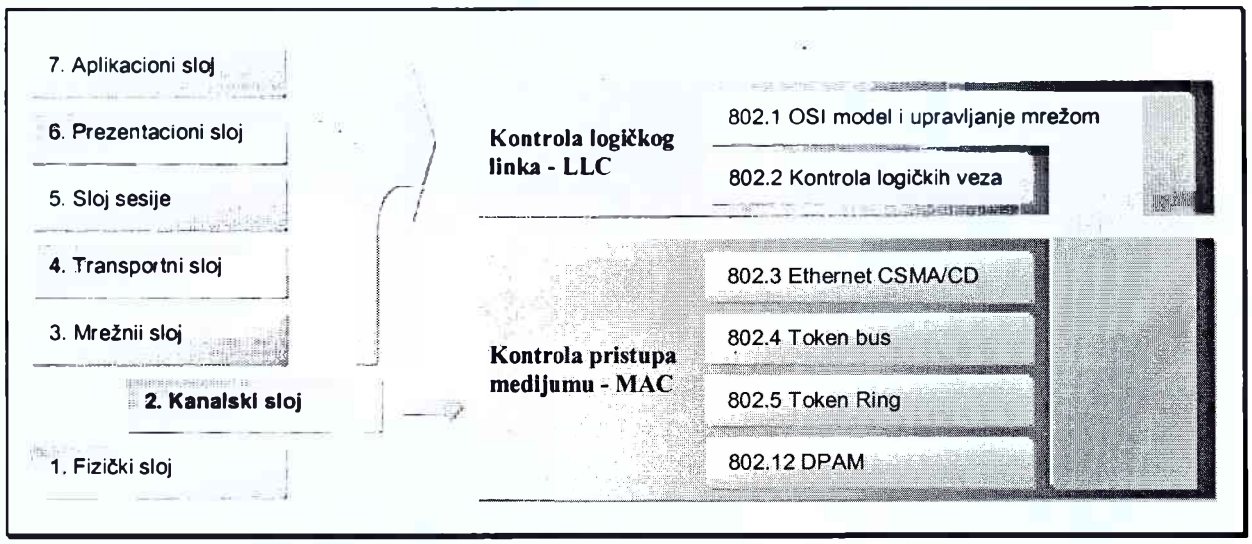
Specifikacije 802 definišu mrežne standarde za fizičke komponente mreže (mrežni adapter i kablove) kojima se u OSI modelu bave kanalski i fizički sloj. Ove specifikacije opisuju način na koji mrežni adapter pristupa i prenosi podatke preko fizičkog medijuma. Standard 802 dijeli kanalski sloj OSI modela na dva podsloja:

- podsloj za kontrolu logičkog linka, odnosno kontrolu logičkih veza - **LLC** (engl. *Logical Link Control*),
- podsloj za kontrolu pristupa medijumu i ispravku grešaka u prenosu - **MAC** (engl. *Media Access Control*).

Specifikacije ova dva podsloja prikazane su na slici 2.2 i one definišu dominantno LAN okruženje [8].

Podsloj za kontrolu logičkog linka upravlja logičkom vezom između računara tz. komunikacijskim linkom i vrši njegovu kontrolu. Ovaj podsloj pruža usluge višim slojevima OSI modela i prenosi informacije za njih. Te standarde definiše 802.2.

Kao što prikazuje slika 2.2, podsloj za kontrolu pristupa medijumu niži je od podsloja za kontrolu linka. On omogućava da više računara, sa svojim mrežnim adapterima, zajednički pristupe prenosnom medijumu odnosno fizičkom sloju. U okviru ovog podsloja nalaze si upravljački programi ili drajveri (engl. *drivers*) mrežnih adaptera. Oni omogućavaju direktnu komunikaciju između računara i mrežne kartice, obezbeđujući time logičku vezu (engl. *interface*) između računara i ostatka mreže.



Slika 2.2 Podslojevi LLC i MAC prema specifikaciji 802

Jedinstveni sistem standarda, oličen u OSI modelu i specifikacijama 802, ogleda se u jedinstvenim postupcima razmjene podataka i pravilima međusobnog povezivanja, a takođe i u mogućnostima primjene u distribuiranim sistemima proizvoljne konfiguracije. Postupci su definisani protokolima ili pravilima ponašanja učesnika u mreži. *Protokoli* su pravila i tehničke procedure koje upravljaju komunikacijom i interakcijom više računara unutar jedinstvenog sistema. Oni vode računa o svim aktivnostima, od uspostave veze do uspješnog okončanja prenosa podataka. Rad različitih protokola mora se koordinirati tako da ne bude konflikta ili nedovršenih radnji. Koordinacija između protokola se obezbjeđuje slojevitošću mrežne arhitekture.

Više međusobno zavisnih protokola radi zajedno obezbeđujući komunikacije kroz mrežu. Ti protokoli su organizovani u tz. stekove [6], [7], [8], [10]. Stek protokola je niz više međusobno zavisnih protokola. Svaki sloj ima svoj skup pravila i postupaka za prenos podataka u mrežnom okruženju i zahtjeva drugačiji protokol za rukovanje određenim funkcijom. Kako se ide odozdo nagore, prema vrhu steka, poslovi, kao i

odgovarajući protokoli postaju sve sofisticiraniji. U računarskoj industriji se nekoliko stekova prihvata kao standardni modeli protokola. Najvažniji su:

- paket protokola ISO/OSI,
- Microsoft-ov NetBEUI,
- IBM-ova SNA,
- Digital-ov DECnet,
- Novell-ov IPX/SPX,
- Paket protokola za Internet, TCP/IP.

Protokoli postoje na svim slojevima ovih stekova i obavljaju poslove koji su definisani za taj sloj.

2.6.3 Mrežne arhitekture

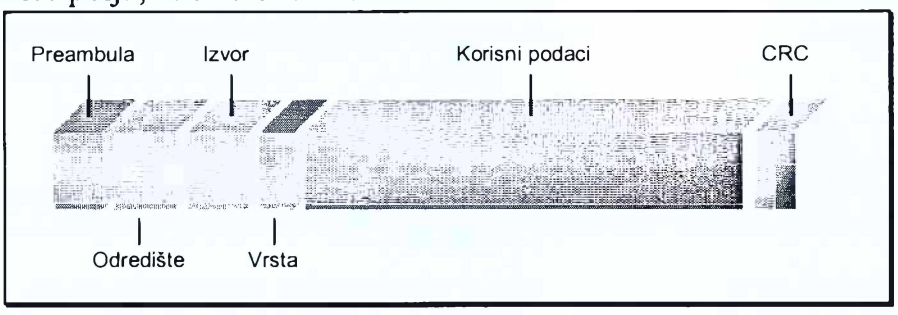
Pod arhitekturom računarske mreže podrazumijeva se ukupna struktura i sve komponente koje mrežu čine funkcionalnom [7], [8]. Najpoznatije mrežne arhitekture lokalnih mreža su:

- Ethernet,
- Token Ring,
- AppleTalk,
- Arc Net.

Mrežna arhitektura obuhvata standarde, tipologije i protokole koji svi zajedno obezbjeđuju uspješan rad odnosno ispravno funkcionisanje računarske mreže.

2.6.3.1 Ethernet

Ethernet je danas najpopularnija mrežna arhitektura. Ova arhitektura ima topologiju magistrale, nominalna brzina prenosa je 10 Mbps, 100 Mbps ili 1 Gbps i oslanja se na CSMA/CD metod pristupa prenosnom medijumu. Komunikacija se obavlja razmjenom poruka. Ethernet poruke rastavlja u pakete. Paketi se smještaju u okvire (engl. *frames*), koji se dalje prenose mrežom. Okviri su dužine od 64 do 1518 bajta, ali se za kontrolne podatke koristi najmanje 18 bajtova, tako da korisni podaci mogu biti dužine između 46 i 1500 bajta. Svaki okvir sadrži kontrolne informacije i osnovna organizacija svakog okvira je ista. Primjer okvira ethernet II, koji se koristi za TCP/IP, sastoji se od više polja, kao na slici 2.3.

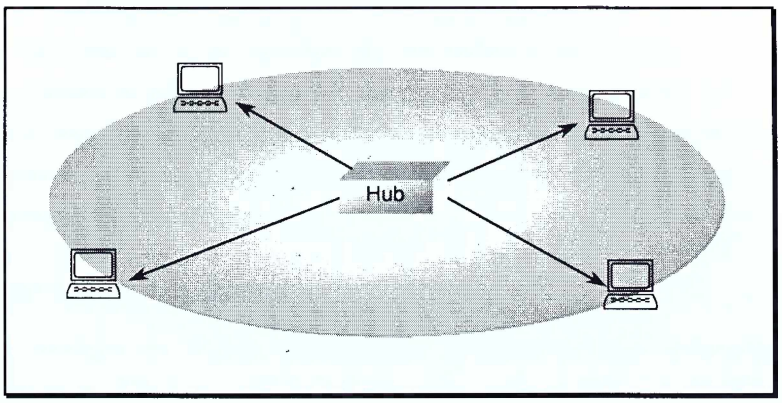


Slika 2.3 Primjer okvira Ethernet II

2.6.3.2 Token Ring

Token Ring arhitekturu je proizvela i na tržište plasirala firma IBM, pa se ona najčešće i koristi za izgradnju lokalnih mreža na bazi IBM računara. Mreža Token Ring

je zasnovana na logičkoj topologiji prstena. Međutim, u njenoj IBM implementaciji računari iz mreže povezani su fizički zvezdasto sa centralnim haba, koji se zove MAU uređaj (engl. *Multistation Access Unit*). Logički prsten predstavlja putanju tokena od računara do računara. Sam prsten je fizički implementiran u habu. Korisnici su dio prstena, ali se sa njim povezuju preko haba kao na slici 2.4.



Slika 2.4 Logički prsten mreže Token Ring

Fizička topologija Token Ring mreže je zvezdasta, odnosno kombinacija zvijezde i prstena sa predavanjem tokena kao metodom pristupa i brzinama prenosa od 4 do 16 Mbps. U Token Ring mrežama nema kolizije podataka, jer prenos može vršiti samo računar koji posjeduje token.

2.6.3.3 AppleTalk i ArcNet

AppleTalk je arhitektura za Macintosh računare. U Macintosh računare ugrađene su mrežne funkcije koje mrežu AppleTalk čine znatno jednostavnijom od drugih mreža. Pojedinačne AppleTalk mreže mogu se spojiti u veću mrežu, zahvaljujući zonama. Svaka povezana manja mreža prepoznaje se po imenu zone. Mreže sa drugim arhitekturama, recimo Token Ring, takođe se na taj način mogu spajati u AppleTalk mrežu.

ArcNet je jedna od najstarijih arhitektura za lokalne mreže. To je jednostavna, jeftina i fleksibilna mrežna arhitektura, projektovana za manje mreže veličine radnih grupa. Mreže ArcNet mogu da imaju kombinovanu topologiju zvijezde i magistrale ili samo magistralnu topologiju. U ovim mrežama koristi se predavanje tokena kao metod pristupa, sa brzinom prenosa od 2,5 Mbps. Naslednik prvobitnog ArcNet-a je ArcNet koji podržava brzine od 20 Mbps.

2.7 Sistemski softver distribuiranog računarskog sistema

Cjelokupno programsko obezbeđenje (engl. *software*) distribuiranog sistema, kao i kod svih računarskih sistema, može se podijeliti na aplikativne programe korisnika i sistemski softver. Aplikativni programi predstavljaju skupove zadataka, koje korisnici prezentiraju sistemu na obradu, a sistemski softver je neka vrsta interfejsa između aplikativnih programa i hardvera sistema. Zajedno sa hardverskom strukturom sistemski softver čini sredinu, odnosno okruženje, u kojem se mogu izvršavati aplikativni programi.

Osnovni dio sistemskog softvera čine samostalni ili mrežni operativni sistemi pod kojim rade personalni računari u mreži. Pored njih tu spadaju: komunikacioni programi odnosno mrežni softver, sistemi za upravljanje bazama podataka, kompajleri i drugi uslužni (engl. *utility*) programi.

Programsko obezbeđenje distribuiranog sistema, odnosno sistemski i aplikativni programi i resursi podataka, razdijeljeni su horizontalno po autonomnim računarima. Programi, tačnije procesi, koji se izvršavaju na jednom računaru, zahvaljujući uslugama koje pruža mreža, mogu pristupati resursima podataka smještenim na drugim lokacijama. Mrežne usluge se ostvaruju uz pomoć mrežnog ili komunikacionog softvera. Neki savremeni operativni sistemi imaju integrisane mrežne funkcije pa se mogu slobodno nazvati *mrežnim operativnim sistemima*.

2.7.1 Samostalni (lokalni) operativni sistemi

Operativni sistem je skup sistemskih programa koji obezbjeđuju interakciju korisnika sa računarom. On kontroliše i upravlja radom računarskog sistema i njegovih hardverskih resursa, kao što su: CPU, memorija, diskovi, periferni uređaji ili mrežni adapteri. Operativni sistem puni u memoriju i inicijalizira izvršavanje aplikativnih programa. On omogućava korisniku i aplikativnom programu komfornu komunikaciju sa računarskim sistemom. Tačno odvijanje procesa, koje proizvodi operativni sistem, nije uvijek predvidivo od strane istraživača, pa se njihov uticaj na odvijanje aplikativnih programa mora posredno uzimati u obzir.

Personalni računar kao dio distribuiranog sistem radi pod svojim ali može koristiti i usluge nekog, u sistemu prisutnog, mrežnog operativnog sistema. Na personalnim računarima, koji ulaze u sastav distribuiranog sistema može biti instaliran neki od sljedećih operativnih sistema:

- **MS-DOS**
- **UNIX/LINUX**
- **Windows 95/98/Me/XP**
- **Windows NT/2000 Server**
- **Windows NT Workstation, 2000 professional**
- **Novell NetWare**
- **OS/2**.

Sa izuzetkom operativnog sistema MS-DOS svi ostali operativni sistemi personalnih računara imaju mogućnost višeprocesnog rada (engl. *multitasking*), što naročito dolazi do izražaja kada računar radi u mreži. Operativni sistemi za višeprocesni rad omogućuju računaru da obavlja više poslova ili procesa u isto vrijeme. Najpovoljnije je da sistem obrađuje onoliko poslova koliko ima procesora. Ukoliko ima više poslova nego procesora, računar dijeli procesorsko vrijeme tako da raspoloživi procesori posvećuju određeno vrijeme svakom poslu, prelazeći sa jednog posla na drugi. Na taj način se postiže efekat kao da računar radi sa nekoliko poslova odjednom.

Postoje dvije osnovne vrste višeprocesnog rada:

- *Upravljani* višeprocesni rad (engl. *preemptive multitasking*), kod koga operativni sistem može da preuzme kontrolu nad procesorom u svakom trenutku i bez bilo kakve interakcije sa određenim procesom.

- *Kooperativni* višeprocesni rad (engl. *non-preemptive multitasking*), kod koga procesor nikad ne napušta posao. Posao sam odlučuje kada treba da napusti procesor. Programi koji su napisani za kooperativne višeprocesne sisteme moraju da sadrže mehanizme za vraćanje kontrole nad procesorom drugim programima i procesima. Ni jedan drugi program ne može da radi dok kooperativni program ne napusti procesor.

Za rad u mreži, zbog neprestane interakcije između operativnih sistema koji rade kao samostalni i onih koji rade kao mrežni, upravljani višeprocesni sistemi imaju niz prednosti. Npr. kada se zahtijeva prebacivanje procesora sa lokalnog na mrežni nivo, upravljani višeprocesni rad omogućava da se to obavi za kraće vrijeme na istoj hardverskoj platformi.

2.7.2 Mrežni operativni sistemi

Sve aktivnosti u mreži inicira i njima upravlja mrežni operativni sistem. On koordinira funkcije svih umreženih računara i perifernih uređaja, tako da mogu da rade kao jedinstven distribuirani sistem. Računarske mreže su obično heterogeni sistemi, pogotovu u softverskom pogledu. Razni proizvođači softvera ugradili su različita rješenja, kojima se obezbjeđuje međusobna kompatabilnost. Tako da klijentska komponenta mrežnog softver jednog proizvođača može da radi sa serverskom komponentom drugog proizvođača, bez ozbiljnijih problema.

Postoje dva osnovna dijela mrežnog softvera: dio mrežnog softvera koji se instalira na klijentu i dio koji se instalira na serveru. Tako npr. Windows 2000 Professional je klijentski orjentisan mrežni operativni sistem i instalira se ili na samostalnim računarima ili na računarima koji igraju ulogu klijenta u mreži. Windows 2000 Server je serverski orjentisan i instalira se na računarima koji će biti serveri u mreži.

Klijentski softver ima ulogu da proslijedi i preusmjeri zahtjev korisnika za mrežnim resursom, kroz mrežu, do servera sa zahtijevanim resursom. Serverski softver omogućuje korisnicima da na svojim računarima koriste podatke sa servera i njegove periferne uređaje.

Da bi se obezbijedile mrežne mogućnosti i rad u distribuiranom okruženju, nekim operativnim sistemima kao što su MS-DOS, starije verzije UNIX-a ili OS/2, treba dodati odgovarajuće komunikacione programe, odnosno mrežni softver. Savremeni operativni sistemi kao što su Windows 95/98/Me, Windows NT, Windows 2000 Server i Windows 2000 Professional imaju već integrisane mrežne funkcije. Na osnovu toga se ovi operativni sistemi nazivaju mrežnim operativnim sistemima. Oni mogu raditi i na samostalnim računarima i u mreži. U njihov sastav ulazi kompletan komunikacioni softver. Tako npr., upravljački programi ili drajveri većine mrežnih adaptera kao i protokoli koji se najčešće koriste sadržani su u operativnim sistema Windows 95/98/Me, Windows NT/2000 i Windows XP.

Puna funkcionalnost distribuiranog sistema postiže se tek primjenom mrežnih operativnih sistema. Mrežni operativni sistem:

- povezuje sve računare i periferne uređaje distribuiranog sistema,
- koordinira funkcije svih autonomnih računara i pristup zajedničkim resursima,
- određuje nivo i stepen pristupa zajedničkim resursima od strane različitih korisnika,

- pruža bezbjednost podataka i pristup podacima iz mreže,
- dodijeljuje ili oduzima privilegije korisnicima mreže,
- eventualno, može vršiti upravljanje mrežom i distribuiranim sistemom u cjelini.

Alati za upravljanje mrežom, koje sadrže neki napredniji mrežni operativni sistemi, mogu da otkriju znake nastajanja problema. Administrator distribuiranog sistema na osnovu ovih pokazatelja vrši praćenje ponašanja mreže i preduzima potrebne mjere za njeno ozdravljenje.

Tipični predstavnici mrežnih operativnih sistema, su operativni sistemi NetWare firme Novell. Svi korisnici mreže pod upravljanjem ovog operativnog sistema raspolažu jedinstvenim korisničkim interfejsom i jedinstvenim mogućnostima. Većina proizvođača mrežnih adaptera prilagođava svoje proizvode radu u mrežama pod upravljanjem ovog operativnog sistema. To znači da kupci mrežnog adaptera zasnivaju svoj izbor na uređajima koji su kompatibilni sa NetWare. Samim tim sama topologija računarske mreže postaje sporedna za operativni sistem. Ranije verzije ovog operativnog sistema mogu raditi u posvećenom i neposvećenom režimu. Novije verzije NetWare-a isključivo rade na serverima lokalnih mreža. Stanice-klijenti sarađuju sa serverom, da dobiju pristup mrežnom operativnom sistemu. Mnoge funkcije servera, takve kao: usluge dijeljenja datoteka, usluge zajedničkog štampača, upravljanje imenima, blokiranje i sinhronizacija, dostupni su korisnicima kroz nukleus operativnog sistema NetWare NCP (engl. *NetWare Core Protocol*).

2.7.3 Mrežni uslužni programi

Mrežni uslužni programi se mogu podijeliti u dvije grupe i to na: čisto mrežne (engl. *Pure Network Applications*) i uslužne programe koji mogu raditi samostalno i u mreži (engl. *Standalone Network Applications*). Čisto mrežni uslužni programi razvijeni su za primjenu u mrežama. Njihovo korišćenje na odvojenim računarima nema smisla. Druga grupa ovdje pomenutih uslužnih programa prvobitno je bili razvijeni za autonomne računare a kasnije su, višekorisničke verzije ovih programa, proširenjem njihovih mogućnosti, prilagođene su za rad u mreži, ali sa lakoćom mogu raditi i na odvojenim računarima.

Čisto mrežni uslužni programi su izgrađeni radi efikasnog korišćenja mogućnosti mreže. Svaki ima svoj odvojeni mrežni interfejs i zahtijeva ispunjavanje nekog niza "mrežnih" komandi. Najpoznatiji čisto mrežni uslužni programi su programi kojima se ostvaruju sljedeći servisi:

- emulacija terminala (engl. *terminal emulation*),
- prenos datoteka (engl. *file transfer*),
- elektronska pošta (engl. *E-mail, electronic mail*),
- slanje poruka (engl. *Massaging*),
- organizacija rada dodijele resursa (engl. *Scheduling*),
- rad u grupi (engl. *GroupWare*).

Razvojem računarskih mreža i distribuiranih sistema mnogi poznati uslužni programi su adaptirani za rad u klijent-server okolini, pa se mogu svrstati u mrežne uslužne programe. Prilikom adaptacije za rad u mrežnoj okolini, oni se razbijaju na dva

dijela: dio koja radi na klijentskoj stanici i čini korisnički interfejs i dio programa koji radi na serveru i koji uključuje operacije koje zahtijevaju značajni rad procesora.

Među njima su najpoznatiji:

- tekst procesor (engl. *Word processing*),
- elektronske tablice (engl. *Spreadsheet*),
- distribuirane baze podataka (engl. *Data Base*),
- upravljanje projektima (engl. *Project Management*).

2.8 Distribuirane baze podataka

Naglim razvojem računarstva i pojavom globalne računarske mreže Internet javila se potreba za stvaranjem velikih baza podataka i njihovo korišćenje od strane teritorijalno distribuiranih organizacija i pojedinaca. U takvim uslovima distribuirane baze podataka dobijaju sve više na značaju i poprimaju novi kvalitet. One su tipičan primjer primjene distribuiranih računarskih sistema posmatrane klase.

Distribuirana baza podataka, je baza podataka smještena ili razdijeljena na više autonomnih računara distribuiranog sistema. Osnovna razlika između centralizovane i distribuirane baze podataka leži u činjenici da su kod centralizovane baze podaci nalaze na jednoj lokaciji tj. na jednom centralnom računaru, dok su kod distribuiranih baza podaci prisutni na više lokacija odnosno više računara.

Sa rastom obima informacija i broja transakcija, kod klasičnih centralizovanih baza podataka, javljaju se dodatni problemi, među kojima su najočigledniji:

- velika količina razmjene podataka mrežom,
- smanjena pouzdanost i sigurnost podataka,
- niska opšta produktivnost,
- veliki gubici pri obradi.

Bez obzira što je kod centralizovane organizacije baze podataka lakše obezbijediti sigurnost, cjelovitost i neprotivurečnost informacija pri obnavljanju, dodatni problemi nastali povećanjem veličine baze stvaraju dodatne teškoće pri njihovoj eksploataciji. Jedno od mogućih rješenja tih problema je decentralizacija podataka. Pri decentralizaciji postiže se :

- viši stepen jednovremene obrade u zavisnosti od distribucije opterećenja,
- poboljšano korišćenje podataka u lokalu, bez potrebe udaljenih upita,
- manji gubici,
- prostije upravljanje.

Distribuirana baza podataka je skup datoteka ili relacija, koje se čuvaju na različitim računarima tj. u različitim čvorovima distribuiranog sistema i logički su povezani na taj način, da čine jedan jedinstven sistem. Veza može biti funkcionalna ili kroz kopije jedne iste datoteke. Sa tačke gledišta korisnika distribuirana baza podataka predstavlja logičku cjelokupnost lokalnih i udaljenih tj. distribuiranih podataka. Obezbeđenjem jedinstvene cjelokupnosti podataka, korisnik stiče utisak da se radi o centralizovanom sistemu, odnosno ostvaruje se transparentnost decentralizacije i distribucije podataka.

2.8.1 Struktura distribuirane baze

Sistem distribuiranih baza podataka predstavlja skup čvorova (računara) u kojima se čuvaju lokalni sistemi baza podataka. Svaki čvor je sposoban da procesira *lokalne* transakcije, tj. one transakcija koje pristupaju podacima smještenim u tom čvoru. Dodatno, čvorovi mogu učestvovati u izvršavanju *globalnih* transakcija, tj. onih transakcija koje pristupaju ostalim čvorovima u mreži. Izvršavanje globalnih transakcija zahtijeva komunikaciju između čvorova. Distribuirani računarski sistem, koji može da podrži ovakvu strukturu baze, treba da bude tako osmišljen da:

- svaki čvor ima pristup svim ostalim,
- svaki čvor mora biti osposobljen da izvršava i lokalne i globalne transakcije.

2.8.2 Prednosti i nedostatci distribucije podataka

Motivi, kojima se vode projektanti, pri izgradnji distribuiranih baza podataka a samim tim i hardversko-softverske platforme na kojoj će baza biti instalirana, mogu biti veoma različiti. Između ostalih to su: prirodna potreba dijeljenja i distribucije podataka na više lokacija, povećana raspoloživost i pouzdanost sistema a takođe i smanjenje vremena procesiranja upita. Pored očiglednih prednosti, pri distribuciji podataka prisutni su i gubici, kao što su: povećanje cijene softvera, veća vjerovatnoća grešaka i dodatni procesorski rad (engl. *overhead*).

Prednost distribucije podataka ogleda se, prvenstveno, u sposobnosti dijeljenja i pristupa podacima na pouzdan i efektan način. Ako računarska mreža omogućava komunikaciju jednog računara sa svim drugim to znači da jedan računar može pristupiti podacima smještenim na drugim lokacijama. Dislokaciju podataka treba vršiti tako da svaki računar bude sposoban da zadrži stepen kontrole nad lokalnim podacima. U centralizovanim sistemima kontrola i upravljanje bazom vrši se iz centralnog čvora tj vrši je centralni računar. Kod distribuirane baze globalni administrator baze odgovara je za čitavi sistem. Dio te odgovornosti preuzimaju administratori lokalnih baza. Zavisno od dizajna baze svaki lokalni čvor mora imati određeni stepen autonomije. Ovo je označeno kao lokalna autonomija. Mogućnost lokalne autonomije u upravljanju bazama podataka često predstavlja glavnu prednost distribuiranih baza podataka.

Kod distribuiranih baza prisutne su i već pomenute prednosti distribuiranih sistema, takve kao što su pouzdanost i raspoloživost. Naime, ako jedan čvor iz različitih razloga postane nepristupačan, ostali čvorovi moraju biti sposobni da preuzmu njegovu ulogu. Praktično to znači da se dupliranjem podataka na više čvorova obezbjeđuje da transakcija kojoj je potreban neki podatak isti može naći na više različitih čvorova. Time se obezbjeđuje da ispad jednog ili više čvorova ne mora obavezno da dovede do pada čitavog sistema.

Sistem mora detektovati ispad nekog od čvorova i preuzeti odgovarajuću akciju za njegov "oporavak" ili popravku. Sistem ne može duže vremena raditi sa nedostajućim čvorom. Konačno, kad je obavljena popravka ili zamjena čvora koji je ispao, mehanizam mora da obezbijedi njegov lak povratak natrag u sistem. Iako je oporavak čvora koji je ispao kompleksniji kod distribuiranog nego kod centralizovanog sistema, sposobnost da sistem nastavi sa radom uprkos ispadu čvora rezultira povećanju raspoloživosti.

Raspoloživost je odlučujuća kod baza podataka korištenih od starane aplikacija koje rade u realnom vremenu.

Ako upit obuhvata podatke smještene na raznim računarima, to će vjerovatno dovesti do razdvajanja upita na podupite koji se mogu izvršavati paralelno na različitim računarima. Tako paralelno procesiranje dozvoljava brže izvršavanje korisničkih upita. U slučaju kad su podaci duplirani, upiti mogu biti usmjereni od strane sistema najmanje opterećenim čvorovima tj. računarima.

Gubici, koji mogu nastati, pri distribuciji podataka ogledaju se u dodatnim kompleksnim zahtjevima za osiguranje odgovarajuće koordinacije između čvorova. Ovo povećanje kompleksnosti zahtjeva ogleda se u sljedećem:

- Povećanje cijene razvoja softvera, koje je prouzrokovano težom implementacijom distribuiranih baza podataka.
- Veća je mogućnost grešaka, jer računari sarađujući u distribuiranom sistemu rade paralelno, pa je teže osigurati korektnost algoritama. Osim toga javlja se mogućnost krajnje nepredvidivih grešaka. Vještina izgradnje distribuiranih algoritama i njihova implementacije ostaje pravi i važan prostor za istraživanje.
- Dodatni procesorski rad (engl. *overhead*). Razmjena poruka i dodatna procesiranja koja su potrebna prilikom među-računarske koordinacije predstavljaju oblik dodatnog procesorskog rada koji nije prisutan kod centralizovanih sistema.

Prilikom izbora dizajna distribuirane baze podataka, projektant mora odmjeriti prednosti i gubitke, koje donosi distribucija podataka kroz mrežu. Ovaj dizajn se kreće između pune distribucije i dizajna koji uključuje visok stepen centralizacije. Optimalni dizajn, uz najmanje troškove, se može dobiti metodama imitacionog modeliranja.

2.8.3 Organizacija podataka i načini distribucije

Principi na osnovu kojih se projektuju distribuirane baze zasnivaju se na tehnikama dupliranje programa i/ili podataka, njihovom dijeljenju po čvorovima i paralelnim izvršavanjem programa ili djelova programa na različitim računarima. Postoji nekoliko mogućnosti organizacije podataka u distribuiranoj bazi. Najvažnije načini smještaja podataka, bolje rečeno relacija, u bazu su:

- **Dupliranje** (engl. *replication*). Sistem podržava više identičnih kopija jedne iste relacije. Svaka kopija je smještena u drugom čvoru, kao rezultat se dobija dupliranje podataka. Alternativa dupliranju je smještaj samo jedne kopije relacije.
- **Dijeljenje** (engl. *fragmentation*). Relacija se dijeli na više djelova ili fragmenata. Svaki dio se smiješta u drugom čvoru.
- **Dupliranje i dijeljenje**. Ovo je kombinacija dva gornja načina. Relacija se dijeli na više dijelova. U sistemu se čuva više identičnih kopija svakog od ovih dijelova.

Kod duplirane distribucije podataka relacije se dupliraju, a kopije se nalaze na dva ili više čvorova. Takva distribucija se naziva djelimično dupliranom. U nekom krajnjem slučaju potpunog dupliranja kopije relacije su smještene u svim čvorovima sistema. Ovakav način organizacije podataka ima svoje prednosti i nedostatke. Dupliranjem podataka povećava se raspoloživost sistema, jer u slučaju ispada čvora koji sadrži neku relaciju, njena kopija se može naći na drugom čvoru. Prema tome sistem može nastaviti sa procesiranjem upita koji obuhvataju ovu relaciju uprkos ispadu jednog od čvorova. U slučaju kad većina pristupa relaciji samo čita podatke iz nje, onda više

čvorova može paralelno izvršavati upite. Više kopija rezultira tome da je veliki broj traženih podataka nađen na čvoru na kom se transakcija izvršava. U ovom slučaju dupliranje minimizira prenos podataka između čvorova.

Negativna strana dupliranja podataka ogleda se u povećanom procesorskom radu prilikom ažuriranja podataka. Sistem mora biti siguran da su sve kopije neke relacije dosljedne tj. važeće ili prave, u suprotnom doći će do pojave grešaka. Ovo znači da uvijek kad se ažurira neka relacija to se mora proširiti na sve čvorove koji sadrže njenu kopiju. Uopšteno, dupliranje podataka unapređuje performanse kod operacije čitanja i povećava raspoloživost podataka kod transakcija čitanja. Međutim, transakcije ažuriranja dovodi do povećanog rada procesora tz. overheda i povećanog saobraćaja mrežom. Problemi kontrole konkurentnog ažuriranja nekoliko transakcija na zamjeni podataka je veoma kompleksan. Ovaj problem upravljanja ažuriranjem podataka se donekle može pojednostaviti izborom jedne kopije kao primarne.

Razdijeljena distribucija podrazumijeva da se relacija može podijeliti na više dijelova. Ovi dijelovi sadrže dovoljan broj informacija pomoću kojih se može rekonstruisati originalna relacija. Ova rekonstrukcija se može izvršiti bilo uz pomoć operacije unije, bilo uz pomoć specijalne operacije spajanja pojedinih djelova ili fragmenata. Postoje dvije različite šeme dijeljenja relacija: horizontalno dijeljenje i vertikalno dijeljenje. Horizontalno dijeljenje djeli relaciju na podskupove primjeraka a vertikalno dijeljenje dijeli relaciju vršeći dekompresiju same šeme te relacije. Ove dvije šeme mogu biti primijenjene nad istom relacijom, pri čemu je moguća i pojava redundantnih podataka, tj. jedna ista informacija se može pojaviti u više različitih djelova.

2.8.4 Transparentnost i lokalna autonomija

U prethodnoj sekciji je pokazano kako relacije na različite načine mogu biti smještene u okviru distribuirane baze podataka. Neophodno je da sistem minimizira stepen u kom će korisnik znati kako je relacija smještena. Sistem mora biti u stanju da sakrije detalje distribucije podataka kroz mrežu. Ovo nazivamo providnost ili transparentnost mreže (engl. *Network transparency*).

Transparentnost mreže je u suprotnosti sa lokalnom autonomijom. Transparentnost mreže se mjeri stepenom u kojem sistem korisnika može ostati nesvjestan plana ili dizajna (engl. *design*) distribuiranog sistema. Lokalna autonomija se mjeri stepenom u kojem projektant ili administrator jednog čvora može ostati nezavistan od ostalog dijela distribuiranog sistema. Transparentnost i autonomija se regulišu, odnosno optimiziraju sljedećim postupcima:

- imenovanjem elementarnih podataka,
- dupliranjem elementarnih podataka,
- dijeljenjem elementarnih podataka,
- lociranjem djelova i kopija podataka.

Svaki elementarni podatak u bazi mora imati jedinstveno ime. Ovo svojstvo je lako obezbijediti u centralizovanoj tj. nedistribuiranoj bazi. Međutim, u distribuiranoj bazi različiti čvorovi moraju obezbijediti da dva čvora ne koriste istu identifikaciju za razne elemente podataka.

Jedno rješenje ovog problema je zahtjev da sva imena budu registrovana u nekom centralnom serveru imena. Ovakav prilaz ima niz nedostataka:

- server imena postaje "usko grlo",
- u slučaju kvara servera imena, može se desiti da neki od čvorova ili čitav sistem ne može da nastavi da radi,
- umanjuje se lokalna autonomija, jer je imenovanje elementarnih podataka kontroliše centralistički.

Alternativni prilaz koji dovodi do povećanja lokalne autonomije je zahtjev da svaki elemenat podataka generisan na nekom čvoru ima jedan dodatni identifikator čvora. Ovo garantuje da dva čvora neće generisati isto ime jer svaki čvor ima jedinstvenu identifikaciju i centralna kontrola nije potrebna.

2.8.5 Implementacija distribuiranih baza podataka

Mnogi problemi centralizovanih baza postoje i kod distribuiranih, ali distribucija donosi i nove. Da bi se distribuirana baza podataka, mogla na zadovoljavajući način instalirati na distribuiranom računarskom sistemu on mora da ispuni niz dodatnih zahtjeva, među kojima treba istaći sljedeće:

- Jedinstven opšti model podataka distribuiranog sistema. Tj. mora postojati jedinstvena koncepcijska šemu čitavog sistema. To krajnjem korisniku obezbjeđuje mogućnost logičke transparentnosti podataka. Na taj način korisnik stiče utisak kao da radi sa centralizovanom bazom podataka tj. može zadati upit nad cijelom bazom.
- Neophodnost postojanja šema koje određuje lokaciju ili mjesto gdje se nalaze podaci u sistemu. To omogućava transparentnost razmještaja podataka u mreži, zahvaljujući čemu korisnik može, ne ukazujući eksplicitno gdje je poslao upit, da dobije tražene podatke.
- Transparentnost konverzije softvera. Distribuirani računarski sistemi mogu biti homogeni ili heterogeni u smislu hardverskih i softverskih sredstava. U ovom radu se uglavnom proučavaju hardverski homogeni distribuirani sistemi, tj. sistemi na bazi personalnih računara, pa sistem može biti samo nehomogen u softverskom smislu. Ukoliko je to slučaj, potrebno je obezbijediti transparentnost konverzije softverskih struktura u čvorovima distribuiranog sistema.
- Organizacija i upravljanje rečnicima podataka. Za obezbeđenje svih vidova transparentnosti u distribuiranom sistemu su potrebni programi koji će obezbijediti upravljanje mnogobrojnim rečnicima podataka.
- Koordinacija procesa. Metode ostvarivanja upita u distribuiranu bazu podataka razlikuju se od sličnih metoda centralizovanih baza. Dijelove upita potrebno je obrađivati na mjestu gdje se nalaze odgovarajući podaci i prenositi samo odvojene rezultate obrade na druge čvorove. Pri tome se mora obezbijediti koordinacija svih procesa u sistemu.
- Sinhronizacija paralelne obrade. U distribuiranom sistemu nužno je obezbijediti složen mehanizam upravljanja istovremenom obradom, koji mora da obezbijedi sinhronizaciju pri obnavljanju informacija, što garantuje neprotivurečnost podataka.
- Razvijena metodologija distribucije i razmještaja podataka.

2.9 Distribuirani aplikativni programi

Progres u razvoju arhitektura računarskih sistema i bitno povećanje njihove produktivnosti, zasniva se na sve većem stepenu paralelizma u radu sistema. *Paralelizam* se, kod distribuiranih računarskih sistema, prvenstveno ogleda u angažovanju dva ili više računara na rješavanju jednog istog zadatka. Korisnički programi, koji se paralelno tj. istovremeno izvršavaju na autonomnim računarima distribuiranog sistema, a rade zajedno na obavljanju istog posla, označeni su kao paralelni aplikativni programi. U cilju istraživanja, skup paralelnih aplikativnih programa se mora posmatrati kao jedinstven sistem. Njihovo odvojeno posmatranje ne bi imalo nikakvog smisla, pa se ovakvi sistemi paralelnih programa označavaju kao *distribuirani aplikativni programi* ili distribuirane aplikacije. Tipičan primjer distribuiranih aplikativnih programa jesu programi izrađeni uz pomoć tehnologije klijent/server.

2.9.1 Aplikativni programi u distribuiranoj računarskoj sredini

U distribuiranim računarskim sistemima i računarskim mrežama postoji mogućnost izgradnje centralizovanih i distribuiranih aplikativnih programa. Centralizovani aplikativni programi su mrežni programi, koji se pripremaju za izvršavanje na jednom centralnom (engl. *host*) računaru. Sve obrade i izračunavanja obavlja centralni procesor te mašine, a rezultati obrade se zatim prenose mrežom do mjesta gdje se eksploatišu. Autonomni računari ili radne stanice povezane u mrežu, mogu i samostalno vršiti obradu tj. mogu imati i samostalne aplikativne programe, kako svojih lokalnih, tako i mrežnih podataka. U tom slučaju obrada se obavlja tamo gdje su se i javili zahtjevi za njom, međutim obim podataka, koji mora biti prenesen mrežom, može biti vrlo veliki. Za razliku od centralizovanih, kod *distribuiranih aplikacija* obrada je razdijeljena, tj. distribuirana i obavlja se paralelno na više računara. Time se minimizira obim podataka koji se prenosi mrežom i smanjuje vrijeme obrade. Distribuirane aplikacije objedinjuju dobre strane pomenutih tehnologija samostalnih i centralizovanih aplikacija.

Distribucijom aplikacije ukupni program se dijeli na cjeline (engl. *partitioning*) koje se mogu izvršavati paralelno na različitim procesorima, odnosno računarima. Svakom računaru se dodijeljuje jedan dio posla (engl. *assignment*). Računari, tačnije njihovi procesori, razmjenom međurezultata i sinhronizacijom međuaktivnosti, sarađuju u ostvarivanju jedinstvenog posla. Distribuirane aplikacije, u ovakvom sistemu, moraju biti osposobljene za detekciju otkaza i njegovo prevazilaženje. Izrada kriterijuma za dodjeljivanje dijelova posla pojedinim procesorima, sinhronizacija međuaktivnosti kao i provjera otkaza zahtijevaju dodatne kompenzacije u softveru i hardveru, čime se povećavaju složenost i cijena distribuiranih aplikativnih programa.

Distribucija programa i ostalih hardversko-softverskih resursa, unutar jednog ovakvog računarskog sistema, može biti logička i fizička. S obzirom na lokaciju izvršavanja programa a samim tim i način distribuiranosti razlikuju se sledeće vrste računarskih sistema:

1. Logički i fizički nedistribuirani sistemi. To su klasični računarski sistemi kao i multiprogramski sistemi kod kojih se programi izvršavaju kvazi paralelno, dijeleći vrijeme jednog istog centralnog procesora.
2. Logički distribuirani sistemi na fizički nedistribuiranom hardveru. To su, već pomenuti, multiprocesorski sistemi sa djeljivom memorijom, kod kojih se komunikacija između paralelnih programskih procesa ostvaruje uz pomoć zajedničke memorije.
3. Logički nedistribuirani sistemi, na fizički distribuiranom hardveru. Tu spadaju lokalne računarske mreže, ali bez implementiranih distribuiranih aplikacija. Ovdje se pokušava podržati ideja apstraktne globalne memorije. Obrada se obavlja na individualnim računarima u mreži i nema direktne saradnje, na rješavanju zadataka, između programa u sistemu.
4. Fizički i logički distribuirani sistemi. To su pravi distribuirani sistemi kod kojih se programski procesi, jednog istog posla, odvijaju paralelno na više računara. Oni međusobno komuniciraju razmjenom poruka. Kod ovih sistema najčešće se primjenjuje, danas veoma popularna, *klijent/server* tehnologija.

2.9.2 Izgradnja paralelnih i distribuiranih programa

Zadatak paralelnog programiranja u distribuiranim sistemima sastoji se u razradi metoda i mehanizama sinteze distribuiranog programa, koji će obaviti zadatke skupa procesa izvršavanih na autonomnim računarima. Ovaj zadatak se u suštini svodi na problem izgradnje paralelnog programa na osnovu zadatog grafa zavisnosti i on se može rješavati na različite načine. Najprostiji način rješavanja ovog problema se svodi na razbijanje grafa na *spratove* ili paralelne korake. Svaki sprat sadrži paralelne nezavisne aktivnosti i spratovi slijede sekvencijalno jedan za drugim. To znači da aktivnosti nekog sprata mogu započeti samo poslije toga, kada se završe sve aktivnosti prethodnog sprata.

Da bi bilo isprogramirano paralelno izvršavaju operatora (po spratovima), uvodi se pojam *paralelnog operatora,* sastavljenog od nezavisnih operatora. Na taj način paralelni program će predstavljati niz običnih i paralelnih operatora. Nešto razvijenija od ove metode je metoda *paralelnih grana.* U tom slučaju paralelni operator (segment) sastoji se od nezavisno i paralelno izvršavanih grana tj. fragmenata programa, koji mogu biti operatori ili nizovi operatora. Segment ima početak tj. tačku razdvajanja paralelnih grana i kraj tj. tačku spajanja paralelnih grana. Glavna razlika između metoda paralelnih segmenata i metode paralelnih operatora sastoji se u tome što se upravljanje izračunavanjima dijeli na lokalna upravljanja granama. Metoda upravljanja granama daje fleksibilniju organizaciju paralelnih odnosno distribuiranih programa.

U obe razmatrane metode dobijaju se programi koji se mogu nazvati paralelno-sekvencijalnim. Oni predstavljaju nizove paralelnih operatora ili paralelno izvršavane sekvencijalne grane. Dijelovi distribuiranih aplikativnih programa, koji se izvršavaju na autonomnim računarima mogu se, sa izvesnim aproksimacijama, posmatrati kao sekvencijalni grane jedinstvenog paralelnog programa.

Dva jednoznačna paralelna programa su ekvivalentna, ako je za iste početne podatke bilo koji proces stvoren od strane jednog programa ekvivalentan nekom procesu, stvorenom drugim programom. U klasi ekvivalentnih paralelnih programa mogu se uvesti sljedeće relacije: neki program nije manje paralelan od drugog, ako rađa

skup procesa koji je nadskup procesa koji rađa drugi program. *Maksimalno paralelni program* je onaj koji nije manje paralelan, nego ma koji njemu ekvivalentan program. Maksimalno paralelni programi izvršavaju se optimalno na distribuiranim računarskim sistemima različitih konfiguracija kao i na sistemima kod kojih se konfiguracija dinamički mijenja tj. sistemima koji u toku izvršavanja distribuiranih paralelnih programa nepredvidivo mijenjaju broj dostupnih računara ili brzinu rada.

Paralelno-sekvencijalni način organizacije distribuirane obrade predstavlja prelaznu formu od sekvencijalnog prema visoko-paralelnom. U osnovi tog metoda leži tradicionalna sekvencijalna organizacija programa, proširena dodatkom specijalnih sredstava za označavanje nezavisnih fragmenata koji se paralelno izvršavaju. Metode paralelnih operatora i paralelnih grana tipično se koriste u strukturama upravljanja ovih programa.

Opšte karakteristike ovih programa su:

- odsustvo bilo kakvih međudejstava između paralelnih fragmenata,
- upravljanje obradom se sastoji u tom da poslije izvršavanja svakog paralelnog operatora ili segmenta tačno se ukazuje koji se operatori ili segmenti izvršavaju na sljedećem koraku,
- komunikacija se obavlja razmjenom poruka.

Prilikom izgradnje paralelnih distribuiranih programa potrebno je uzeti u obzir kako matematičke osobenosti zadataka koji se rješava, tako i konfiguraciju računara i distribuiranog sistema u cjelini. Za postizanje efektivnog ubrzanja izračunavanja neophodno je ne samo izabrati odgovarajući metod izračunavanja, nego je potrebno i poznavati strukturne osobine distribuiranog sistema na kom se obavlja obrada. Direktnom transformacijom efektivnog sekvencijalnog algoritma u paralelni on može postati neefektivan i obrnuto.

2.9.3 Tehnologija klijent/server

Ključnom komponentom u oblasti naprednih računarskih tehnologija danas se bez sumnje javlja tehnologija klijent/server. Realno, to je univerzalni model koji služi kao osnova za izgradnju sistema ma koje složenosti, pa prema tome i mrežnih. Projektanti sistema za upravljanje bazama podataka, komunikacionih mreža, sistema elektronske pošte, bankarskih sistema i sl. intenzivno koriste ovu tehnologiju. Ona dobija posebno na značaju okretanjem savremenog tržišta računara jeftinim i visokoproduktivnim personalnim računarima i njihovim povezivanjem u lokalnim računarskim mrežama. Time ova tehnologija postaje veoma konkurentna alternativa centralizovanim aplikativnim obradama i izračunavanjima.

Tehnologija klijent/server u potpunosti mijenja metode pisanja, izvršavanja i servisiranja aplikativnih programa. Ova tehnologija, najjednostavnije rečeno, predstavlja podjelu distribuiranog aplikativnog programa na dvije logički odvojene komponente, od kojih svaka izvršava svoje odvojene funkcije. U distribuiranom računarskom sistemima klijent/server arhitekture prisutan je jedan ili više procesa server i jedan ili više procesa klijent. Proces klijent zahtijeva operaciju ili uslugu koju će neki drugi proces označen kao server obezbijediti ili pripremiti. Zahtjev za uslugom tj. za izvršavanjem određenog

zadatka, klijent šalje serveru u vidu poruke. Na dospjeli zahtjev klijenta, server izvršava zahtijevani servis i vraća klijentu rezultat, opet u vidu poruke.

Zadatak servera je obrada zahtjeva i vraćanje rezultata klijentu. Proces server se u distribuiranom računarskom sistemu odvija na računaru koji je fizički odvojen od klijentske stanice a sa njom je povezan lokalnom računarskom mrežom. Na klijentskim stanicama se odvijaju procesi primarnih obrada, koji koordiniraju logiku dijela aplikacije na serveru, sa mehanizmom komunikacije lokalnih mreža.

Računari na kojima se izvršavaju serverski procesi obično su boljih performansi, da bi mogli da obave poslove koji dolaze od drugih računara tj. da bi mogli da opsluže veći broj klijenata. Faktički u skladu sa logikom unutrašnje obrade aplikacije može se pojaviti više servera.

Arhitektura klijent/server dozvoljava izvršavanje jednog posla paralelno na dva ili više računara koristeći mogućnosti obrade svakog od njih. Ovakva distribucija aplikacije obezbeđuje elastičnost i dozvoljava izvršavanje aplikacije ili dijela aplikacije tamo gdje je to najefikasnije. Sa stanovišta programiranja, veoma je ugodno dozvoliti komponentama aplikacije da se baziraju ili na klijentu ili na serveru. Prenoseći dijelove aplikativnih programa na klijent ili server, projektant može dobiti optimalne rezultate. U nekim sistemima dozvoljava se dinamičko određivanje mjesta obrade tj. u toku etape izvršavanja.

Dijeljenjem aplikacije na klijentski i serverski dio, odnosno njeno distribuiranje na više računara, ima prednosti za sve koji se sreću sa informacionim sistemom kompanija. Dolazi do uvećanje opšte propusne sposobnosti, produktivnosti, mogućnosti rada u multitasking okruženju, elastičnosti i potpunijeg iskorišćenje svih resursa sistema što je veoma bitno za korisnike. Kompanijama je od interesa integracija diskretnih komponenata i njihovo funkcionisanje kao jedinstvene cjeline. To povećava efikasnost i smanjuje troškove obrade podataka.

Izvršavanjem programa na više računara, može se potencijalno povećati produktivnost i obim izračunavanja. Međutim, arhitektura klijent/server ne može se primijeniti u svim slučajevima obrade podataka. Nepravilnim projektovanjem ili slabom realizacijom ove tehnologije može se ne samo sniziti produktivnost, nego to može dovesti do znatnog uvećanja problema upravljanja. Kao i u slučaju bilo koje druge tehnologije obrade podataka, i tehnologija klijent/server ima svojih prednosti, koje treba iskoristiti i nedostataka koji se moraju izbjeći.

3. ANALIZA EFIKASNOSTI DISTRIBUIRANIH RAČUNARSKIH SISTEMA

3.1 Uvod

Kao što je već bilo napomenuto u uvodnom poglavlju, cilj istraživanja u ovom radu je prognoza efikasnosti, izgradnja metoda i načina analize, projektovanja i eksploatacije distribuiranih i paralelnih računarskih sistema i računarskih mreža. Pod efikasnošću svakog, pa i distribuiranog računarskog sistema podrazumijeva se koliko taj sistem odgovara svojoj namjeni, tj. koliko ponašanje, odnosno funkcionisanje, sistema odgovara cilju za koji se primjenjuje. Efikasnost određuju veličine koje su označene kao pokazatelji efikasnosti ili pokazatelji performansi sistema.

Predmetom ovog rada javljaju se različiti aspekti ponašanja distribuiranih računarskih sistema. Poseban akcenat se stavlja na posmatranje onih karakteristika distribuiranih sistema koje su povezane sa algoritamskim i informacionim aspektima njihovog ponašanja, odnosno njihov uticaj na najvažnije pokazatelje efikasnosti.

3.2 Formulacija zadatka analize distribuiranog sistema

Problem analize performansi distribuiranog računarskog sistema može se, u opštem slučaju, formulisati na sljedeći način: zadat je distribuirani računarski sistem S kao kompleks mnogobrojnih hardverske i softverske komponenti i zadat je skup distribuiranih aplikativnih programa P. Potrebno je naći način procjene pokazatelja performansi distribuiranog računarskog sistema S na skupu programa P bez korišćenja samog sistema S ili njegovog prototipa.

Ovakva formulacija zahtijeva i dodatna razjašnjenje. Odmah se postavlja pitanje: šta je poznato i kako su zadate hardverska i softverska struktura sistema? Kako je definisan skup korisničkih zahtjeve za uslugama, sadržan u aplikativnim programima koji se izvršavaju na zadatom hardveru uz zadatu sistemsko-softversku podršku? Koji su to pokazatelji performansi koji karakterišu dati računarski sistem? Potrebno je definisati i sam pojam produktivnosti distribuiranog računarskog sistema. U sljedećim poglavljima će biti detaljnije razjašnjena gornja formulacija zadatka analize distribuiranog računarskog sistema, što je neophodno za njegovo rješavanje.

3.3 Glavne komponente distribuiranog računarskog sistema

U cilju analize i istraživanja cjelishodno je izvršiti dekompoziciju posmatranog sistema. Shodno tome, unutar distribuiranog računarskog sistema mogu se identifikovati tri glavna dijela:

- *Fizička sredina*, koju čine hardverski elementi sistema i komunikaciona mreža. Ona predstavlja sredinu u kojoj se odvijaju svi kako sistemski tako i aplikativni programi. Fizička sredina se još naziva i fizička struktura ili hardverska struktura.

- *Izvršna sredina*, pod kojom se podrazumijeva skup sistemskih softverskih procesa i izvršnih korisničkih programa. Oni čine interfejs između korisnika i fizičke strukture i omogućavaju izvršavanje korisničkih zahtjeva u fizičkoj sredini.
- *Radno opterećenje*, pod kojim se podrazumijeva niz korisničkih zahtjeva kao što su npr. zahtjevi za uslugom upisa ili čitanja iz baza podataka. To je niz korisničkih zahtjeva koji sadrži njihove karakteristike i opis redoslijeda kojim oni dolaze u sistem. Ovaj niz korisničkih zahtjeva najčešće je sadržan u okviru aplikativnih programa, pa se radno opterećenje distribuiranog sistema, može definisati i kao *ponašanje* distribuiranih aplikativnih programa.

Prva dva dijela su označena kao *distribuirana računarska sredina* ili hardversko-softverska struktura posmatranog sistema. U ovom kontekstu, termin "distribuiran" označava decentralizaciju upravljanja i kontrole fizičke i izvršne sredine jednog ovakvog sistema.

U drugom poglavlju su već opisane osnovne karakteristike distribuirane računarske sredine, ovdje će biti izdvojene samo one koje su od posebnog interesa za ovo istraživanje. Kako je već naglašeno posmatra se *klasa distribuiranih računarskih sistema*, čija je fizička sredina okarakterisana sa:

1. Postojanjem sistema autonomnih personalnih računara klasične Fon-nojmanove arhitekture. Oni posjeduju jedan ili više mikroprocesora opšte namjene, svoju lokalnu operativnu memoriju, a opciono mogu posjedovati: uređaje spoljašnje memorije, različite periferne uređaja i slično.
2. Računari mogu raditi samostalno i odvojeno od ostalog dijela sistema, pod upravljanjem svog operativnog sistema. Oni mogu biti različitih tehničkih performansi, ali su u potpunosti autonomni i ravnopravni u upravljanju. Samim tim u ovakvom distribuiranom sistemu pojavljuje se nekoliko tokova upravljanja koji uopšteno govoreći imaju različite brzine.
3. Svaki računar posjeduje individualni generator takta i samim tim sopstveno logičko vrijeme, različito od logičkog vremena drugih računara u sistemu. Odavde proizilazi da sistem mjerenja vremena nije jedinstven. Može se reći da svaki računar radi u svom sistemu vremena
4. Računari su međusobno povezani brzim komunikacionim kanalom, čineći lokalnu računarsku mrežu visoke propusne moći, koja obezbjeđuje razmjenu podataka između računara, odnosno procesora. Vrijeme prenosa podataka ovim kanalom srazmjerno je vremenu pristupa procesora njegovoj lokalnoj operativnoj memoriji.
5. Pretpostavlja se da je struktura veza je statična, tj. u toku rada sistema, odnosno tokom jedne serije istraživanja, broj računara se ne mijenja.
6. Svi uređaji spoljašnje memorije u sistemu su opštedostupni, tj. ma koji procesor pomoću odgovarajućih sredstava može čitati iz ili zapisivati u spoljašnju memoriju drugog računara proizvoljan obim podataka.
7. Izvršna sredina ima veliki broj djeljivih (engl. *sharable*) resursa i to kako sistemskih i aplikativnih procesa tako i resursa podataka.
8. Upravljanje programima i podacima je decentralizovano. Samim tim pored fizičke i izvršna sredina sistema je distribuirana. Zbog decentralizacije i postojanja nekoliko tokova upravljanja različitih brzina bitno se komplikuju problemi sinhronizacije

procesa. To u njenu organizaciju uvodi dodatne probleme, jer upravljanje softverskim resursima sistema postaje bitno složenije.

9. Procesi, koji se odvijaju na različitim računarima, nemaju opštih promjenljivih. Oni mogu uzajamno sarađivati samo uz pomoć nadležnih sredstava distribuirane računarske sredine. Ovo znači da je saradnja među procesima koji se izvršavaju na odvojenim računarima uvijek prikazana javno i to se najčešće obavlja putem razmjene poruka. Pri tom je uvijek ukazan inicijator, adresat, i tip objekta pomoću kojeg se ostvaruje uzajamno dejstvo. Uzajamno dejstvo se koristi kako za prenos poruka i podataka, ali i u cilju sinhronizacije.
10. Svi procesi u programima su konačni a proces koji se odvija na jednom od računara ne može biti prekinut procesom sa drugog računara.
11. Radno opterećenje sistema je opisano aplikativnim programima, koji se izvršavaju paralelno u klijent/server okruženju. Njih karakteriše asinhrono uzajamno dejstvo aplikativnih procesa. Nikakve pretpostavke o relativnoj brzini izvršavanja aplikativnog programa se ne koriste. Tj. aplikativni programi ne rade u realnom vremenu.

Opisana klasa distribuiranih računarskih sistema na osnovu personalnih računara je veoma rasprostranjena i ima raznovrstan spektar primjene. Tome je doprinela, prije svega, *otvorena arhitektura* ovakvih sistema odnosno način njihove organizacije koji dozvoljava modularni princip izgradnje sistema, brzo proširenje lokalnih memorija i kapaciteta autonomnih računara, jednostavno uvećavanje broja čvorova tj. autonomnih računarskih jedinica, komfornost pri organizaciji klijent/server arhitekture itd. Primjeni personalnih računara u realizaciji distribuiranog računarskog sistema znatno su doprinijeli i mogućnost rada u realnom vremenu, razvijene logičke mogućnosti, kao i bogata programska podrška.

3.4 Matematička sredstva opisa distribuiranog sistema

Problemi procjene i analize efikasnosti distribuiranih računarskih sistema pripadaju oblasti *teorije računarskih sistema*. Ova teorija se može posmatrati kao dio informatike koja matematičkim sredstvima istražuje ponašanje računarskog sistema. Predmetom teorije računarskih sistema javljaju se relevantne karakteristike ponašanja ovih sistema ili pokazatelji njihove efikasnosti. Kao primjeri pokazatelja efikasnosti ili performansi distribuiranog računarskog sistema mogu se izdvojiti: njegova produktivnost, sigurnost, otpornost na ispade i slično. Performanse distribuiranog sistema zavise od raznih parametara sistema takvih kao što su: fizička struktura sistema, način pristupa u sistem, sistemsko-softverska podrška sistema, sastav i karakter radnog opterećenja i sl. *Parametri sistema* predstavljaju matematičke objekte, npr. veličine, koja određuju arhitekturu i organizaciju računarskog sistema, broj i strukturu uređaja, njihovu brzinu, kapacitet memorije i sl. To takođe mogu biti programski moduli, algoritmi i strukture itd. *Pokazatelji performansi* sistema određuju ponašanje distribuiranog sistema i funkcije su parametara sistema.

3.4.1 Logička struktura distribuiranog računarskog sistema

Distribuirani računarski sistema može biti zadat u formi logičke ili funkcionalne strukture (engl. *Logical design*). Ovakav pogled je cjelishodan sa tačke gledišta proučavanja distribuiranog sistema, a takođe i pri rješavanju niza problema kod njegovog projektovanja. Logička ili funkcionalna struktura distribuiranog računarskog sistema ima svoj odraz, tj. preslikava se u njegovu hardversko-softversku strukturu. Na osnovu nje izgrađuje se koncepcija, odnosno, koncepcijski model distribuiranog računarskog sistema, koji je u fazi projektovanja.

Pod logičkom strukturom distribuiranog računarskog sistema podrazumijeva se konačan skup Φ uzajamno zavisnih funkcija sistema. Skup Ψ uzajamno zavisnih hardverskih elemenata, sistemsko-softverskih modula, aplikativnih programskih rutina i operativnih procedura predstavlja posmatrani distribuirani računarski sistem i realizuje zadanu logičku strukturu. Između logičke strukture sa jedne strane i hardversko-softverske strukture sa druge strane nema jednoznačnog odnosa, jer se zadana logička struktura može u opštem slučaju realizovati različitim fizičkim strukturama i sa različitom softverskom podrškom [29], [123]. Izbor Φ i nalaženje najboljeg odnosa Φ i Ψ je jedan od najvažnijih zadataka projektovanja distribuiranih računarskih sistema. Fizička struktura zajedno sa sistemskom i aplikativnom programskom podrškom, kako je već naglašeno, čine distribuiranu računarsku sredinu u kojoj se ostvaruju korisnički zahtjevi.

3.4.1.1 Funkcije distribuiranog sistema

Analiza mnoštva zadataka, koje treba da rješava distribuirani računarski sistem, pokazuje da je skup funkcija sistema Φ celishodno razbiti na niz podskupova tj.

$$\bigcup_{i \in I} \Phi_i = \Phi . \qquad (3.1)$$

Tako mogu biti izdvojene:

- mrežno-komunikacione funkcije sistema Φ_m,
- funkcije prezentacije podataka Φ_p,
- funkcije obrade podataka Φ_o,
- servisne funkcije mreže Φ_s,
- dijagnostičke funkcije Φ_d,
- funkcije obezbjeđenja veće pouzdanosti sistema Φ_p,
- funkcije organizacije podataka Φ_f,
- funkcije upravljanja distribuiranim bazama podataka Φ_b,
- kontrolno-upravljačke i sinhronizacione funkcije distribuirane obrade Φ_c,
- funkcije nadgledanja i upravljanja mrežom Φ_u i mnoge druge.

Mrežne funkcije distribuiranog sistema, spadaju u osnovne funkcije računarskih mreža i distribuiranih računarskih sistema. One su povezane sa prenosom podataka i upravljanjem tokovima podataka. Ovdje takođe spadaju kompleksni zadaci uspostave, vođenja i raskidanja veze. Mrežne funkcije se realizuju uz pomoć tri nivoa protokola: fizičkog, kanalnog i mrežnog. Ove funkcije su najčešće skoncentrisane u okviru mrežnih adaptera i njihovih drajvera. Komunikacione funkcije najčešće obezbjeđuju mrežni operativni sistemi.

Funkcije prezentacije podataka obezbjeđuju interfejs distribuiranog sistema sa krajnjim korisnicima. Ove funkcije, koje se još mogu označiti i kao terminalsko-korisničke funkcije, uključuju u sebi takve elemente kao što su: editovanje podataka relevantnih za funkcionisanje mreže i njihovu kontrolu, prikaz ovih podataka na monitoru ili štampaču, zaštita od nedozvoljenog pristupa i drugo.

Servisne funkcije distribuiranog sistema povezane su sa prikupljanjem statističkih podataka o razmjeni podataka, opterećenju, stanju komunikacionih portova, sa vođenjem arhive, čuvanjem poruka, realizacijom veze sa operaterom, prikazom stanja sistema operateru i nizom drugih pomoćnih operacija. Ove funkcije realizuje mrežni operativni sistem na server mašinama u mreži.

Rad dobro projektovane i instalirane lokalne računarske mreže odvija se neprimjetno. Poruke i podaci se brzo prosleđuju, bez posebnog angažovanja korisnika, pa se može reći da je mreža za njih providna (engl. *transparent*). Da bi se to ostvarilo potrebno je obezbijediti dobru kontrolu i upravljanje mrežom. *Nadzor, kontrola i upravljanje mrežom* kao i prikupljanje statističkih podataka o njenom radu mogu biti funkcije pojedinih za to specijalizovanih računara u mreži npr. mrežnih servera. Ove funkcije su obuhvaćene podskupom Φ_u.

Posebnu pažnju pri upravljanju mrežom treba obratiti na globalno upravljanje prometom. Dođe li do zagušenja, zbog više uzastopnih kolizija ili nemogućnosti predaje pristiglih paketa nekom od susjednih čvorova ili abonenata, može lako doći do pada performansi čitave mreže. Pored već navedenih, u funkcije upravljanja mrežom spadaju i startovanje, inicijalizacija i oporavak mreže.

Sistemi za dijagnostiku i vođenje statistike obezbjeđuju *dijagnostičke funkcije* i sastoje se od više podsistema kao što su: podsistem za kontrolu, izvještavanje i prikupljanje podataka za cijelu mrežu ili neki njen određeni segment, podsistem evidencija o količini saobraćaja i ostvarenom prometu, podsistem kontrole i izvještavanje na nivou svakog komunikacionog čvora, podsistem analize protokola, statističke analize i tome slično. Među ovim podsistemima ne postoji oštra granica i oni se međusobno prepliću.

3.4.2 Pokazatelji performansi distribuiranog sistema

Pokazatelji performansi (engl. *Performance inductors*), ili kratko performanse, opisuju distribuirani sistem sa tačke gledišta korisnika i karakterišu ga, ako je neophodno, kao podsistem u okviru većeg sistema npr. nekog šireg distribuiranog sistema ili globalne računarske mreže. Na pokazatelje performansi, koji su označeni skupom $Y = \{y_1, y_2, \ldots, y_j, \ldots, y_m\}$, moraju se primijeniti određena ograničenja koja potiču od njihovog realnog tehničkog značenja ili dozvoljenih standarda. Matematički ta ograničenja se zadaju sistemom jednačina ili nejednačina sledećeg oblika:

$$y_i = a; \quad y_k \geq b; \quad y_l \leq c; \ldots \tag{3.2}$$

U složenijim slučajevima ograničenja pokazatelja performansi uključuju i zavisnost među njima, pa dobijamo složene sisteme jednačina ili nejednačina

$$\varphi_{y_i}(Y) = 0; \qquad \varphi_{y_j}(Y) \leq 0. \tag{3.3}$$

Slijedi da je dozvoljeni skup pokazatelja performansi Y_{dop} podskup skupa Y ($Y_{dop} \subset Y$). Neke od ovih pokazatelja performansi se mogu prestaviti matematičkim funkcijama a neke su samo opisnog karaktera.

Kao pokazatelji performansi ili efikasnosti distribuiranog računarskog sistema mogu se izdvojiti:

- *Produktivnost sistema* C_Σ, (engl. *Throughput*) ili pojedinih njegovih elemenata, se u opštem slučaju, određuje kao količina obrađenih korisničkih zahtjeva u jedinici vremena. Ako se uzme u obzir i kategorija prioriteta, onda ovu veličinu možemo definisati i kao vektor čije su komponente količine zahtjeva i-tog prioriteta koje se obrade u jedinici vremena:
$$\bar{C}_\Sigma = \{C_0, C_1, C_2, ..., C_m\} \qquad (3.4)$$
- *Vrijeme odziva* ili odgovora na postavljeni zahtjev (engl. *Response time*), recimo to može biti vrijeme izvršavanja neke transakcije ili operacije nad distribuiranom bazom podataka, ili srednje vrijeme rješavanja složenog aplikativnog zadatka;
- *Vrijeme prenosa* poruka kroz komunikacionu mrežu, pri čemu se vrlo često uvodi i kategorija prioriteta poruka $i \in \{0,1,2,...,m\}$, tada se srednje $\overline{T_z}$ i maksimalno $T_{z_{max}}$ vrijeme prenosa poruka kroz mrežu može dobiti uz pomoć sljedećih formula:
$$\overline{T_z} = \frac{\sum_{i=0}^{m} \overline{T_{z_i}}}{m}; \qquad T_{z_{max}} = \max\{T_{1_{max}}, T_{2_{max}}, ..., T_{m_{max}}\} \qquad (3.5)$$
- *Propusna sposobnost* komunikacione mreže izražava u broju poruka ili drugih jedinica podataka koji u jedinici vremena prođu komunikacionim kanalom;
- *Procenat grešaka* $G_n(t)$ u prenosu, obično se izražava kao odnos broja pogrešnih i ukupno prenesenih jedinica podataka u toku nekog posmatranog vremenskog intervala. Ovaj parametar se može okarakterisati i kao *kvalitet prenosa* poruke kroz komunikacionu mrežu.
- *Iskorišćenost* resursa sistema (engl. *Utilization*), se najčešće izražava koeficijentima iskorišćenja pojedinih sastavnih elemenata distribuiranog sistema (npr. procesora η_p, komunikacionog kanala η_i, memorije η_m itd.) i predstavlja odnosom vremena zauzetosti tog elementa prema ukupnom vremenu posmatranja;
- *Sigurnost* sistema i njegova otpornost na otkaze (engl. *Fault tolerance*);
- *Preciznost* ili tačnost dobijenog odgovora (engl. *Accuracy*);
- *Raspoloživost* sistema (engl. *Availability*);
- *Pouzdanost* odnosno srednje vrijeme rada bez kvarova (engl. *Reliability*);
- *Srednje vrijeme opravke* ili ponovnog dovođenja u pogonsko stanje;
- *Komfornost* i lakoća korišćenja;
- *Cijena* sistema ili pojedinih njegovih elemenata itd.

Različiti pokazatelji performansi distribuiranog računarskog sistema opisuju razne aspekte ponašanja sistema. Oni, uopšteno govoreći, nijesu nezavisni jedan od drugog i često su povezani međusobnim zavisnostima. Poboljšanje nekog pokazatelja performansi ne dovodi uvijek do poboljšanja i drugih pokazatelja, nego često može prouzrokovati i njihovu degradaciju. Zato se kod analize distribuiranih sistema neki pokazatelji performansi označavaju kao primarni a drugi kao sekundarni, pa se sekundarni pokazatelji izražavaju pomoću primarnih.

3.4.2.1 Produktivnosti kao pokazatelj efikasnosti distribuiranih sistema

Jedan od najvažnijih pokazatelja efikasnosti distribuiranog računarskog sistema je svakako njegova produktivnost. Određivanjem ili procjenom produktivnosti dobija se relativno kvalitetna slika o efikasnosti posmatranog sistema, tako da će ona biti predmet posebne pažnje u ovom radu.

Produktivnost određuje "računarsku snagu" sistema kroz količinu obrađenih korisničkih zahtjeva ili računarskih usluga koje obezbjeđuje sistem u jedinici vremena. Razumijevanje riječi usluga može biti veoma različito i zavisi od cilja primjene sistema. Tako na primjer usluga može biti neka transakcija nad distribuiranom bazom podataka, tada je produktivnost količina transakcija koje sistem obradi u jedinici vremena. To može biti obim informacije, izmjeren npr. u bajtima koji je sistem sposoban prenijeti za određeni vremenski interval sa svojeg ulaza na izlaz. Usluga može biti vrijeme korišćenja procesora ili perifernog uređaja od strane korisničkog programa.

Danas ne postoji opšteprihvaćena metodika ocjene performansi distribuiranih sistema pa se i sam pojam produktivnosti sistema ne određuje jednoznačno. Donekle je uzrok tome, odsustvo jedinstvene matematičke teorije i matematičkog modela, koji bi odražavali nužne aspekte dinamike informacionih procesa u distribuiranom računarskom sistemu. Tako, sve do danas nije standardizovana jedinica mjerenja računarskog rada, odnosno računarskog posla i težina obrade informacija. Nema opšteprihvaćene matematičke apstrakcije pojma ponašanja distribuiranog računarskog sistema, drugim riječima dinamička suština distribuiranog računarskog sistema još uvijek je nedovoljno objašnjena. To se posebno odnosi na aplikativni dio programske podrške distribuiranog računarskog sistema.

Produktivnost se kao jedan od oblika efikasnosti sistema može posmatrati sa dvije različite pozicije i to kao:

- *Spoljašnja* ili korisnička efikasnost sistema, drugim riječima to je efikasnosti sistema za konkretnu primjenu. Ovakva produktivnost je interesantna samo krajnjem korisniku posmatranog distribuiranog sistema i služe za kontrolu i planiranje iz takozvane korisničke perspektive.
- *Unutrašnja* efikasnost sistema, a to je pokazatelj kako sistem ekonomiše sa svojim resursima. On treba npr. da pokaže koji su dodatni rashodi pri radu sistema i da odgovore na pitanje da li se ti rashodi mogu smanjiti.

Sa razvojem mikroprocesorske tehnologije uz stalno pojeftinjenje hardverskih komponenti personalnih računara, pri jednovremenom usložnjenju i poskupljenju programske podrške (softvera), značaj unutrašnje efikasnosti distribuiranog sistema u poređenju sa spoljašnjom slabi. Sva postojeća razmatranja spoljašnje produktivnosti mogu se svesti na propusnu sposobnost sistema prema korisničkim zahtjevima (engl. *throughput*) i vrijeme odziva ili odgovora na postavljene zahtjeve (engl. *response time*).

U zavisnosti od nivoa posmatranja sistema mogu se razlikovati tri oblika produktivnosti sistema:

- ***tehnička***,
- ***kompleksna*** i
- ***sistemska***.

Svaki hardverski uređaj u sistemu karakteriše se sopstvenim performansama, i ima sopstvenu produktivnost koja ne zavise od spoljašnje sredine, a određena je samo mogućnostima tog uređaja. *Tehnička produktivnost* hardverskog uređaja odlikuje se količinom operacija, koje je sposoban da izvršiti taj uređaj u jedinici vremena. Tehnička produktivnost se obično izražava tablicama ili vektorima koje daju proizvođači ovih uređaja. Tablice sadrže listu operacija, koje je sposoban da izvrši dati uređaj kao i vrijeme njihovog izvršavanja. Skup tehničkih performansi svih hardverskih uređaja koji ulaze u sastav distribuiranog računarskog sistema, čini tehničke performanse tog sistema.

Tehničke performanse karakterišu samo potencijalne mogućnosti nekog računarskog uređaja i one u suštini ne mogu biti iskorišćene u potpunosti. To proizilazi zato, što struktura veza među uređajima u sistemu indukuje uzajamni uticaj tehničkih performansi uređaja. Npr. ukupna produktivnost autonomnih računara u jednom distribuiranom računarskom sistemu ograničena je propusnom mogućnošću komunikacionog kanala.

Sa ciljem određivanja uticaja strukture veza na tehničke performanse sistema ponekad se uvodi pojam *komplekse produktivnosti*. Ovaj oblik produktivnosti pokazuje koji dio tehničke produktivnosti svakog uređaja može biti iskorišćen pri njihovom zajedničkom radu, pri uslovima maksimalnog opterećenja, tj. izbalansiranosti tehničkih performansi svakog od uređaja ili autonomnih računara, ako se uzme ovaj nivo posmatranja.

Performanse računarskih uređaja koji rade zajedno ili računara, ako se posmatra distribuirani sistem zavise ne samo od strukture veza među uređajima, tehničkih performansi odvojenih uređaja (računara), nego i u određenom stepenu, od dinamike računarskog procesa koji pobuđuju aplikativni programi. Ako posmatramo aplikativni program, kao generator zahtjeva za izvršavanje aktivnosti ka komponentama računarskog sistema, onda je jasno da baš on određuje njihovo opterećenje odnosno punjenje ili zaposlenost. Proces povremenog opterećenja odnosno zaposlenosti komponenti distribuiranog računarskog sistema, od strane aplikativnog programa naziva se ponašanjem programa. Produktivnost se sada može definisati i kao mogući stepen opterećenosti distribuiranog sistema aplikativnim programima.

Performanse, a među njima je najvažnija produktivnost, koje pokazuje ili ispoljava distribuirani računarski sistem pri zajedničkom radu aplikativnih programa, sistemske programske podrške datog računarskog sistema i njene aparature zvaćemo sistemskim produktivnošću ili prosto performansama distribuiranog računarskog sistema. *Sistemska produktivnost* zavisi od sve tri osnovne komponente distribuiranih računarskih sistema, fizičke sredine, izvršne sredine i radnog opterećenja. Od aplikativnog programa zavisi sama suština pojave zahtijeva, poredak dolazaka zahtjeva u sistem i količinske karakteristike zahtjeva. Brzina obrade ili ispunjavanja zahtijevaju, arbitraža konflikta, da li postoji više konkurentnih zahtjeva i sl. određuje se unutar fizičke i izvršne sredine distribuiranog računarskog sistema, odnosno: tehničkom produktivnošću hardverskih elemenata sistema, njihovom izbalansiranošću, strukturama veza među njima, metodama i sredstvima distribucije resursa u sistemu itd. Na taj način, samo sistemska produktivnost, odnosno *sistemske performanse*, daju potpunu

karakteristiku distribuiranog računarskog sistema kao kompleksa programskih i hardverskih sredstava.

Sva postojeća objašnjenja pojma produktivnosti sistema povezani su sa vremenom tj. koliko određenih računarskih usluga u jedinici vremena ili za određeni vremenski interval može obezbijediti razmatrani sistem. Zato, će se ovdje produktivnost kao glavni pokazatelj performansi sistema razmatrati kao funkcija vremena. Npr. sistemska produktivnost može biti potpuno određena vremenskim dijagramom opterećenja hardverskih uređaja sistema. Na taj način zadatak dobijanja adekvatne ocjene sistemske produktivnosti se svodi na zadatak dobijanja tačnog vremenskog dijagrama opterećenja hardverskih uređaja pri izvršavanju zadatog aplikativnog programa ili skupa aplikativnih programa. U tom dijagramu za svaki uređaj posmatranog distribuiranog računarskog sistema treba pokazati koji dio vremena njegovog rada zauzimaju aktivnosti aplikativnog programa, a koji dio su aktivnosti sistema, odnosno sistemskih programa. Preko pojma aplikativnih programa odražava se korisnički pogled na sistem.

3.4.3 Parametri distribuiranog sistema

Skup parametara distribuiranog računarskog sistema $X = \{x_1, x_2, .., x_j, ..., x_m\}$ obuhvata veličine koje matematički opisuju komponente posmatranog distribuirani sistema. Ovaj skup parametara se dijeli na podskup *parametara sistema* X_s i podskup *parametara radnog opterećenja* X_r.

Parametre sistema možemo podijeliti na fizičke i sistemsko-softverske parametre sistema. Fizički parametri opisuju fizičku sredinu odnosno fizičku strukturu sistema. Ona obuhvata hardverske komponente autonomnih računara i sistem veza među njima. Tu prije svega spadaju:

- karakteristike sastavnih elemenata mikroprocesorskog sistema,
- arhitektura mikroprocesora koju definišu:
  - razrednost,
  - spisak komandi,
  - način adresacije,
  - broj registara opšte namjene,
  - broj nivoa prekida,
  - tehnologija izrade,
  - učestanost takta (engl. *clock*) mikroprocesora,
  - brzina izvršavanja operacija,
  - broj izvora napajanja i tome slično.
- operativna memorija računara koju karakterišu:
  - kapacitet,
  - vrijeme pristupa,
  - način pristupa,
  - način prenosa podataka od memorije ka interfejsima i obrnuto,
- broj i karakteristike interfejsa komunikacionih kanala,
- brzina i način prenosa podataka po ulaznim i izlaznim kanalima,
- organizacija redova čekanja itd.

Sistemsko-softverski parametri čine distribuiranu računarsku sredinu, odnosno softversko okruženje koje obezbeđuje rad sistema i ponaša se kao interfejs između hardvera i aplikativnih programa korisnika. Ovdje spadaju:

- karakteristike operativnih sistema autonomnih računara,
  - sistem upravljanja resursima,
  - sistem upravljanja memorijom,
  - dispečer procesa,
  - načini sinhronizacije i komunikacije između kooperativnih procesa,
  - sistem upravljanja datotekama,
- specijalizovani mrežni operativni sistemi,
- strategije i metode opsluživanja kanala,
- komunikacioni softver,
- algoritmi raspodjele baferne memorije,
- efikasnost funkcionisanja drajvera i tome slično.

Mnogi od pomenutih parametara sistema se teško mogu definisati analitički, pa se pribjegava drugim, obično opisnim, metodama. Od ovih metoda najznačajnija i najčešće korišćena je metoda *imitacionog modeliranja*.

Pri ma kojoj modifikaciji parametara sistema dobija se principijelno drugačiji računarski sistem. Istina, ako je modifikacija minimalna, smatramo da smo dobili *novu konfiguraciju*, ili novu verziju istog sistema.

3.4.4 Promjenljive pri modeliranju distribuiranog sistema

Pri modeliranju distribuiranih sistema razlikuju se dvije vrste promjenljivih: *egzogene* i *endogene*. Egzogene promjenljive X_{eg} se još nazivaju i ulazne promjenljive. To znači da one nastaju van sistema ili su posljedica spoljašnjih uzroka. Skup ovih promjenljivih se najčešće označava kao *radno opterećenje* ili parametri spoljašnje sredine jer svi zahtjevi za obradu podataka pristižu u računarski sistem spolja [25], [26], [27], [29]. Tu spadaju korisnički zahtjevi ili zadaci sistemu i poredak njihovog pristizanja u sistem. Ovaj niz naredbi ili zadataka korisnik obično prezentira računarskom sistemu u formi programa, koji je u poglavlju 2.9, označen kao *distribuirani aplikativni program*. Postoje i drugi oblici prezentacije korisničkih zahtjeva računarskom sistemu kao što su npr. komandni upiti ili zahtjevi za transakcijama u bazi podataka. Međutim i oni imaju sličnu prirodu pa se mogu svesti na formu programa. Kao ulazne (egzogene) promjenljive distribuiranog sistema, moraju biti uzete i one koje zahtijevaju funkcije tog sistema, odnosno to su promjenljive koje odgovaraju namjeni za koju se sistem koristi.

Korisnički zahtjevi sadržani u izvornom aplikativnom programu se spolja, uz pomoć nekog perifernog uređaja (tastature, diskete, diska i slično) unose u sistem, pa se i označavaju kao dio spoljašnje sredine distribuiranog sistema. Spoljašnja sredina ili radno opterećenje ima različite uticaje na posmatrani distribuirani računarski sistem. Ove uticaje karakteriše između ostalog slučajnost nastanka i nepredvidljivost toka čime se u znatnoj mjeri otežava njihov opis i analiza. Koji su to parametri koji definišu aplikativni program i kako ih opisati? Šta karakteriše aplikativne programe u distribuiranoj računarskoj sredini? Kako aplikativni programi djeluju na sistem i šta karakteriše njihov uticaj? Samo su neka od pitanja na koja će se pokušati odgovoriti u ovom radu.

Grupe zahtjeva, unesene u distribuirani sistem u formi aplikativnih programa ili na drugi način, transformišu se u dinamičke forme poznate pod nazivom procesi. Određivanje karakteristika aplikativnih procesa, modela njihovih ulaznih tokova i modela razmjene poruka između tih procesa i procesa operativnog sistema jedan je od osnovnih zadataka analize distribuiranih sistema.

Promjenljive označene kao endogene X_{en} su one koje nastaju u samom sistemu ili su posljedica unutrašnjih aktivnosti distribuiranog sistema. One se nazivaju promjenljive stanja, kada karakterišu stanje ili uslove koji su stvoreni u sistemu, a kada se govori o rezultatima sistema onda su označene kao izlazne promjenljive. Endogene promjenljive se još označavaju "zavisnim" za razliku od egzogenih promjenljivih koje su "nezavisno" promjenljive.

3.4.5 Neformalna analiza zadatka i definisanje matematičkog modela

Pri definisanju zadatka analize pretpostavljeno je da ne postoji distribuirani računarski sistem S ili njegov prototip. Tada je za procjenu njegovih performansi Y potreban model F sistema S, koji bi postavio odnos $Y = F(X)$. U opštem slučaju skup X predstavlja parametre i ulazne promjenljive distribuiranog sistema, odnosno njegovu hardversko-softversku strukturu i njegovo radno opterećenje. Dok su fizička struktura i ponašanje sistemskih programa manje-više konstantni tokom jedne serije ispitivanja, radno opterećenje odnosno ponašanja aplikativnih programa se mijenja. Zato se pri definisanju modela F sa dovoljnom tačnošću može uzeti da X, matematičkim sredstvima, opisuje ponašanja distribuiranih aplikativnih programa iz skupa P. Ovdje se odmah nameće sljedeće pitanje: šta se podrazumijeva pod pojmom ponašanja aplikativnog programa u distribuiranoj računarskoj sredini? U sljedećem poglavlju pokušaće se odgovoriti na ovo pitanje.

Dakle, produktivnost sistema ili njegovi pokazatelji performansi se uz pomoć modela, mogu matematički predstaviti u zavisnosti od ponašanja distribuiranih aplikativnih programa, i to na sljedeći način:

$$\left.\begin{aligned} y_1 &= f_1(x_1, x_2, \cdots, x_n) \\ y_2 &= f_2(x_1, x_2, \cdots, x_n) \\ &\vdots \\ y_m &= f_m(x_1, x_2, \cdots, x_n) \end{aligned}\right\} \Leftrightarrow Y = F(X) \qquad (3.6)$$

Sistem gornjih jednačina predstavlja distribuirani računarski sistem u formi *matematičkog modela*. Izgradnja numeričkog modela je cjelishodna radi lakšeg i bržeg proučavanja i dobijanja potrebnih informacije o radu posmatranog sistema.

Matematički model opisuje funkcionisanje sistema posredstvom matematičkih operatora i objekata. Na taj način, analiza realnog distribuiranog sistema zamjenjuje se ispitivanjem njegovog matematičkog ili numeričkog modela. Drugim riječima, ispitivanje sistema se svodi na rješavanje konačnog broja logičkih i računskih operacija nad brojevima uz pomoć odgovarajućih numeričkih metoda. Ako se ove operacije, pomoću odgovarajućih algoritama, realizuju na računaru onda se radi o ***imitacionom modeliranju***. Metod imitacionog modeliranja predstavlja numerički eksperiment na računaru sa matematičkim modelom sistema kao objektom ispitivanja.

3.5 Problemi izgradnje opisa ponašanja programa u distribuiranoj sredini

Kao što je već naglašeno opis ponašanja distribuiranih aplikativnih programa igra značajnu ulogu u izgradnji modela njegovog radnog opterećenja i rješavanju problema analize distribuiranih računarskih sistema. Zato će se ovdje ovoj problematici posvetiti posebna pažnja.

Program ima dvostruku prirodu. S jedne strane to je statički objekat, predstavljen tekstom programa, odnosno nizom zahtjeva za uslugama koje korisnik traži od distribuiranog računarskog sistema. Sa druge strane kada je program uveden u sistem, to je proces, tj. dinamički objekat. Malo je što poznato o tome kakva svojstva prvog određuju svojstva drugog i u kom stepenu i kakva su svojstva drugog. Za prvu formu (kao teksta) postoji u literaturi bogat skup algoritama za analizu različitih svojstava i načina opisa. Za drugu formu (kao procesa) analogna sredstva još nijesu dovoljno razrađena, pogotovu ne za ponašanje programa u distribuiranoj računarskoj sredini.

Program-tekst konačno određuje program-proces. Ovdje se odmah nameće pitanje da li je ovo preslikavanje uvijek jednoznačno i drugo, da li izvođenje te dinamičke forme iz teksta može biti jednako izvršenju programa. Tako npr. tekst paralelnih programa, pripremljenih za rad u distribuiranoj sredini ne daje potpuno i jednoznačno određivanje programa u formi procesa. Uzrok ovoga se nalazi u činjenici da se izvršavanje programa u distribuiranoj sredini odvija paralelno i u saradnji sa drugim sličnim programima i sa programima operativnog sistema. U tekstu programa mogu postojati npr. *tačke nedeterminizma* odnosno neodređenosti, čija konkretizacija može zavisiti od parametara distribuiranog računarskog sistema tj. čija se suština ne nalazi unutar programa već van njega. Znači da pretvaranje distribuiranih programa iz prve forme u drugu, koristeći samo tekst, nije moguća. Samim tim, metode izgradnje opisa ponašanja aplikativnog programa u distribuiranoj sredini takođe postaju predmet istraživanja i sastavni su dio analize efikasnosti sistema. Poželjno je da se tokom analize, to pretvaranje programa iz forme teksta u formu procesa, tj. opis dinamike programa, sprovodi što je moguće ređe. Drugim riječima, treba težiti da se jedan isti opis dinamike distribuiranih aplikativnih programa može koristiti tokom više seansi istraživanja. U tom cilju je potrebno izgraditi sredstvo za opis dinamike programa, tj. potrebno je znati jednoznačno opisati distribuirani program kao skup kooperativnih procesa. Za izgradnju takvog sredstva potrebno definisati model dinamike programa, koji bi uzimao u obzir specifičnosti aplikativnih programa u posmatranoj distribuiranoj sredini, npr. potrebno je da se eksplicitno pokažu sve tačke u programu koje mogu izazvati neodređenost prilikom njegovog preslikavanja iz forme teksta u formu procesa. Ove tačke su označene kao tačke nedeterminizma.

Model dinamičkog ponašanja aplikativnih programa u posmatranoj distribuiranoj sredini do sada nije adekvatno opisan. Modeli iz literaturnih izvora: [31], [62], [63], [70], [71], [72], [95], [97] nijesu pogodni za istraživanje ove klase distribuiranih sistema, zato što su oni prije svega izgrađeni za analizu algoritamskih svojstava programa [77]. Oni ili uopšte ne sadrže vrijeme kao količinsku stvarnost, ili je kod njih program i fizička sredina njegovog izvršavanja jedna cjelina. Problematika uzajamne veze programa i vremena do danas je istraživana slabo. Navedeni modeli koriste za opis

paralelizma samo semantiku redanja [102]. Pojam paralelizma, kod distribuiranih računarskih sistema, u literaturi se obično objašnjava na jedan od sljedeća dva načina: kao forma nezavisnosti procesa po informaciji i upravljanju ili kao aktivnosti koje se odvijaju u jedno isto vrijeme. Ako model ne sadrži takvu kategoriju kao što je vrijeme, njime se ne može obuhvatiti drugi oblik paralelizma. Uvodeći u model oba oblika paralelizma, rješava se problem njihove uzajamne veze. To je još jedan krug problema koji se mora proučavati.

Sa druge strane model Smeljanskog [80], kod koga su ovi problemi dobrim dijelom riješeni, je primijenjen na specijalizovan multiprocesorski računarskih sistema sa zajedničkom memorijom. Ovaj sistem je sličan čvrsto spregnutim distribuiranim sistemima, jer svaki procesor u sistemu ima svoju operativnu memoriju, zbog čega ga autor, mada nepravilno, i naziva distribuiranim. Međutim, sva memorija u sistemu je opštedostupna, tj. svaki procesor, pomoću odgovarajućih sredstava, može pročitati iz ili zapisati u operativnu memoriju ma kog drugog procesora, tako da ovaj sistem pripada multiprocesorskim računarskim sistemima. Problemi analize ovog multiprocesorskog sistema, mada slični problemima distribuiranih sistema, su zbog kompaktnosti, kratkih veza između procesora i postojanja jedinstvenog operativnog sistema manje izraženi.

Prilikom izgradnje adekvatnog modela radnog opterećenja posmatranog distribuiranog sistema, odnosno, opisa ponašanja aplikativnog programa treba težiti da on ne zavisi od računarskog sistema na kom se program izvršava. U suprotnom slučaju, pri izmijeni bilo kojih karakteristika sistema, npr. u toku analize, mora se ponovo izgrađivati taj opis. Istovremeno, postojanje takvog opisa je dosta problematično i njegovo traženje je jedan od ciljeva ovog rada. Taj opis treba da karakteriše ponašanje programa kako kvalitativno tako i količinski.

Nezavisnost opisa ponašanja programa od računarske sredine može biti dostignuta rješavanjem sljedećih problema:

- *Izbor potrebnog nivoa detaljizacije* opisa ponašanja programa. Što podrobnije mi opisujemo ponašanje programa npr. na nivou komandi konkretne mašine, time je opis više sistemski zavistan. Sa druge strane, što je viši njegov nivo, tim je manje u njemu informacija o opterećenju komponenti fizičke strukture računarskog sistema ali time se smanjuje tačnost modeliranja;
- *Čuvanjem u opisu ponašanja programa svih njenih unutrašnjih uzročno-posljedičnih veza*, određenih u programu uzajamnom zavisnošću informacionih veza i upravljanja [100], [101]. Za distribuirane programe to je izuzetno važno jer te uzajamne veze mogu uticati na ponašanje svih procesa, koji obrazuje sistem distribuiranih aplikativnih programa;
- *Izgradnja sistemski nezavisne mjere računarskog rada*, koji obavlja fizička struktura sa odgovarajućom sistemsko-softverskom podrškom distribuiranog sistema pod uticajem aplikativnog programa.

Zadavajući ponašanje aplikativnog programa u obliku vremenskog dijagrama opterećenja komponenti fizičke strukture distribuiranog računarskog sistema može se dobiti vrlo tačna procjena produktivnosti sistema. Isto tako zadato ponašanje programa čvrsto je povezano za konkretnu distribuiranu računarsku sredinu. Izmijena tehničkih performansi jednog od elemenata hardverske strukture sistema, strukture veza između elemenata ili elemenata sistemsko-softverske strukture izmijenit će i ponašanje

aplikativnog programa izraženog u terminima vremenskih intervala. Odavde slijedi da ako se želi dobijanje efektivnog metoda analize, onda opis ponašanja aplikativnog programa ne treba da zavisi od takvih karakteristika sistema kao što je struktura veza, tehničke performanse i sl. Postojanje takve sistemski nezavisne mjere je takođe problematično, ako se tako može reći, jer je ponašanje programa kategorija koja, u opštem slučaju, nije nezavisna u odnosu na fizičku i sistemsko-softversku strukturu računarskog sistema.

Rješivši ove probleme, dobija se kvalitetan siguran i adekvatan opis ponašanja aplikativnog programa u distribuiranoj računarskog sredini. Pomoću njega, može se izgraditi model sistema S, npr. imitacioni, i izračunati vremenski dijagram opterećenja hardverskih elemenata fizičke strukture. Po tom dijagramu, zatim, se može procijeniti produktivnost i druge količinske karakteristike funkcionisanja distribuiranog sistema. Poželjno je da nivo detaljizacije opisa ponašanja aplikativnog programa i modela može varirati. To dozvoljava da se sistem S istražuje sa raznim stepenima detaljizacije.

3.6 Postupci analize performansi distribuiranog sistema

Postupci analize i procjene (engl. *evaluation*) performansi obično se sastoje iz niza koraka ili etapa. Prvim se obično javlja izbor mjere ili mjera performansi, tj. tih parametara, na osnovu kojih će se vršiti procjena. Tu prije svega spada mjera produktivnosti posmatranog sistema. Poslije toga se određuje zavisnost performansi od strukture *analizirajućeg sistema* i njegovog radnog opterećenja.

Konkretizacija ove šeme zavisi od mnogih faktora a prije svega od toga za kakve se ciljeve procjenjuju performanse. Svi dosadašnji radovi analize i procjene performansi sa tačke gledišta njihovih ciljeva mogu se usmjeriti u tri pravca [29], [30], [31], [32], [34], [9]:

- *Uporedna analiza* postojećih sistema. Ona se obično sprovodi u cilju klasifikacije ili izbora računarskog sistema iz već postojećih;
- *Optimizacija*, tj. regulacija i kontrola performansi. U datom slučaju performanse se ocjenjuju u odnosu na različite parametre sistema i njegovog radnog opterećenja. To se radi sa ciljem regulacije i prilagođavanja npr. operativnog sistema na zadatu konfiguraciju sistema, ili sa ciljem procjene različitih pravaca razvitka sistema. Ovdje je važno napomenuti, da razlike između računarskog sistema čije se performanse procjenjuju i stvarnog realnog sistema, ne nose principijelni karakter a odnose se samo na količinske vrijednosti ovih ili onih karakteristika;
- *Prognoziranje performansi* nekog računarskog sistema koji ne postoji i tek treba da bude projektovan. Ovakav prilaz izaziva ozbiljne teškoće pri izradi modela radnog opterećenja odnosno ponašanja programa u jednoj takvoj sredini i provjeru modela sistema.

Kako je već naglašeno u formulaciji zadatka u poglavlju 3.2, u ovom radu je aktuelan prilaz prognoziranja performansi analizirajućeg sistema. On omogućava da se, znajući ponašanje programa u nekoj, "eksperimentalnoj" distribuiranoj računarskoj sredini (u *kontrolnom distribuiranom sistemu)*, može prognozirati njegovo ponašanje u

drugoj "analizirajućoj" sredini. Prognoza u datom slučaju označava ne samo predviđanje redoslijeda aktivnosti programa u novoj sredini, već i procjenu vremena izvršavanja tih aktivnosti. Ključni problem ostvarljivosti tog prilaza javlja se obrazloženje hipoteze o postojanju sistemski nezavisne forme opisa ponašanja distribuiranog programa. Tj. definisanje kada i pod kojim uslovima se može uzeti da je ponašanja aplikativnog programa u distribuiranoj sredini nezavisno od te sredine. Tek poslije rješavanja ovih problema, može se riješiti polazni zadatak analize i procjene performansi distribuiranog sistema kroz ***sljedeći niz koraka:***

1. ***Definisati i izgraditi sistemski nezavistan opis ponašanja aplikativnog programa***, koji mora karakterisati kako njegovo kvalitativno tako i količinsko ponašanje. Takav opis se može dobiti na osnovu analize rada aplikativnog programa u nekoj specijalizovanoj računarskoj sredini. Odmah treba primijetiti, da izgradnja takvog opisa samo na osnovu analize teksta programa nije moguća, već zbog toga što tekst sadrži tačke nedeterminizma;
2. Na osnovu logičke strukture distribuiranog računarskog sistema tj. po njegovom funkcionalnom opisu ***izgraditi model, neophodnog nivoa detaljizacije***. Osnovu takvog opisa treba da čini matematički model funkcionisanja distribuiranog sistema. Taj model mora da uključi algoritamske, strukturne i količinske karakteristike fizičke i sistemsko-softverske strukture analiziranog distribuiranog sistema;
3. Po opisu ponašanja aplikativnog programa ***izgraditi model radnog opterećenja*** za model distribuiranog računarskog sistema;
4. Pomoću modela sistema i modela radnog opterećenja ***izgraditi vremenski dijagram opterećenja hardverskih elemenata*** (procesora, kanala veza, memorijskih modula itd.) toga sistema radi njegove analize;
5. Po dobijenim vremenskim dijagramima ***izračunati potrebne pokazatelje performansi distribuiranog računarskog sistema***.

Ovakav prilaz dozvoljava analizu ponašanja istog programa u različitim računarskim sredinama. Garanciju sigurnosti i tačnosti rezultata modeliranja predstavlja stepen do kog se došlo u očuvanju opisa svih unutrašnjih relacija ili uzajamnih veza, svojstvenih datom programu. Naravno pretpostavlja se da je izgrađeni imitacioni model sistema korektan.

Ovdje predstavljen prilaz nije karakterističan za neku usku klasu distribuiranih računarskih sistema. Njena ograničenja su uslovljena matematičkim modelom funkcionisanja paralelnih računarskog sistema i već pomenutim karakteristikama razmatranih distribuiranih računarskih sistema i programa. Kao što se vidi iz posljednjeg pregleda osnovnih prilaza ka procjeni performansi računarskih sistema, složenost toga prilaza leži između složenosti analitičkog modeliranja i složenosti detaljne emulacije rada komandi autonomnih računara koji ulaze u sastav razmatranog distribuiranog računarskog sistema. U isto vrijeme njegova tačnost zauzima srednji položaj između metoda prototipa i prilaza orjentisanog na trase.

GLAVA

4. METODE I SREDSTVA ANALIZE PERFORMANSI TRADICIONALNIH RAČUNARSKIH SISTEMA

4.1 Uvod

U cilju izbora najbolje metodologije za analizu distribuiranog računarskog sistema posmatrane klase, ovdje je dat pregled osnovnih metoda analiza i ocjena performansi računarskih sistema sa tradicionalnom arhitekturom ili računarskih sistema klase SISD (engl. *Single Instruction Stream Single Data Stream.*) po klasifikaciji Flina (M.J. Flynn, 1972) [16], [23]. Za distribuirane računarske sisteme, koji po klasifikaciji Flina pripadaju klasi MIMD (engl. Multiple Instruction *Stream* Multiple *Data Stream.*), ovakva metodologija se još uvijek izgrađuje. Ovim pregledom se pokušava dati odgovor na sljedeća pitanja:

- Kakvi novi problemi se javljaju pri procjeni performansi distribuiranih računarskih sistema u poređenju sa sistemima klase SISD ?
- Da li su za analizu i procjenu ponašanja distribuiranih sistema potrebni novi instrumenti ili u potpunosti odgovaraju stari ?

4.2 Modeli radnog opterećenja

Jedno od najvažnijih pitanja modeliranja računarskih sistema je definisanje njegovog radnog opterećenja. Pod radnim opterećenjem računarskog sistema podrazumijeva se skup svih ulaznih uticaja koji spolja dolaze u sistem. Praktično, to je niz ili *tok zahtjeva* za obradom ili uslugom, koji iniciran od strane aplikativnog programa ili direktno od korisnika dolazi u sistem. Računarski sistem svojim funkcionisanjem, po pravilu ne utiče, ili ne bi trebao da utiče na karakteristike ulaznog toka zahtjeva. Od pravilnog određivanja ovog toka zavisi tačnost rezultata analize.

Produktivnost računarskog sistema je u određenom smislu reakcija sistema na konkretno radno opterećenje. Zato, od toga koliko korišćeni model radnog opterećenja adekvatno odražava karakteristike konkretnog aplikativnog procesa zavisi korektnost kasnije dobijenih ocjena. Drugim riječima, bez dobrog poznavanja radnog opterećenja, analiza i prognoza produktivnosti u najboljem slučaju može biti nedovoljno tačna [29], [51], [80].

Prilikom izgradnje modela radnog opterećenja i modela distribuiranog sistema posebnu pažnju treba posvetiti izboru odgovarajućeg stepena detaljizacije modela. Od optimalnog nivoa detaljizacije s jedne strane zavisi tačnost modela a sa druge strane preteran nivo detaljizacije utiče na njegovu sistemsku zavisnost. Pri izboru nivoa detaljizacije modela radnog opterećenja i modela računarskog sistema mora se voditi računa o njihovoj uzajamnoj zavisnosti. Može se izdvojiti nekoliko dosada korišćenih nivoa posmatranja i analize računarskih sistema [34]:

- nivo šema i registarskih prenosa,
- nivo uzajamnog djelovanja procesor-memorija,
- nivo sistema komandi, ovdje već počinje programski nivo,
- nivo sistemskih poziva i prekida,

- nivo programskih procesa,
- nivo poslova ili programa,
- nivo zadataka ili kompleksa poslova.

Svi osnovni modeli radnog opterećenja koji se danas koriste pri analizi računarskih sistema mogu se svrstati u sljedeće kategorije:

- mješavine komandi [35], [36];
- etaloni [37], [38], [39], [40], [41], [42], [43];
- *stohastički modeli* [29], [30], [44], [45], [51], [52], [122], [123];
- *trase* [46], [47], [48], [49], [50].

Po jednoj drugoj klasifikaciji modeli radnog opterećenja se mogu podijele na *determinističke* i *stohastičke*. Kod determinističkog modela svi ulazni parametri modela su unaprijed određeni i definisani. U determinističke modela radnog opterećenja se mogu ubrojati trase koje sadrže sekvencijalno uređen niz potpuno definisanih, ulaznih zahtjeva i etaloni kod kojih se radno opterećenje računarskog sistema opisuje u formi programa. Ako je bar jedan ulazni parametar analize slučajna veličina, model je stohastički. Samo primjenom stohastičkog modela može biti bitno smanjena količina informacija, koja je neophodna za adekvatno predstavljanje mnogih parametara radnog opterećenja računarskog sistema.

4.2.1 Mješavine komandi

Ovaj prilaz modeliranju radnog opterećenja obično se primjenjuje pri izboru ili projektovanju procesora. Njegova suština se sastoji u izboru, sa stanovišta neke aplikacije, niza komandi tipičnih za posmatrani računar. Ova tipičnost se određuje uz pomoć funkciju statističke raspodjele pojavljivanja komandi u programu za određenu aplikaciju. Ova se funkcija izgrađuje metodama matematičke statistike, npr. pomoću klasterske ili regresione analize.

Širok spektar formata komandi, primijenjeni načini adresacije, uticaj kompajlera i sistemskog softvera čine taj metod teško primjenljivim čak i za sekvencijalne procesore sa tradicionalnom arhitekturom. Za druge tipove procesora, npr. vektorske ili superskalarne kakvi su pentijum procesori, gdje je bitna ne samo relativna frekvencija korišćenja komandi nego i redosljed njihovog pojavljivanja, on nije uopšte primjenljiv. Kod procesora pomenutog tipa postoji mogućnost da se ne obuhvati komanda koja može bitno uticati na produktivnost. Taj nedostatak javlja se kao posljedica toga što dati prilaz za modeliranje radnog opterećenja ne obuhvata unutrašnje međusobne veze za upravljanje u programima, koji proizilaze iz informacionih veza između promjenljivih programa. Ovdje se pretpostavlja da pojava pojedinih komandi statistički nezavisna, što prirodno ne odgovara istini [35], [36]. U današnje vrijeme ovaj prilaz izgradnje radnog opterećenja se rijetko upotrebljava.

4.2.2 Etaloni (engl. benchmarks)

Etaloni su obrasci korišćenja resursa sistema aplikativnim procesom. Drugim riječima to su etaloni radnog opterećenja za određeni računarski sistem ili njegov prototip. Kao etaloni obično se javljaju odvojeni programi ili skupovi programa.

Etaloni mogu biti izgrađeni iz već postojećih programa. Načini izgradnje etalona mogu biti veoma različiti: od slučajnog izbora programa na ulazu sistema do detaljnog

izbora svakog operatora u etalonu. Npr. etaloni na nivou ulaznog toka poslova izgrađuju se na sljedeći način. Cijeli tok poslova, koji prolaze kroz sistem, dijeli se na klase. Određuje se težina svake klase u toku. Za svaku klase pomoću klasterske analize izgrađuje se jedan ili više etalona, iz kojih se sa ranije određenom težinom dobijaju etaloni ulaznog toka poslova. Međutim veoma je teško korišćenjem jednog ili nekoliko etalona obuhvatiti čitavu klasu poslova.

Ovaj prilaz se uglavnom koristi za upoređivanje postojećih sistema. Njegova primjena u slučaju odsustva analizirajućeg sistema po pravilu zasnovana je na primjeni emulatora na nivou sistema komandi i karakteriše se niskom efikasnošću i velikim dodatnim troškovima eksperimentisanja, kao i opasnošću da dođe do izobličenja dobijenih podataka o radu sistema komponentama mjernog sistema, ili tom sredinom u kojoj ona radi.

4.2.3 Stohastički modeli radnog opterećenja

Kod ovog prilaza radno opterećenje se opisuje uz pomoć slučajnih veličina, koje predstavljaju zahtjeve za resursima. Raspodjele ovih slučajnih veličina biraju se na osnovu statističke analize rezultata mjerenja toka zahtjeva za odgovarajućim resursima postrojenja koje realno radi. Po pravilu dobijeni rezultati se aproksimiraju eksponencijalnom raspodjelom, čija su matematičke osobine poznate.. Takav model radnog opterećenja nije tačan i tačnost procjene performansi, dobijenih pomoću njih nije veća od 30-40%. Nekoliko je uzroka ovome.

- Najprije podaci dobijeni na postojećem postrojenju mogu imati na sebi pečat tog sistema na kojem su prikupljeni. Taj uticaj koji se naziva sistemskom zavisnošću, može biti sakriven na raznim mjestima i u različitim formama. Konkretno to zavisi od nivoa detaljizacije opisa radnog opterećenja i osobenosti organizacije sistema.
- Drugo, mjerni sistem, korišćen za sakupljanje podataka takođe može izobličavati dobijene podatke toliko koliko ona sama koristi resurse sistema. Bez detaljnog znanja kako sistem radi sistem i prethodne analize procjena tog uticaja je praktički nemoguća.
- Treće, pomoću takvog matematičkog aparata teško je obuhvatiti međusobne veze između različitih karakteristika tokova zahtjeva, predstavljenih raznim slučajnim veličinama. Zato pri korišćenju tog matematičkog aparata direktno ili ne koristi se hipoteza o statističkoj nezavisnosti tih veličina, što uopšteno govoreći, nije korektno. Pri opisu radnog opterećenja čak na nivou koraka odvojenih zadataka ova hipoteza može nositi izobličenja dobijene procjene.

Ovo je primjenljivo kod univerzalnih računarskih sistema kod kojih je teško predskazati kakvo će biti radno opterećenje a takođe kod sistema čiji je značaj mali. Što je sistem uži to je veći zahtjev za tačnošću radnog opterećenja. Statističke metode opisa, koje se nalaze u osnovi tog modela radnog opterećenja nijesu tačne po svojoj prirodi i daju samo prosječnu sliku pojave [51], [52], [115], [116], [117], [122], [123].

Na osnovu rečenog, korišćenjem ovog prilaza za projektovanje računarskih sistema veoma je ograničeno. Ograničeno je ono još i zato, što za ocjenu različitih varijanti sistema može biti potrebna detaljnija informacija, u poređenju sa već dobijenom. Da bi se dobila ova detaljnija analiza potrebno je ponavljanje cijelog tehnološkog ciklusa, skupljanja statističke obrade i primjene tih podataka. Nemoguće je

skupiti sve podatke unaprijed, zbog mogućeg unošenja izobličenja u mjerni sistem. Pa se takva mjerenja, mogu pokazati problematičnim sa stanovišta tačnosti mjernog sistema. I drugo, nije ni moguće unaprijed sve predvidjeti, pogotovu u tako složenom poslu kakav je analiza i projektovanje distribuiranih računarskih sistema.

4.2.4 Trase

Kod ovog prilaza radno opterećenje se predstavlja u obliku skupa uređenih slogova koji sadrže podatke o radu programa. Ovi podaci se skupljaju uz pomoć specijalnih mjernih sredstava, a zatim poslije specijalne obrade, se koriste kao model radnog opterećenja. Glavnom prednošću ovog prilaza javlja se to što on čuva sve uzajamne veze između aktivnosti kako odvojenih programa tako i kompleksa programa. Ovaj prilaz se primjenjuje isključivo za rješavanje zadataka optimizacije sistema, tj. kada objekat modeliranja stvarno postoji. U tom slučaju važnim dostignućem tog prilaza javlja se jednostavnost provjere modela sistema. Za provjeru modela dovoljno je uporediti rezultate modeliranja i rezultate mjerenja pri istom radnom opterećenju [29], [30], [46], [49].

Kao slabu stranu ovog prilaza, izgradnje modela radnog opterećenja, može se izdvojiti sljedeće:

- Kao i u prethodnom prilazu, pri mjerenju uvijek postoji opasnost, da mjerni instrument unose smetnje u rad sistema, koji zatim dovode do izobličenja izmjerenih podataka.
- Mjerenja, kao i u slučaju stohastičkih modela radnog opterećenja, uvijek nose sistemsko-zavistan karakter kako na dijelu nivoa na kom se izvodi mjerenje, tako i u dijelu rezultata mjerenja (tj. formata podataka, jedinica mjerenja i sl.), algoritama, korišćenih u sistemu. Zato je sve što je rečeno kod u opisu stohastičkog modela radnog opterećenja o sistemskoj zavisnosti važi i ovdje.
- Izgradnja programa za obradu trase i njenu pripremu za korišćenje u modelu sistema dosta je teško. Dobijeni program je usko usmjeren po svom ulazu i izlazu na sistem skupljanja trase i model posmatranog sistema. Osnovni cilj tog programa je da odstrani odvojene aspekte sistemske zavisnosti trasa, npr. ako se želi optimizirati algoritam planiranja zadataka u sistemu.

Ovaj metod izgradnje modela radnog opterećenja primjenjuje se samo kod programa koji ne rade u distribuiranoj računarskoj sredini [31], [32], [33]. Ovo je veoma važno zato što ako je program distribuiran, onda sa jednim istim ulaznim podacima istorija njegovog odvijanja se mijenja od propuštanja do propuštanja tj. od seanse do seanse, i podaci sakupljeni tokom jednog propuštanja, uopšteno govoreći, mogu biti beskorisni za korišćenje u drugom propuštanju. U svim poznatim primjenama ovog prilaza pretpostavlja se da arhitektura sistema koji se analizira i sistema, u kom se sakuplja trasa nemaju principijelnih razlika (npr. u sistemu komandi, topologiji veza i sl.).

Pored navedenih metoda izgradnje modela radnog opterećenja ponekad se koristi metod *često korišćenih dijelova programa* [29], [30], [51], [52] i vještačkih etalona. Međutim ovaj metod se koristi rede i nema nekih izrazitih dostignuća u poređenju sa pomenutim metodama.

4.3 Modeli sistema

Najčešće, analiza sistema se svodi na analizu njegovog modela. Za analizu produktivnosti koriste se tri oblika modela sistema:

- strukturni modeli,
- funkcionalni modeli i
- modeli produktivnosti, koji se dijele na analitičke i empirijske.

Strukturni modeli karakterišu sastavne dijelove sistema, uzajamnih relacija i interfejsa između njih. Funkcionalni modeli opisuju ponašanje sistema u formi pogodnoj za istraživanje matematičkim metodama. Modeli produktivnosti određuju zavisnost produktivnosti od takvih parametara sistema kakvi su radno opterećenje, fizička struktura sa njenom programskom podrškom i sl. Ovaj način modeliranja, dobija se kao rezultat zajedničke analize funkcionalnog modela i modela radnog opterećenja.

Kad se govori o modeliranju sistema nikako se ne smije zanemariti činjenica, da se modeliranjem ipak samo procjenjuju performanse sistema, a osnovu za tačnu analizu ipak obezbjeđuju rezultati mjerenja tih performansi.

4.3.1 Strukturni modeli računarskih sistema

Po pravilu ovi modeli se izgrađuju pomoću različitog reda strukturnih dijagrama, u suštini koristeći teoriju grafova [55], [56], [95]. Ovi modeli opisuju topologiju informacionih i upravljačkih tokova u sistemu, ali ne određuju njihovo međusobno dejstvo jednog na drugog. Za opis tih modela bili su razrađeni specijalni jezici [57], [583]. Modeli tog tipa predstavljaju prvi korak u izgradnji funkcionalnih modela.

4.3.2 Funkcionalni modeli računarskih sistema

Funkcionalne modele, korišćene pri analizi produktivnosti, moguće je podijeliti u četiri grupe, u odnosu na matematički aparat, koji je korišćen u osnovi ovih metoda:

- modeli izgrađeni uz pomoć grafova.
- modeli na osnovu konačnih automata.
- mreže.
- modeli sa redovima čekanja.

Teorija grafova je široko korišćena pri modeliranju kako programskih, tako i hardverskih komponenti računarskih sistema. Na primjer, ona se često koristi za ocjenu vremena izvršavanja programa različitog nivoa [59]. U tim modelima čvorovi predstavljaju pojedine programe, module ili operatore, a lukovi – prenos upravljanja među njima. Ili obratno, lukovi odgovaraju fragmentima programa, a čvorovi su ili tačke grananja ili predaje upravljanja.

Ako je poznato vrijeme izvršavanja fragmenta programa i vjerovatni prelaz u tačkama grananja, onda se uz pomoć ovakvih modela mogu dobiti mnoge korisne informacije o radu programa u cjelini i odvojenih njegovih dijelova. Na osnovu toga mogu se ocijeniti karakteristike korišćenja resursa sistema [60]. Međutim, izračunavanje sa dovoljnim stepenom tačnosti, vjerovatnih prelaza u grafovima, koji predstavljaju operativnu šemu programa je izuzetno teško [51]. Jedan od uzroka leži u tome što se u tim modelima po pravilu koristi pojam globalnog stanja sistema. Funkcionisanje sistema

se posmatra kao niz prelaza iz jednog stanja u drugo. Čak i kod manjih programa koji sadrže tek nekoliko desetina operatora broj mogućih stanja dostiže takve razmjere, da rad sa njima zahtijeva značajne računarske gubitke.

U modelima izgrađenim na osnovama *teorije automata* [61] stanje sistema se izgrađuje od lokalnih stanja odvojenih njegovih komponenti, što dozvoljava da se dovoljno prosto uzme u obzir paralelno funkcionisanje odvojenih dijelova sistema. Isto tako, i u ovim modelima važnu ulogu igra poznavanje vjerovatnih prelaza između lokalnih stanja. Imajući u vidu vjerovatnoću prelaza u grafu, gdje čvorovi predstavljaju određena lokalna stanja. Njihovo izračunavanje je takođe izuzetno teško.

Mreže, korišćene za izgradnju funkcionalnih modela, javile su se kao rezultat pokušaja opisa dinamike funkcioniranja paralelnih sistema. Najpopularnije od njih su mreže Petri [95], [96] i njena uopštenja. Kod ovog modela najveći problemi se ogledaju u trima osnovnim karakteristikama:

- Svaki korak u razvitku sistema određuje se ne na opštem ili globalnom stanju sistema, već samo na stanju tih njenih komponenti od kojih je on zavisan i gdje su nastale kakve-takve izmijene. Nezavisne aktivnosti ostaju nezavisne i pojavila se mogućnost opisa paralelizma.
- Modelira se samo upravljanje. Model se apstrahuje od tih izmijena, koje ili ne utiču na tok upravljanja, ili su toliko složeni za opis, da je prostije bilo modelirati te izmijene za račun uvođenja nedeterminizma.
- Model je grafički tj. sve moguće osobine se zadaju bilo grafovima bilo oznakama na grafovima. Ovi modeli se koriste samo za analizu algoritamskih svojstava sistema [77], a ne i njihovih količinskih karakteristika.

Modeli sa redovima čekanja [28], [81], [82] zasnivaju se na teoriji masovnog opsluživanja [115]. U tim modelima računarski sistem se predstavlja kao skup resursa – servera i redova čekanja na njihovu uslugu. Kad zadatak ili zahtjev dođe u sistem on staje u red čekanja za odgovarajući resurs. Dobivši pristup k njemu, poslije opsluživanja, zadatak staje u red za drugi resurs itd. Na taj način modeli sa redovima čekanja opisuju tokove zadataka i njihovo zadovoljenje u sistemu. Ovi su modeli dobro poznati i masovno se koriste. Međutim, oni dozvoljavaju da se izvrši samo količinska analiza funkcioniranja sistema. Za algoritamsku analizu oni ne odgovaraju.

Važnim pitanjem izgradnje funkcionalnih modela javlja se kako je kod njih uvedeno vrijeme. Postoje dva principijelno različita rješenja ovog problema. U prvom se vrijeme uvodi kao samostalna, nezavisna i neprekidna suština. Vrijeme se odvija nezavisno od sistema i nikako nije povezano sa procesima koji se u njemu odvijaju. Kod drugog rješenja prelazak od jedne tačke na vremenskoj osi do druge dešava se samo tada, kada se u sistemu desio neki događaj, npr. zahtjev za resursom, izvršavanje komande, zamjena aplikativnog procesa u procesoru itd. Svi događaji u sistemu su uređeni. Taj odnos i odražava tok vremena. Izraženo kroz odnose, vrijeme se javlja kao čisto kvalitativna kategorija [83], [84]. To je tz. Diskretni model vremena. Ali i u prvom i u drugom slučaju vremenska osa je jedinstvena za čitav sistem.

Problemi sinhronizacije i uređenja događaja u sistemu su dovoljno prosti i njihova rješenja su dobro poznata. Od načina uvođenja vremena u model zavisi mnogo kao npr.: složenost problema sinhronizacije, izbor metodologije i mjernih sredstava za mjerenje radnog opterećenja.

4.3.3 Analitički modeli produktivnosti računarskih sistema

Analitički model se definiše sljedećom relacijom:

$$U = F(w) \qquad (4.1)$$

gdje je w – model radnog opterećenja, a $F(w)$ - model sistema, dobijen iz funkcionalnog modela primijenjen na konkretni model radnog opterećenja. Za dobijanje relacije 4.1 najčešće se koristi metoda masovnog opsluživanja.

Klasa problema koji se mogu riješiti pomoću odgovarajućih matematičkih metoda veoma je ograničena. Da bi se, neki i ne veoma složen sistem, opisao analitički neophodno je uvesti mnoge aproksimacije radi uprošćenja. Zato tačnost procjene, dobijene uz pomoć ovih metoda, nije velika. Bez obzira na ovo, analitički modeli su veoma rasprostranjeni. Uzrok ovome leži u činjenici, da se primjenom ovih modela dozvoljava da se brzo dobiju približne ocijene produktivnosti, pripreme parametri za imitacioni eksperiment, ocijeni perspektivnost izabranog smjera rada i tome slično.

4.3.4 Empirijski modeli produktivnosti računarskih sistema

Empirijski modeli produktivnosti se izgrađuju na osnovu analize podataka, dobijenih mjerenjem parametara radnog opterećenja i odgovarajućih vrijednosti produktivnosti sistema. U suštini empirijski podaci su jedna od formi predstavljanja vremenskog dijagrama korišćenja resursa sistema u onom aspektu koji je od interesa za konkretno istraživanje. Empirijski model može biti predstavljen tablično, grafički i slično, tj. u formi predstavljanja funkcija. Sama funkcija može, npr., biti dobijena uz pomoć regresione analize.

Podaci za analizu mogu biti dobijeni na razne načine. To mogu biti eksperimentalni podaci, dobijeni na realnom računarskom postrojenju, koji se javlja prototipom analizirajućeg sistema. Pri uslovima dovoljno "čistog" eksperimenta, to je najtačnija i adekvatna informacija o radu sistema. Nivo detaljizacije, na kom se sprovodi kako mjerenje radnog opterećenja, tako i karakteristika produktivnosti sistema ograničen je samo konstruktivnim osobinama opslužujućeg sistema i mjernim sredstvima. Ovaj metod dobijanja izlaznih podataka za izgradnju modela produktivnosti naziva se *metodom prototipova*.

Drugi način dobijanja podataka potrebnih za izgradnju modela produktivnosti javlja se numeričko riješenje funkcionalnog modela, kada je on izgrađen, npr. u obliku sistema s redovima [80]. Kao što je već rečeno podaci su u tom slučaju prilično uopšteni, koriste se srednje vrijednosti veličina, što se prirodno odražava na model produktivnosti. U osnovi stohastičkog prilaza leži nerazumijevanje detaljne problematike ili nemogućnost determinističkog opisa procesa koji se dešavaju, zato se i javljaju pokušaji da se opis uprosti kroz aproksimacije.

Korišćenje *emulatora* analizirajućeg sistema predstavlja drugi put dobijanja relevantnih podataka. Ideja ovog prilaza sastoji se u izgradnji emulatora komandi procesora. On imitira rad aparaturnih blokova procesora. Kao model radnog opterećenja koriste se etaloni. Ovaj prilaz se karakteriše:

- ✓ visokom tačnošću dobijenih podataka,
- ✓ vanredno visokim gubicima na dobijanju potrebnih podataka,

- ✓ niskom brzinom rada (0,01-0,001 brzine stvarnog računarskog sistema), što dozvoljava da se ispituju računarski sistemi samo na manjim i sporijim zadacima [24],
- ✓ ovdje se praktično ne uzima u obzir uticaj sistemskog softvera i podsistema ulaza/izlaza,
- ✓ razrada i izgradnja ovakvog kompleksa je teška i skupa, zahtijeva podrobno opisivanje rada računarskog sistema,
- ✓ dobijena informacija o ponašanju programa je sistemski zavisna, orijentisana je na određeni nivo detaljizacije funkcionisanja modela računarskog sistema i ne može biti iskorišćena kao radno opterećenje za ocjenu drugih računarskih sistema.

Za izgradnju empirijskih modela produktivnosti široko se primjenjuje i tehnika *imitacionog modeliranja*. Tehnika imitacionog modeliranja predstavlja kombinaciju mjerenja i modeliranja. Za njenu primjenu nužan je funkcionalni ili logički model sistema, model radnog opterećenja i sistem modeliranja. Imitacioni model proizvodi ponašanje sistema u saglasnosti sa opisom njenog funkcionalnog modela i modela radnog opterećenja, i mjeri ranije izabrane parametre ponašanja sistema, neophodne za izračunavanje vrijednosti produktivnosti [24], [25], [26].

Za razliku od slučaja primjene matematičke analize i aparata teorije masovnog opsluživanja, kada model sistema i model radnog opterećenja zahtijevaju značajna uprošćenja, imitaciono modeliranje dozvoljava izučavanje ponašanja sistema sa bilo kojim stepenom detaljizacije. Ponekad, kad se govori o imitacionom modeliranju ima se u vidu numeričko rješenje funkcionalnog modela, tj. zamjenu analitičke analize numeričkim rješenjem kada rješenje ne može biti dobijeno u analitičkoj formi [24], [29], [30], ili kada se ono uzima za provjeru analitičkog modela.

Primjena ove tehnike sa detaljnim funkcionalnim modelom sistema i radnog opterećenja daje veoma visok stepen tačnosti procjene 80 % i više. Ovo je veoma važno jer istraživač može intuitivno, na osnovu svog iskustva, dati procjenu nekog rješenja sa tačnošću 30% - 40%.

Imitacioni model može uključiti u sebi i faktore, koji ne mogu biti uključeni u analitičkom modelu. Takvi su npr. dinamička distribucija resursa, dodatni sistemski gubici i sl. Tehnika imitacionog modeliranja ne zahtijeva da se model radnog opterećenja opisuje stacionarnom statističkom raspodjelom. Iako se i takvi modeli radnog opterećenja mogu koristiti. U principu se može koristiti bilo koji model radnog opterećenja. U svakom konkretnom slučaju izbor modela radnog opterećenja određuje nivo detaljizacije imitacionog modela sistema.

Izbor modela radnog opterećenja zavisi od detaljizacije funkcionalnog modela sistema i od naših mogućnosti dobijanja podataka o realnom radnom opterećenju sa potrebnim stepenom podrobnosti. To zavisi i od postojanja ili odsustva pristupa realnom radnom opterećenju, sredstvima njegovog mjerenja, mogućnostima njihove ugradnje u sistem, "čistoće" mjerenja, tj. stepena smanjenja unošenja smetnji u sistem i sl.

Imitaciono modeliranje široko se primjenjuje kako za projektovanje novih računarskih sistema tako i za optimizaciju i podešavanje postojećih, procjenu računarskih mreža i tome slično.

4.4 Zaključci

Upoređivanjem gore razmatranih prilaza i metoda analize produktivnosti računarskih sistema tradicionalne arhitekture mogu se izvesti sljedeći zaključci, koji su bitni za izbor metode analize distribuiranih sistema.

Kao najmoćnije sredstvo analize javlja se prilaz, koji koristi tehniku prototipova spojenu sa etalonima. Jasno je da se kao etaloni u tom slučaju mogu koristiti sami programi. Taj prilaz dozvoljava da se dobiju vrlo tačne procjene, primjenjujući tehniku hardverskih posmatrača, koji praktično ne unose izobličenja u rad računarskog sistema. Međutim, ovaj prilaz ima niz ozbiljnih nedostataka:

- skup je kako u finansijskom tako i u vremenskom pogledu,
- zahtijeva izgradnju prototipa i kompletne sistemsko-softverske okoline i
- zahtijeva pri svom rješavanju prethodnu analizu i modeliranje.

Glavni zaključak se ogleda u tom, da mnoge karakteristike performansi distribuiranih računarskih sistema nelinearno zavise od izmijene obima mnogih resursa. Tako npr.: poznata je činjenica da sa porastom broja autonomnih računara, zbog uzastopnih kolizija, opada propusna sposobnost mreže [11], [16], [66], [76]. Ta činjenica zahtijeva dodatne izmijene funkcionalnog modela sistema, u datom slučaju prototipa. Ovo je povezano sa velikim teškoćama i gubicima i može se pokazati neprimjenljivim bilo iz tehničkih ili logičkih razloga.

Drugi prilaz, koji se može izdvojiti kao perspektivan, sastoji se od sakupljanja trase, kao modela radnog opterećenja i tehnike imitacionog modeliranja, kao sredstva izgradnje funkcionalnog modela. Ovakav prilaz dozvoljava dobijanje detaljne prognoze ponašanja sistema i zadovoljava uslove postavljenog zadatka. Kratak rezime primjene ove metode, kod klasičnih računarskih sistema, izgledao bi ovako:

- metoda čuva sve unutrašnje uzročno-posljedične veze, kako unutar programa tako i među procesima,
- dozvoljava se višekratno korišćenje jedne iste trase u raznim eksperimentima sa funkcionalnim modelima sistema,
- trase se mogu koristiti kao post-mortum program ili u realnom vremenu njegovog izvršavanja,
- ova metoda se primjenjivala samo kod jednoprocesorskih ili čvrsto spregnuti distribuiranih sistema, kada su se kao radno opterećenje pojavljivali modeli sekvencijalnih programa i samo kod već razvijenih sistema,
- korišćenje trasa, kao modela radnog opterećenja, javlja se sistemski zavisnim,
- nema specijalnog jezika za opis trasa i rad sa njima,
- postojeće metode i sredstva skupljanja trasa prilagođene su za sekvencijalne programe.
- imitacioni model računarskog sistema dozvoljava posmatranje procesa koji se u njemu dešavaju praktično u svim mogućim uslovima i sa bilo kojim stepenom detaljizacije.
- elementarne aktivnosti koje sačinjavaju procese imitiraju se, pri čemu se zadržava njegova unutrašnja struktura i vremenske sekvence.

Fizička i sistemsko-softverska struktura distribuiranih računarskih sistema kao i njihovi aplikativni programi, principijelno se razlikuje od analognih komponenti

tradicionalnih sistema. Ove razlike detaljno su opisane u literaturi: [4], [8], [9], [11], [16], [18], [103], [104], [108], [109], a ovdje će biti izdvojene samo one koje su relevantne za izbor metodologije istraživanja.

- Fizička, a prije svega komunikaciona sredina distribuiranog sistema ili sredina interakcije je intenzivno djeljivi resurs. U sistemu je prisutno više autonomnih računara koji mogu biti različitih performansi. Oni imaju svoj sistem mjerenja vremena i rade pod individualnim operativnim sistemom. Iz toga slijedi da je upravljanje u sistemu distribuirano, sistem mjerenja vremena nije jedinstven a tokovi upravljanja mogu u opštem slučaju biti nejednakih brzina.
- U izvršnoj sredini postoji veliki broj raznih tipova djeljivih resursa (informacionih, programskih i sl.). Upravljanje sa njima je decentralizovano, tj. izvršna sredina je distribuirana, što daje njenoj organizaciji principijelnu novinu. Upravljanje resursima u sistemu postaje veoma složeno a upravljanje bitno usložnjava probleme sinhronizacije sistemskih i aplikativnih programa koji se odvijaju u izvršnoj sredini.

Na osnovu izloženog, može se zaključiti da, modeliranje radnog opterećenja, orjentisano na trase, iako sa dobrom perspektivom, ne može se direktno koristiti za modeliranje radnog opterećenja distribuiranih sistema. Uzroci ovog leže u sljedećem:

- Metode orjentisana na trase se primjenjuju kod sistema čija razlika u arhitekturi između eksperimentalne (kontrolne) računarske konfiguracije i one koja se analizira, nosi u osnovi količinski karakter. Uticaj tih razlika imalo je ograničeni karakter na ponašanje aplikativnih programa. Takvih problema kao što je uticaj različitih formi paralelizma na zbirnu trasu tamo uopšte nije bilo.
- U sistemima, u kojima su se primjenjivale te metode, vrijeme je bilo jedinstveno, a brzina odvijanja operacija ista i konstantna. Problem uzajamnog uticaja programa i vremena odnosno analiza ponašanja distribuiranog programa u vremenu, koja nije jednostavna, nije se razmatrala.
- Za analizu sistema jedne dosta uske klase, svaki put se izgrađivao novi sistem skupljanja i obrade trase. Ove trase su bile orijentisane na konkretan skup posmatranih aktivnosti pri čemu je korišćen specifičan jezik opisa trase. Mnogi problemi skupljanja i obrade trase, karakteristični za distribuirane sisteme, nijesu razmatrani. Kao npr. problemi opisa trasa paralelnih procesa (engl. concurrency process) koji rade sa različitim brzinama nijesu rješavani [98], [99].

Ostale metode navedene u ovom poglavlju, ne daju dovoljan stepen detaljizacije tj. podrobnosti kako u prognoziranju ponašanja računarskih sistema, tako i opisa osobina ponašanja aplikativnih programa. Isto tako, paralelni i distribuirani računarski sistemi vanredno su osjetljivi na ponašanje programa. Zato je za te računarske sisteme vrlo važno sačuvati sve unutrašnje uzročno-posledične veze koje postoje u programu. Posebno se to odnosi na informacione i upravljačke tokove podataka. Sve ovo nije bilo prisutno u tradicionalnim metodama analize računarskih sistema.

Pregled postojećih metoda za analizu računarskih sistema pokazao je da poznata sredstva opisa ponašanja programa ne dozvoljavaju obuhvatanje svih poznatih formi i oblika paralelizma, kao ni opis raznih oblika nepredvidivog ponašanja distribuiranih programa, tz. *nedeterminizama* u ponašanju programa. Karakteristično za klasične prilaze je bilo to što se dinamika programa uvijek razmatrala u nerazdvojnoj vezi sa nekom hardverskom strukturom, a ne kao samostalna pojava.

GLAVA

5. PONAŠANJE APLIKATIVNIH PROGRAMA U DISTRIBUIRANOJ SREDINI

5.1 Uvod

Izgradnjom distribuiranih aplikativnih programa zacrtan je jedan od važnijih puteva povećanja brzine rješavanja složnih i teških zadataka. Međutim, procesi koje oni rađaju kao i njihova organizacija, znatno su složeniji nego kod sekvencijalnih programa ili paralelnih programa koji se izvršavaju u multiprocesorskom okruženju. Tradicionalne metode projektovanja i programiranja aplikacija nijesu primjenljive kod ovih sistema ili su u najboljem slučaju neefikasne. Osnovni teorijski problemi distribuirane obrade podataka su:

- radikalna izmijena numeričkih metoda,
- problemi distribucije izračunavanja,
- problemi sinhronizacije i koordinacije distribuiranih procesa.

Već duže vrijeme sprovode se intenzivna istraživanja različitih tehnologija za izgradnju aplikacija, koje će iskoristi sve prednosti distribuiranih računarskih platformi. Ovaj rad predstavlja doprinos u tom pravcu.

5.2 Aktivnosti, procesi i protokoli distribuiranih programa

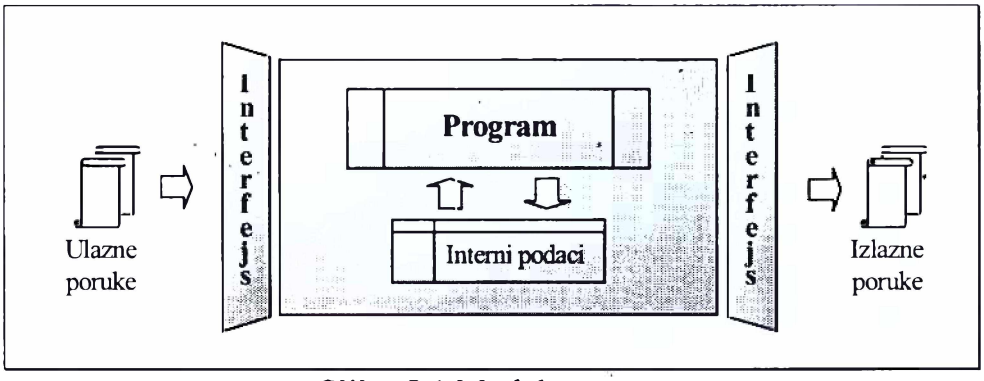
Između osnovnih konstrukcija programa, takvih kao što su promjenljive, operacije, operatori, upravljački primitivi i sl. i hardverskih elemenata distribuiranog računarskog sistema postoje određeni odnosi, koji se ne mogu uvijek jednoznačno odrediti. Radi daljeg razmatranja pogodno je pretpostaviti, da za ma koji programski jezik, koji služi za izgradnju programa, postoji hipotetička mašina, u kojoj za sve osnovne jezičke konstrukcije postoje odgovarajući uređaji koji ih neposredno realizuju. Tako npr. promjenljive se upoređuju sa odgovarajućim memorijskim ćelijama, operacije sa procesorima, upravljački primitivi sa upravljačkom jedinicama itd. Uz ove pretpostavke moguće je modelirajući ponašanje aplikativnih programa analizirati računarski sistem u cjelini.

Naredbe i operatori programa zadaju neke aktivnosti za obradu podataka, koje izvršava računarski sistem. Upravljački primitivi dozvoljavaju izgradnju složenih aktivnosti iz prostih, koje se realizuju uz pomoć naredbi i operatora. Skup upravljačkih primitiva u programu obrazuje njegovu strukturu upravljanja. Ona, kod sekvencijalnih programa, ima hijerarhijski karakter na osnovu toga što ova klasa programa ima hijerarhijsku strukturu: od operacija formiraju se operatori, operatori grupišu u sastavne operatore (blokove, module i sl).

Skup složenih aktivnosti definisanih programom označava se kao *proces*. Proces se može posmatrati i kao program u izvršavanju ili kao logički odnosno virtuelni procesor. Proces mora da posjeduje sposobnost obrade, tj. da obradi zahtjeve definisane programom ili tačnije tekstom programa. Sama obrada se vrši van procesa, a obezbjeđuje se razmjenom poruka sa drugim sličnim procesima ili procesima operativnog sistema. Startovanjem distribuiranog aplikativnog programa, istovremeno se

aktivira više paralelnih procesa. Ulazi u ove paralelne procese ostvaruje se uz pomoć ulaznih softverskih interfejsa. Uz pomoć ovih interfejsa procesi primaju poruke od sličnih procesa koji su aktivni u sistemu i nalaze se na istom ili drugim računarima. Skup internih podataka i atributa procesa definiše njegovo trenutno stanje i određuje aktivnosti koje treba preduzeti kada se primi poruka. Rezultati obrade ili računanja jednog procesa šalju se kroz izlazni interfejs u obliku poruke drugom procesu.

Najjednostavniji proces je onaj koji prima poruku, obradi je i vrati poruku kao odgovor. Ovaj događaja je nezavistan od bilo kog drugog koji se, u sistemu, odigrava prije ili poslije njega. Proces koji generiše poruku mora da zna adresu procesa, kome je poruka upućena. Kada početni proces pošalje poruku, prelazi u stanje čekanja u kome čeka na odgovor na jednom od svojih ulaznih interfejsa. Proces koji primi poruku, izvršava funkcije definisane u njoj, formira izlaznu poruku koja sadrži rezultate obrade i adresu početnog procesa i šalje je preko svog izlaznog interfejsa. Slika 5.1 ilustruje pojam jednostavnog procesa.



Slika 5.1 Model procesa

Trajanje procesa u trenutku njegovog stvaranja tj. pisanja programa nije poznato. Vrijeme njegovog izvršavanja, odnosno trajanja, zavisi od vrste i konfiguracije hardvera na kom se izvršava, od aktivnosti paralelnih tj. konkurentskih procesa koji zajedno sa njima rade na istom zadatku, od opterećenja zajedničkih resursa i niza drugih razloga. Osim toga kod uslovnog grananja ne zna se na koju će stranu program otići. Zbog toga se uvodi sljedeća pretpostavka o proizvoljnom pozitivnom vremenu trajanja: koliko god da proces traje ne smije se narušiti konzistentnost programa.

Interakcija dva distribuirana paralelna procesa ostvaruje se: sinhronizacijom i komunikacijom. Kada jedan proces stigne do određene tačke on šalje signal sinhronizacije drugom procesu, čime ga obaveštava o svom trenutnom stanju. Dva su glavna tipa komunikacije između paralelnih procesa: razmjena podataka i pristup opštim resursima koji nisu djeljivi. Metode orijentisane na razmjenu podataka označavaju se kao metode komunikacije razmjenom poruka. Komunikacija se obavlja putem razmjene odnosno prosljeđivanja poruka (engl. *message passing*). Da bi se obezbijedila korektnost izvršavanja distribuiranih programa uvode se tz. nedjeljive operacije. tj. operacije koje se moraju izvršiti u cjelini prije prenosa kontrole na drugi proces. Za međusobno isključenje rada procesa primjenjuju se različite proceduralne metode a sama komunikacija između procesa, koja se obavlja razmjenom poruka, može biti sinhrona i asinhrona.

Distribuirani aplikativni programi se još definišu i kao sistemi sastavljeni iz više paralelnih procesa koji komuniciraju kroz eksplicitno prosljeđivanje poruka. U tom kontekstu uvode se logičke komponente distribuiranog programa: aktivnost, proces i komponente sa više procesa koje se konkurentno izvršavaju. Uopšte, sistemi sa više paralelnih procesa označavaju se kao *konkurentni sistemi*. Distribuirani aplikativni programi pripadaju klasi konkurentnih programa, jer se sastoje od više procesa koji se paralelno izvršavaju na odvojenim računarima [98], [99].

Kako je već naglašeno, dva procesa koja se odvijaju paralelno na različitim lokacijama odnosno na različitim računarima koriste razmjenu poruka da bi koordinirali svoje akcije i postigli međusobnu sinhronizaciju. Ova razmjena poruka definisana je procedurama opisanim unutar paralelnih programa. Takve procedure se nazivaju *protokoli*. Njihova glavna karakteristika je mogućnost da rade kada su vrijeme i redosljed odvijanja događaja nepoznati [4], [5], [7], [8], [98]. Termin protokol se koristi za opisivanje procedure za razmjenu informacija između procesa u mrežama za prenos podataka a isto tako i u multiprocesorskim i distribuiranim sistemima i sistemima koji rade u realnom vremenu. Uopšte, protokoli se primjenjuju u svim sistemima u kojima ne postoji vremenska zavisnost između nastajanja događaja i posljedica koje oni izazivaju. Ukratko, protokolima se opisuju procedure za upravljanje interakcijom paralelnih procesa.

Funkcije protokola izvršavaju se razmjenom poruka između procesa. Format i značenje ovih poruka predstavljaju logičku definiciju protokola. Pravila u proceduri određuju aktivnosti procesa koji učestvuju u protokolu. Skup svih pravila čini proceduralnu definiciju protokola. Koristeći ove pojmove, može se dati formalna definicija protokola. To je logička i proceduralna specifikacija komunikacionog mehanizma između paralelnih procesa. Protokoli se definišu u programima, preko sintakse i semantike, a realizuje se uz pomoć procesa. Logička definicija predstavlja sintaksu, a proceduralna semantiku protokola. Definicija realizacije opisuje aktivnosti koje zahtijeva proces. Aktivnosti procesa se mogu opisati pomoću stanja procesa i događaja koji prouzrokuju prelaske iz jednog stanja u drugo. Tako npr. dolazak poruke u proces je događaj.

5.3 Načini predstavljanja sekvencijalnih i paralelnih programa

Koncepcijsku osnovu klasičnog sekvencijalnog programiranja obrazuje pojam algoritma. *Algoritam* je tačno uputstvo, koje definiše proces sekvencijalnog preslikavanja konstruktivnih objekata, opisanih diskretnim koracima. Algoritam počinje od proizvoljnih početnih podataka i usmjeren je na dobijanje nekog traženog rezultata, jednoznačno određenog tim početnim podacima. Sekvencijalni program opisuje algoritam i realizuje se na računaru kao računarski proces, u kojem se svaki korak sastoji od izvršavanja neke aktivnosti, zadate operacijom ili operatorom programa. Struktura upravljanja sekvencijalnim programom određuje niz tih aktivnosti, odnosno koraka.

Ukoliko se proces, stvoren pri izvršavanju prostih linearnih programa, može predstaviti kao linearno uređen niz aktivnosti, onda se za proste linearne programe, koji

ne sadrže uslovne naredbe, naredbe bezuslovnog skoka, ciklične naredbe i sl., proces može izjednačiti sa samim programom. Linearni programi posjeduju dvije važne osobine. Prva se sastoji u tome, da za jedne iste početne podatke program može stvoriti ne jedan isti strogo određen sekvencijalni proces, već više njih. Pri svakom konkretnom izvršavanju stvara se neki od tih procesa. Program je jednoznačan, ako se sa bilo kojim od dozvoljenih njegovih procesa dobija isti rezultat [97], [98], [99]. Druga osobina linearnih programa se sastoji u tome da se uporedo sa sekvencijalnim procesima programa mogu roditi paralelni procesi, u kojima se neki operatori izvršavaju istovremeno.

Glavna razlika između sekvencijalnih i paralelnih programa javlja se na nivou strukture upravljanja. Ako se sekvencijalni procesi predstavljaju u obliku linearnog niza aktivnosti, onda je za paralelne procese nužan drugi oblik predstavljanja. Uzimajući u obzir da aktivnosti imaju vrijeme odvijanja, najpotpunijom formom predstavljanja paralelnog procesa može poslužiti *prostorno-vremenski dijagram*. Isto tako često je jednostavno i dovoljno predstaviti paralelne procese kao konačan ili beskonačan *djelimično uređen skup* aktivnosti. Djelimični poredak se zadaje relacijom "neposredno slijedi", npr. neka aktivnost x_2 neposredno slijedi za aktivnošću x_1, ako x_2 počinje pošto se x_1 završi, a među tim aktivnostima nema nikakve aktivnosti x_3, koji počinje poslije završetka aktivnosti x_1 i završava se do početka x_2. Takvo predstavljanje procesa prikazuje se u obliku grafa, u kojem su čvorovi aktivnosti x_1 i x_2 a iz čvora x_1 u čvor x_2 ide luk, ako x_2 neposredno slijedi za x_1. Tranzitivno zatvaranje relacija "neposredno slijedi" na skupu aktivnosti u procesu rađa relacije slijeda aktivnosti: aktivnost x_2 slijedi poslije x_1 u datom procesu, ako postoji put po grafu iz čvora x_1 u čvor x_2.

Paralelni procesi se mogu međusobno upoređivati po "paralelnosti". Iz skupa svih međusobno ekvivalentnih procesa mogu se izdvojiti *maksimalno paralelni*, koji nisu "manje paralelni", od svih ostalih procesa iz tog skupa.

Osnovnim izvorom paralelizma u programima služi informaciona nezavisnost operatora i operacija, a u procesima to je informaciona nezavisnost odgovarajućih aktivnosti. Tako npr., aktivnost x_2 neposredno (informaciono) zavisi od aktivnosti x_1, ako se rezultat aktivnosti x_1 koristi u svojstvu operatora aktivnosti x_2. Prenos rezultata može se ostvariti uz pomoć zajedničke memorije, kroz mrežu, pomoću specijalnih registara ili slanjem poruka između procesa.

Tranzitivno zatvaranje neposredne zavisnosti rađa relaciju informacione zavisnosti. Za linearni program može se izgraditi *informacioni graf programa*, u kojem su čvorovi operatori i međusobno su povezani lukom ako neposredno zavise jedan od drugog. Analogno se može izgraditi informacioni graf procesa. Uporedo sa informacionim zavisnostima između aktivnosti mogu postojati i drugi tipovi zavisnosti, ograničavajući mogućnosti paralelnog izvršavanja tih aktivnosti.

Za predstavljanje i analizu strukture upravljanja paralelnih programa postoji veći broj formalnih modela. Sve popularniji postaju modeli, zasnovani na mrežama Petri. *Mreže Petri* su bile uvedene kao apstraktni modeli paralelnih diskretnih sistema i procesa koji se u njima odvijaju. Pomoću termina mreža Petri adekvatnije nego pomoću klasične teorije automata može se opisati nedeterminizam u funkcionisanju sistema sa složenim dejstvima između njenih lokalnih paralelnih fragmenata.

5.4 Metode izgradnje radnog opterećenja DRS-a

Jednim od ključnih pitanja pri rješavanju zadatka analize performansi distribuiranog računarskog sistema javlja se to kako i u kojim terminima opisati ponašanje aplikativnih programa u distribuiranoj sredini, odnosno kako izgraditi model radnog opterećenja posmatranog sistema. Rješenje tog problema je neophodno kako za sredstva opisa ponašanja programa tako i za razradu metoda i sredstava njenog mjerenja.

Većina dosadašnjih teorija [63], [64], [70], [71], [72] predstavljanja aplikativnih programa u distribuiranoj sredini, ne obuhvata sve njihove bitne karakteristike. Ove teorije su prije svega izgrađivane sa ciljem izgradnje matematičkog aparata za specifikaciju ponašanja programa i istraživanje njihovih ekvivalentnih transformacija. Kod ovih modela ili nijesu razdvojeni aplikativni program i fizička sredina njegovog izvršavanja ili fizička sredina i vrijeme kao metričke veličine nijesu uzete u obzir. Zadatak ovog istraživanja je izgradnja metode za količinsku i algoritamsku analizu distribuiranih računarskih sistema, kako je to već opisano u poglavlju 3.2. U tom cilju treba predstaviti model koji će obuhvatiti ne samo ponašanje programa, nego i karakteristike hardvera i sistemskog softvera. On mora opisati uticaj fizičke sredine na dinamiku sinhronizacije i komunikacije paralelnih procesa. Model treba da obuhvata mnogostrukost vremena, kao bitnu karakteristiku distribuiranih sistema, a takođe hijerarhijsku i strukturnu organizaciju računarskih sistema. Ovu vrstu modela prvi put je predložio prof. R.L. Smeljanski [77], [78], [80], koristeći je za predstavljanje multiprocesorskog sistema sa djeljivom memorijom i jedinstvenim operativnim sistemom. Ovdje će ona biti proširena i primijenjena na danas veoma aktuelnu klasu distribuiranih računarskih sistema, koja je detaljnije opisana u drugom poglavlju.

Kako je već dijelom opisano u poglavlju 4.2, postoje dva principijelno različita prilaza u odgovoru na ovo pitanje:

1. *Denotacioni prilaz* opisuje ponašanje programa u terminima vrijednosti memorijskih promjenljivih koje taj program sadrži, odnosno stanja radnog opterećenja npr. upita, zahtjeva ili transakcija opisanih programom [110];
2. *Operacioni prilaz* posmatra ponašanje programa kao niz događaja koji se javljaju u distribuiranoj računarskoj sredini [111].

Na računarima sa tradicionalnom arhitekturom uspješno su se primjenjivale obe metode, u sljedećem tekstu razmotriće se mogućnost njihove primjene u distribuiranim računarskim sistemima.

5.4.1 Denotacioni prilaz u opisu ponašanja aplikativnog programa

Kod denotacionog (engl. *denotation*) prilaza se pretpostavlja da se sve aktivnosti, koje proizilaze između sekvencijalnih izmijena stanja, jednoznačno izvršavaju po tekstu programa, a za račun sekvencijalno i strogo determinisanog karaktera izvršavanja programa [110]. Drugim riječima, pri neizmijenjenosti ulaznih podataka istorija izvršavanja sekvencijalnog programa je neizmijenjena. Obrada se vrši uvijek jednim istim putem i dobijaju se isti izlazni podaci.

Ovakvim načinom opisa ponašanja programa u njemu se eksplicitno prikazuju one aktivnosti programa koje izazivaju izmijenu sadržaja memorijskih lokacija dostupnih programeru. Na osnovu ovakvog posmatranja programa slijedi stroga

definisanost i mogućnost obnavljanja obrade pri neizmijenjenim izvornim podacima. Uvijek je poznato potpuno stanje obrade u svim procesima, određenih obično kao cjelokupnost vrijednosti svih promjenljivih programa.

Za sistem distribuiranih programa, koji se izvršava u distribuiranoj računarskoj sredini, ove pretpostavke nijesu tačne. Ponašanje sistema distribuiranih programa podložno je uticaju sinhronizacije i konkurencije između procesa za djeljivim resursima. Taj uticaj može izmijeniti ponašanje programa od izvršavanja do izvršavanja pri neizmijenjenosti izvornih podataka i rezultata izvršavanja. To je tz. *nedeterminizam* ponašanja programa. Slijedi da identičnost ponavljana obrade u distribuiranom računarskom sistemu praktično nije moguća.

Zbog odsustva centralizovanog upravljanja kod distribuiranog računarskog sistema nemoguće je garantovati sinhronizaciju smijene stanja u svim procesima. To usložnjava potpuno korektno određivanja stanja programa.

Za potpuno određivanje stanja obrade u sistemu programa, potrebno je omogućiti zaustavljanje izvršavanja programa da bi se obezbijedio jednovremeni pristup (engl. *debugging*) vrijednostima promjenljivih u memorijskim lokacijama i registrima svih procesa [54]. Izvršavanje sistema distribuiranih programa nije moguće uvijek zaustaviti, tako da ga u datom trenutku možemo ponovo obnoviti. Ponovno obnavljanje stanja i korekcija izmijena, prouzrokovanih zaustavljanjem sistema programa, nije jednostavan zadatak. Njegovo rješavanje zahtijeva dodatne troškove pri mjerenju ili modeliranju.

Dakle, korišćenje denotacionog prilaza povezano je sa nizom teškoća, izazvanih osobenostima distribuiranih računarskih sistema i vanredno visokom složenošću toga prilaza zbog velikog broja stanja kroz koje sistem može prolaziti. U takvim uslovima bolje odgovara operacioni prilaz [80].

5.4.2 Operacioni način opisa ponašanja aplikativnih programa i njegova primjena u izgradnji imitacionog modela distribuiranih sistema

Kod operacionog modela proces izvršavanja programa u distribuiranoj sredini predstavlja se u obliku niza događaja. Događaj predstavlja početak ili kraj aktivnosti koja je od interesa za istraživanje. Operacioni opis ponašanja distribuiranih aplikativnih programa, predstavlja osnovu za izgradnju modela sistema, zasnovanog na koncepciji *stanja* i *prelaza* između stanja [29], [123]. Kod ovog modela, stanje distribuiranog računarskog sistema u trenutku t određuje se kao skup vrijednosti svih njegovih parametara značajnih za analizu. Bilo koja izmijena tih vrijednosti može se posmatrati kao prijelaz ka drugom stanju. Pomoću operacionog opisa gradi se operacioni model sistema distribuiranih programa, odnosno u ovom slučaju radnog opterećenja distribuiranog računarskog sistema. Ako ponašanje modela u vremenu u osnovi predstavlja ponašanje distribuiranog računarskog sistema, koje odgovara njegovim stanjima i prijelazima između tih stanja, dobija se *imitacioni model* ovog sistema.

5.4.2.1 Osnovi objekti operacionog modela

U poglavlju 5.2 program u izvršavanju je označen kao proces, odnosno kao dinamička forma programa. Prema tome, prirodno je da se dinamičko ponašanje programa opiše uz pomoć objekata kao što su:

- ***Aktivnost*** (engl. *activity*) u operacionom modelu je osnovni dinamički objekat, i predstavlja stanje u kom se distribuirani program nalazi. To može biti npr.: izvršavanje mikrokomande, komande, izvršavanje funkcije, slanje saopštenja, formiranje zahtjeva za pristup bazi podataka itd. Pod aktivnošću se može posmatrati i izvršavanje složenijih operacija, sastavljenih kao skup prostih. Aktivnost može biti distribuirana, tj. može uključivati u sebi aktivnosti koje se odvijaju istovremeno na dva ili više računara.
- ***Događaj*** (engl. *event*) je početak ili kraj aktivnosti. On se dešava u trenutku kada pristigne odluka o početku ili završetku aktivnosti, odnosno, prijelasku iz jednog stanja u drugo.
- ***Proces*** se sada može definisati kao vremenski orijentisan niz događaja koji može biti sastavljen iz više aktivnosti. Na taj način se proces u svakom trenutku posmatranja nalazi u stanju određene aktivnosti, ili prelazi iz jednog takvog stanja u drugo.

Događaju se kao dinamičkom objektu mogu dodijeliti tri osnovne karakteristike:

- *vrijeme pojavljivanja* u lokalnom sistemu odbrojavanja,
- *lokacija* ili mjesto pojavljivanja predstavlja identifikaciju procesa u kojem se događaj javio i
- *atributi* (engl. *attribute*), koji predstavljaju skup osobina ili znakova one aktivnosti čijem početku ili kraju događaj odgovara.

Aktivnost posjeduje sljedeće karakteristike:

- *lokaciju* ili mjesto, odnosno identifikaciju procesa u kojem se aktivnost javila,
- *težinu* (težinski faktor), koja određuje troškove ili gubitke izvršavanja te aktivnosti
- *suštinu,* koja predstavlja fizičku smisao te aktivnosti.

Distribuiranu aktivnost karakterišu distribuirani događaji. Distribuirani događaj obrazuju prosti događaji, koji se dešavaju na autonomnim računarima. Ovi prosti događaji odgovaraju aktivnostima koje se odvijaju na autonomnim računarima.

5.4.2.2 Odnos programa i metričkog vremena, prostorno-vremenski sistem

Za težinski faktor aktivnosti ponekad se uzima vrijeme. Tako je npr. u metodi orjentisanoj na trase Čen koristio vrijeme kao težinski faktor [46]. To je i odredilo sistemsku zavisnost metode koju je on predložio. Stvar je u tome da nikakav program ne koristi metričko vrijeme. Čak šta više on ne sadrži ni astronomsko vrijeme. Program određuje samo logički niz događaja i uzročno-posljedičnu vezu među njima. Zato je vrijeme kao težinski faktor ili težinska mjera uvijek sistemski zavistan [74], [75], [77], [79], [80]. Izgradnja sistemski-nezavisne mjere (mjere ponašanja aplikativnog programa koji neće zavisiti od računarskog sistema na kom se dati program izvršava) je fundamentalan problem koji će biti razmatran u sljedećem poglavlju.

Razjašnjenje odnosa programa i metričkog vremena. je važno za razumijevanje rješenja problema izgradnje sistemski-nezavisne mjere ili sistemski-nezavisnog težinskog faktora neke računarske operacije. To pitanje će se razmotriti na primjeru astronomskog vremena kao važnoj formi metričkog vremena. Pod astronomskim vremenom podrazumijeva se vrijeme čiji kôd korisnik može odrediti pomoću astronomskog posmatranja. To je pojava (fenomen) koji nikako ne utiče na rad računarskog sistema, ali je ona izuzetno važna za njegove korisnike. Povezivanje

sekvencijalnog niza aktivnosti programa za osu vremena ne zavisi od teksta programa, isto tako kao što od njega ne zavisi brzina kretanja po tom sekvencijalnom nizu aktivnosti [80], [83], [84], [85]. Sa stanovišta programa koji bi trebalo da rade u realnom vremenu to bi izgledalo paradoksalno [99]. Zato se postavlja pitanje šta znači imati (sadržati) vrijeme. Saglasno Ajšajnovoj teoriji relativiteta nešto ima vrijeme, ako u njemu postoji ciklički proces, na brzinu kojeg ono ne može uticati ni direktno ni indirektno. Takvog cikličkog (fizičkog) procesa program nema, njega njemu daje fizička struktura sistema na kojem se program izvršava. Program se može snabdeti cikličkim procesom koji će računati vrijeme, ali to "programsko" ili logičko vrijeme neposredno zavisi od sredine gdje se program izvršava.

Distribuirani aplikativni program, instaliran na distribuiranom računarskom sistemu obrazuje skup koordinatnih sistema prostor-vrijeme. Svaki autonomni računar određuje svoje lokalno fizičko vrijeme, a svaki proces koji on izvršava ili koji se na njemu izvršava, daje prostornu osu. Drugim riječima cjelokupnost svih kompleta procesa koji se izvršavaju na autonomnim računarima obrazuju jedan takav sistem. Ti, nazovimo ih lokalnim koordinatnim sistemima stavljeni (potopljeni) su u globalni sistem gdje se kao vremenska osa pojavljuje vrijeme korisnika ili astronomsko vrijeme. Sinhronizacija procesa je neophodna, da bi se doveli u međusobni odnos njihove vremenske koordinate. To je određivanje relacija između programa koji se izvršavaju na različitim autonomnim računarima. To je ukratko pogled na prostorno-vremenski sistem u kojem se izvršava distribuirani aplikativni. Ovo je potrebno kasnije za izgradnju težinskog faktora aktivnosti i definisanje formalnog modela distribuiranog računarskog sistema [85].

U ovom radu će se koristiti operacioni prilaz za opis ponašanja aplikativnih programa u distribuiranoj sredini. Jednim iz osnovnih njegovih prednosti javlja se to što je on prost i u isto vrijeme daje značajnu slobodu u izboru matematičke koncepcije opisa paralelizma, i opisa uzajamnih veza programa i vremena. Izbor koncepcije je jedan od fundamentalnih problema teorije programiranja, isto kao teorija računarskih sistema koja je sve do danas ostala otvorena. Takođe je važno i to što je u okviru tog prilaza skupljeno metodičko iskustvo i tehnika analize različitih svojstava programa, koji bi se pri određenim uslovima mogli koristiti.

5.5 Definisanje sistemski-nezavisne mjere programa

Uz primjenu operacionog modela, ponašanje distribuiranog aplikativnog programa može se posmatrati kao skup istorija svih procesa tog programa koji se izvršavaju na posmatranom distribuiranom računarskom sistemu. Istoriju distribuiranog programa formiraju sekvencijalni dijelovi tog programa koji se izvršavaju na autonomnim računarima. Sa druge strane, pod istorijom programa podrazumijeva se niz sekvencijalnih aktivnosti, određenih izvornim tekstom tog programa. Svaku aktivnost karakteriše par događaja jedan koji odgovara početku i drugi koji odgovara kraju aktivnosti. Na taj način se, ponašanje programa svodi na djelimično-uređen skup događaja. Djelimično-uređenje u tom skupu ustanovljeno je sinhronizacijom i međusobnom koordinacijom paralelnih procesa distribuiranih programa, koji se najčešće izvršavaju na odvojenim računarima.

Ovdje se sada, kao ključno, postaviti pitanje: da li se po rezultatima posmatranja ponašanja sistema distribuiranog programa u nekoj *eksperimentalnoj* računarskoj sredini može prognozirati njeno ponašanje u drugoj, *analizirajućoj*, distribuiranoj računarskoj sredini? Pod prognoziranjem se podrazumijeva ne samo predviđanje uzastopnosti odnosno sekvencijalnosti aktivnosti programa u analiziranom sistemu već i vrijeme izvršavanja tih aktivnosti u posmatranom sistemu. Drugim riječima treba odgovoriti na pitanje, da li postoji nezavisnost programa od računarske sredine ili je osobina programa takva da je njegovo ponašanje jedinstveno u svakoj računarskoj sredini.

Za odgovor na ovo pitanje neophodno je riješiti sljedeće probleme:

1. Dokazati postojanje takvih aktivnosti, koje bi bile svojstvene svim sredinama gdje može funkcionisati posmatrani sistem distribuiranih programa. Pri tome sekvencijalni niz tih aktivnosti, kao opis ponašanja treba da odražava sve uzajamne veze (relacije) po informaciji i upravljanju, koje su svojstvene sistemu distribuiranih programa;
2. Znati predskazati sekvencijalnost aktivnosti programa u ciljnoj (analizirajućoj) sredini, znajući razliku njene arhitekture od arhitekture eksperimentalne računarske sredine i sekvencijalnosti aktivnosti programa u eksperimentalnoj sredini. Zbog nedeterminizma ponašanja distribuiranih programa i arhitekturnih razlika skup istorija paralelnih procesa, konkretnog izvršavanja programa na eksperimentalnom sistemu, može se razlikovati od skupa tih istorija koji bi proizišli u analizirajućoj sredini.
3. Znati zadati, odnosno tako specificirati ponašanje programa, da se za svaku aktivnost u ponašanju sistema distribuiranih programa može izračunati interval vremena, neophodan analizirajućoj računarskoj sredini za njeno izvršavanje. Drugim riječima potrebna je *mjera računarskog rada* koji izvršava računarska sredina pod uticajem programa. Ta mjera sa jedne strane ne treba da zavisi od fizičkih karakteristika konkretnog računarskog sistema, a sa druge strane ona treba da dopusti efektivnu interpretaciju na konkretni računarski sistem. Ovakva mjera računarskog rada označena je kao sistemski-nezavisna mjera programa ili *invarijanta* ponašanja programa (engl. *Program behavior Invariant.*).

Ta tri problema u tom poretku, kako su formulisani i određuju dalji plan istraživanja.

5.5.1 Logički resursi

Sekvencijalni program se može posmatrati kao algoritam, opisan za izvršavanje u nekoj računarskoj sredini. U praksi programiranja koriste se razna sredstva za opis ili predstavljanje algoritma. Tu spadaju: blok dijagrami, programski jezici visokog nivoa, asemblerski jezici, mašinski kôd i sl. Ova sredstva u raznim jezicima obezbjeđuju različit stepen zavisnosti programa od fizičke sredine. To je posljedica mnogih faktora npr. toga što se sredstva upravljanja izračunavanjima, pristupa podacima i obrade podataka u programskim jezicima standardizuju kako sintaksno tako i semantički. Sa druge strane, ta sredstva opisa su spoljašnja u odnosu na program. Oni se obično ne realizuju direktno u fizičkoj sredini računarskog sistema [1], [2], [3].

Svi viši programski jezici, takvi kao što su: C, C^{++}, Paskal, Fortran, Cobol, Ada, Modula, PL/1 i sl., imaju veoma slične principe i sredstva programiranja. Kod ovih proceduralno-orijentisanih jezika primjenjuju se složene strukture podataka (nizovi, matrice, zapisi, datoteke i sl.) i operacije nad njima. Ove strukture nemaju neposrednu

podršku u mikroprocesorskoj računarskoj sredini. Slično je i sa operacionim strukturama kao što su: funkcije, procedure, sastavni blokovi, pozivi opštih procedura, automatska konverzija različitih tipova podataka, operacije ulaza/izlaza, prenos poruka i sl. Njihovu realizaciju i obradu obezbjeđuje sistemsko-softverska struktura računarskog sistema.

Pri izgradnji jezičkih sredstava opisa algoritma teži se njihovoj nezavisnosti od fizičke i sistemsko-softverske sredine njihovog kasnijeg izvršavanja. Ova nezavisnost se postiže time što u osnovi svakog od njih leži algoritam, čiji način realizacije nije bitan. Važno je samo to, da se ti algoritmi uvijek formiraju u računarskoj sredini kao samostalni nezavisni objekti, spoljašnji u odnosu na aplikativni program i u činjenici da sama sredstva imaju jezičko-algoritamsku prirodu. Ova sredstva se mogu označiti kao logički resursi ili sistemsko-softverski resursi. *Logički resursi* računarskog sistema se mogu predstaviti kao sredstvo upravljanja izračunavanjima, pristupa i obrade podataka, koji pruža odnosno nudi programu sistemsko-softverska struktura kao uslugu aplikativnim programima. Logički resursi se manifestuju u vidu procesa operativnog sistema i sistemskih uslužnih programa i predstavljaju neku vrstu interfejsa između aplikativnog programa i hardverske strukture distribuiranog sistema.

U cilju određivanja osnovnih karakteristika logičkog resursa, kao pojma, razmotriće se njegova uzajamna veza sa fizičkom i sistemsko-softverskom strukturom. Za to će biti iskorišćeni pojmovi kao što su povezivanje (engl. *linking*) i vrijeme povezivanja. Pod povezivanjem logičkog resursa podrazumijeva se upoređivanje obraćanja k logičkom resursu sa realizacijom tog resursa u terminima fizičke sredine. Drugim riječima, povezivanje je proces pretvaranja programa u formu, pogodnu za njegovo izvršavanje u izabranoj fizičkoj sredini. Vremenski period kada nastaje to povezivanje naziva se vremenom povezivanja.

Sve metode povezivanja možemo, s obzirom na vrijeme povezivanja podijeliti na statičke i dinamičke. U skladu sa tim sve logičke resurse možemo podijeliti na statičke i dinamičke. Statičko povezivanje se dešava pri kompilaciji programa i tokom izvršavanja programa ono se ne mijenja. Obraćanje logičkim resursima zamjenjuje se pri statičkom povezivanju takvom realizacijom tog resursa u terminima fizičke sredine, u kojoj se ne koriste biblioteke potprograma, sistemskih poziva i slična programska sredstva koja postoje u računarskoj sredini nezavisno od programa koji se kompiluje. Statičkog logičkog resursa nema u sistemsko-softverskoj sredini do početka izvršavanja programa. To je jedna od razlika logičkog resursa od pojma virtuelnog resursa, korišćenog kod operacionih sistema [3], [15].

Dinamičko povezivanje obično se dešava tokom startovanja i izvršavanja programa. Ono se primjenjuje samo kod tih logičkih resursa, čija je realizacija u sistemsko-softverskoj sredini nezavisna od programa. Obično se ti logički resursi formiraju u obliku potprograma, sistemskih rutina i sličnih tehnika. Dinamički logički resurs postoji u sistemsko-softverskoj sredini nezavisno od aplikativnog programa. On obično ne zavisi od konteksta u kom se koristi resurs u programu. Određeni dio povezivanja dinamičkog logičkog resursa se dešava u toku kompilacije programa. Taj dio obrazuje niz komandi fizičke sredine, koji obezbjeđuje prenos parametara i upravljanje nadležnom programu, sistemskom pozivu i sl. Ovdje je osnovno to, da taj niz ne zavisi od konteksta obraćanja ka odgovarajućem logičkom resursu, njega određuju pravila sistemsko-softverske sredine. Kod statičkog logičkog resursa uvijek postoji

zavisnost povezivanja od konteksta u kom će se resurs koristiti, a osnovno je to da povezivanje statičkog logičkog resursa bitno zavisi od generatora koda primjenjenog u kompajleru.

Program ne može sadržati samo statičke logičke resurse. U njemu se uvijek postoji djeljivi logički resurs, koji mogu koristiti istovremeno nekoliko procesa, zato povezivanje sa njim može biti samo dinamičko. Takva su sredstva pristupa datotekama, bazama podataka, sredstva ulaza-izlaza, praktično sva sredstva podrške interakcija i sinhronizacije paralelnih procesa. Oni su, kod Windows operativnih sistema, obično nalaze u tz. dinamičkim (DLL) bibliotekama. Štaviše, čistih statičkih logičkih resursa, tj. pri realizaciji kojih ne mora biti obraćanja sistemsko-softverskoj strukturi, strogo govoreći nema. Npr. zahtjevi za obraćanje ka fizičkoj sredini, pri određenim uslovima mogu izazvati prekid, čiji rezultat će biti obraćanje ka sistemsko-softverskoj strukturi, odnosno operativnom sistemu. Smatra se da ako programer izgradi takvu situaciju u programu, to je slučaj dinamičkog povezivanja. Ako se ta situacija javila kao posljedica greške u programu tada slijedi statičko povezivanje. Pretpostavlja se da u ponašanju pravilnog programa nema obraćanja sistemsko-softverskoj sredini u slučaju prekida.

Dinamički logički resursi obrazuju hijerarhiju, u osnovi koje leži fizička sredina. Upravo ta hijerarhija i obrazuje to što se naziva sistemsko-softverska sredina. Zavisnost programa od specifičnosti fizičke sredine sa povećanjem nivoa korišćenih logičkih resursa u toj hijerarhiji slabi i počev od nekog nivoa, praktično nestaje.

Logičkom resursu se mogu dodijeliti tri karakteristike: sintaksna, funkcionalna i operaciona. Sintaksna ga karakteriše kao jezičku strukturu, u suštini određuje formu obraćanja k njemu. Funkcionalna određuje funkcionalni sastav i svaku funkciju odvojeno. Logički resurs može imati u sebi nekoliko funkcija. Operaciona određuje arbitražu konflikata pri obraćanju ka datom resursu i ograničenja korišćenja funkcija koji ulaze u njegov sastav.

Nivo logičkog resursa nije fiksiran. Njegov izbor određuju ciljevi analize, i naravno, do kog stepena detaljizacije se razmatra ponašanja programa. Jasno je da je taj nivo detaljizacije ograničen samo nivoom operatora i strukture podataka korišćenog jezika programiranja.

Logički resursi su sredstvo upravljanja izračunavanjima i pristupa podacima. Zato oni moraju biti osjetljivi prema informacionim interfejsima, inače pomoću njih ne bi bilo moguće upravljati izračunavanjima. Slijedi da sekvencijalni niz tj. uzastopnost obraćanja logičkim resursima odražava sve uzajamne veze, koje postoje u programu.

Ako se ponašanje programa opiše pomoću događaja koji karakterišu korišćenje logičkih resursa, onda svi mogući nizovi takvih događaja, koji nastaju pri radu programa daju sistemski nezavistan opis njegovog ponašanja, koji čuva sve informacione i upravljačke uzajamne veze. Ovaj opis je sistemski nezavistan u tom smislu što u bilo kojoj sredini u kojoj postoji odgovarajući skup logičkih resursa, aplikativni program rađa jedan od nizova obraćanja k njemu.

5.5.2 Klasifikacija ponašanja programa

Klasifikacija ponašanja programa je potrebna za rješavanje sljedećeg problema: kako da se znajući ponašanje programa u instrumentalnoj sredini i razlike te sredine od analizirajuće, odredi niz aktivnosti programa u analizirajućoj sredini? Potrebno je

proanalizirati uticaj interakcije između sistemsko-softverske sredine i programa na ponašanje programa i odrediti faktore koji ih određuju. Upravo te osobine će odrediti potrebe za sredstvima opisa, izgradnje i modeliranja ponašanja programa.

Ponašanje distribuiranih aplikativnih programa karakteriše njegov nedeterminizam. Proces komunicira sa računarskom sredinom radi dobijanja pristupa resursima sistema. Komunikacija znači da on, preko odgovarajućih interfejsa, predaje sistemsko-softverskoj sredini podatke, koji određuju karakteristike upita i dobija od njih odgovor. Odgovor može biti, npr.: kôd uslova završetka zadovoljenja upita, a može biti blok podataka odnosno poruka od drugog procesa.

Nedeterminizam u ponašanju aplikativnih programa je pojava koju vidi spoljašnji posmatrač. Nešto se naziva posmatračem, ako to može, snimiti zahtjev procesa aplikativnog programa a takođe i odgovor sredine i reakciju programa na taj odgovor. Pri tome se ne ulazi u njegovu prirodu. Ako se na jedan isti zahtjev mogu javiti razni odgovori, pri čemu je reakcija programa na svaki od njih različita, tada se smatra da se program ponaša neodređeno ili nepredvidivo. Ovaj nedeterminizam se naziva spoljašnjim. Takođe se smatra da se program ponaša neodređeno, kada je reakcije programa različita, na jedan isti odgovor sredine. Pri tome posmatrač, tu reakciju unaprijed ne može da predskaže. Tada se govori o unutrašnjem nedeterminizmu.

Tako se dobijaju dva oblika nedeterminizma unutrašnji i spoljašnji. Izvorom spoljašnjeg nedeterminizma procesa javlja se nedeterminizam odgovora sredine na njegov zahtjev. Taj nedeterminizam uslovljavaju razni uzroci. Npr. tehničke performanse fizičke sredine, algoritmi arbitraže konflikata u sistemsko-softverskoj sredini, korišćenje neodređenih operatora u programskim jezicima. Na osnovu ovoga se može zaključiti da, u sredinama sa raznim tehničkim performansama i raznim algoritmima arbitraže, slijed obraćanja logičkim resursima u jednom istom programu, može biti različit. Ovo se može desiti i pri jednakim izlaznim podacima tj. jednakim rezultatima obrade procesa. Nedeterminizam toga tipa može nastati i zbog razlike u algoritmima distribucije i/ili raspodjele resursa. Tako npr, ako proces traži poruku i u sistemsko-softverskoj sredini za njega postoji nekoliko poruka, onda dalje ponašanje procesa može zavisiti od toga koju poruku od sredine on prvo dobija. A to može zavisiti, npr., od algoritma baferizacije poruka, korišćenog u sistemsko-softverskoj sredini, od strukture i tehničkih performansi komunikacione sredine.

Izvor unutrašnjeg nedeterminizma su podaci, koje proces dobija u formi poruka. Prirodno je da ponašanje procesa zavisi od ulaznih podataka. Međutim, dvije poruke koje posmatrač smatra jednakim, proces može razmatrati različito.

Analiza ponašanja programa pokazuje, da su osnovni faktori od kojih zavisi slijed obraćanja k logičkim resursima u toku konkretnog izvršavanja programa: izlazni podaci i međurezultati izračunavanja koje razmjenjuju procesi, slijed i brzina razvitka izračunavanja u procesima. Uticaj posljednja dva faktora izgleda ovako:

- ✓ U prvom slučaju proces mora znati, da su neki drugi procesi dostigli određene tačke u svojim izračunavanjima. Pri tome, za izbor daljeg hoda izračunavanja njemu nije važno u kakvom poretku su ti procesi dostigli te tačke.
- ✓ U drugom slučaju važan je ne samo fakt dostizanja tačaka, nego i slijed, u kom su ih procesi dostigli kritične tačke. U prvom slučaju se ponašanje programa naziva stacionarnim a u drugom nestacionarnim.

Među nestacionarnošću ponašanja i spoljašnjim nedeterminizmom postoji uzajamna veza, naime ako je program nestacionaran, onda je u njegovom ponašanju obavezno prisutan spoljašnji nedeterminizam. Slijedi, ako u ponašanju programa postoji samo unutrašnji nedeterminizam, onda njeno ponašanje zavisi od računarske sredine samo u dijelu tačnosti izračunavanja. Pri tom mi pretpostavljamo, da program ne skriva sinhronizaciju među procesima programa od posmatrača tih procesa. U opštem slučaju pod sinhronizacijom se podrazumijevaju ograničenja u poretku izvršavanja procesa programa. Ona zavisi npr.: od raspodjele resursa logičke sredine u fizičkoj sredini, strukture komunikacionih veza, tehničkih performansi računarskih konfiguracija i sl. U stvarnosti ova brzina zavisi od korišćenog oblika paralelizma. Analiza uticaja tog faktora na ponašanje programa biće predmet daljeg istraživanja. Može se zaključiti da, zavisno od toga u kom se stepenu može utvrditi nestacionarnost u ponašanju programu, zavisi i mogućnost utvrđivanja razlika između eksperimentalne i analizirajuće računarske sredine.

Mogu se rezimirati razmatranja osobenosti funkcionisanja distribuiranih računarskih sistema pomoću sljedeće klasifikacije ponašanja programa u noj:

- ✓ Nezavisno od podataka, npr.: furijeve transformacije, množenje matrica;
- ✓ Zavisno od podataka, npr. sortiranje;
- ✓ Stacionarno, ovome mogu služiti razni primjeri iz oblasti matematičke fizike, koje izvršavaju različite procedure obračuna nad velikim skupom podataka. Na svakom vremenskom koraku važno je da budu izvršene sve procedure a koja će biti izvršena ranije a koja kasnije nije važno;
- ✓ Nestacionarni, npr. protokoli lokalnih mreža.

5.5.3 Prognoza slijeda aktivnosti

Na osnovu gornje klasifikacije ponašanja programa i određivanja unutrašnjeg i spoljašnjeg nedeterminizma slijedi, da ako je ponašanje programa stacionarno to pri uslovima neizmijenjenosti izlaznih podataka i tačnosti izračunavanja njegovo ponašanje ne zavisi od računarske sredine. To znači, da za prognoziranje ponašanja programa, pri uslovima neizmijenjenosti skupa izlaznih podataka, dovoljno je fiksirati i izmjeriti parametre istorije programa u nekoj eksperimentalnoj sredini. U ma kojoj drugoj sredini ono ostaje isto takvo. U novoj sredini mijenja se samo brzina zadovoljenja zahtjeva za logičkim resursima.

Tim samim, nije nužno izgrađivati potpuni opis ponašanja programa u svim mogućim i nemogućim sredinama. Sistemsko-softverska sredina eksperimentalnog sistema mora da obezbijedi potreban skup logičkih resursa u neophodnom broju, makar na virtuelnom nivou. Ovo svojstvo programa sa stacionarnim ponašanjem bilo je iskorišćeno pri izgradnji sistema analize produktivnosti hijerarhijskih distribuiranih računarskih sistema [73].

Za nestacionarno ponašanje predlažu se dva načina prognoziranja aktivnosti: sa povratnom spregom i konačnim brojem reakcija. Prvi način je istraživan u [75]. Osnova ove ideje se sastoji u istovremenom izvršavanju distribuiranog programa u eksperimentalnoj sredini i njegovoj simulaciji na modelu analizirajućeg sistema. Svaki put, kada se u eksperimentalnom sistemu proces aplikativnog programa obraća sistemsko-softverskoj sredini za pristup ka dinamičkom logičkom resursu, karakteristika tog zahtjeva predaje se u model. Model određuje reakciju modelirajuće sistemsko-

softverske sredine. Na osnovu te reakcije određuje se adekvatna reakcija eksperimentalne sredine, koja se i predaje aplikativnom procesu koji je generisao zahtjev. Pod adekvatnošću, u datom slučaju, se podrazumijeva da je aplikativnom procesu obezbijeđen pristup zahtijevanom logičkom resursu i u zahtijevanom obimu. Ovaj način zahtijeva velike dodatne troškove i specijalni aparat podrške. Glavni njegov doprinos je u tome što on dozvoljava analiziranje funkcionisanja modelirajućeg sistema pod dejstvom bilo kog programa čak i sa nestacionarnim ponašanjem.

Način konačnog broja reakcija sastoji se u sljedećem. Neka je količina reakcija sistemsko-softverske sredine na zahtjeve programa, koji utiču na dalje njegovo ponašanje, mala i neka je unaprijed poznata. Pritom ponašanje programa poslije zahtijeva jednoznačno određuje reakcija logičke sredine. To svojstvo posjeduju, npr. protokoli u lokalnim mrežama. Pri izvršavanju programa u eksperimentalnoj sredini specijalno se odrađuju sve poznate reakcije sistemsko-softverske sredine na zahtjeve programa i mjeri se njeno dalje ponašanje. Zbirna informacija se koristi pri modeliranju na sledeći način: po reakciji modelirajuće sistemsko-softverske sredine određuju se karakteristike ponašanja programa. U suštini karakteristike najbližeg zahtjeva za logički resurs.

Za istraživanje uticaja različitih oblika paralelizma na ponašanje programa potreban je formalni model funkcionisanja distribuiranog računarskog sistema i njihovog programskog obezbeđenja. Eksperimentalna sredina mora imati isti stepen paralelizma kao i analizirajuća. U tom slučaju javlja se oštra zavisnost između mehanizma distribucije resursa u tim sredinama i niz drugih problema.

5.5.4 Mjera računarskog rada

Potrebno je definisati mjeru računarskog rada, koji obavlja fizička sredina pod dejstvom programa. Ova mjera se može izgraditi, ako se zahtjevi programa za obradu podataka mogu izraziti bez bilo kakvih označavanja kakvim sredstvima i sa kakvom brzinom te zahtjeve zadovoljava konkretna računarska sredina. Zajedno sa tim, potrebno je obezbijediti efikasnost i potrebnu tačnost preslikavanja opisa ponašanja programa u terminima zahtjeva u termine vremena i fizičke sredine, kada ta sredina bude konkretizovana.

Problemi izgradnje sistemski-nezavisne mjere računarskog rada predstavljaju fundamentalne probleme teorije računarskih sistema. Nju su razmatrali nekoliko istraživača. Najpre je ovaj problem nastao pri poređenju produktivnosti računarskih sistema. Ako se sa U označi obim računarskog rada, koji je neophodno ispuniti pri izvršavanju programa P. Ako je za izvršavanje programa na prvom računarskom sistemu bilo potrebno vrijeme T_1, a na drugom vrijeme T_2, onda je produktivnost prvog sistema jednaka U/T_1, a drugog U/T_2. Pri upoređivanju produktivnosti, uzima se odnos tih veličina. Skraćivanjem sa U ostaje samo odnos vremena. Na tom principu je i izgrađen način poređenja produktivnosti računarskih sistema pomoću etalona (engl. *Benchmark*).

Većina pokušaja da se utvrdi mjere računarskog rada svodila se na izgradnju neke kanonske realizacije algoritamskog procesa i na određivanje težine tog procesa kao nekog količinskog svojstva njegove kanonične realizacije. Kanonična realizacija, korišćena u dosadašnjoj teoriji računarskih sistema predstavljaju manje ili više apstraktne mašine, a merom računarskog rada javljaju se prostorne karakteristike

memorije (dužina trake mašine Tjuringa, broj korišćenih memorijskih lokacija, broj softverskih elemenata i sl.) i vremenskih karakteristika (broj operacija sabiranja, množenja, broj iterativnih ciklusa i sl.). Ovi se prilazi obično primjenjuju za izbor najefikasnijeg algoritma za realizaciju jedne iste funkcije. Najboljoj efikasnosti odgovara minimalni obim rada. Takva mjera se javlja sistemski nezavisnom, ne pokazujući suštinske pomoći pri praktičnoj ocjeni produktivnosti. Uzrok tome je pretežna ocjena samo na računarske aspekt (razmjena sa spoljašnjim uređajima, prenosi u memoriju i sl., na čemu se troši značajan dio rada sistema, koji se obično ne uračunava) i odsustvo efektivnih metoda prevoda ocjene obima rada u terminima zahtjeva na resurse zadatog sistema.

Drugi prilaz izgradnji mjere računarskog rada zasniva se na teoriji informacija. Helerman određuje obim rada procesa $f: X \to Y$, gdje je X konačna oblast određenja, koja može biti razbijena na n podoblasti X_i, koji se javljaju klasama ekvivalentnosti u odnosu na Y, kao veličinu

$$\omega(f) = \sum_{i=1}^{n} |X_i| \log_2 \left(|X| / |X_i| \right), \qquad (5.1)$$

gdje $|A|$ označava snagu skupa A.

Nije teško primijetiti, da je $\omega(f)/|X|$ entropija izvora X.

U drugom prilazu svi procesi dijele se na dva tipa: povratni i nepovratni. Povratni procesi mogu samo mijenjati poredak ulaznih elemenata, npr. kao kod sortiranja. Nepovratni mijenjaju ulazne elemente, tj. $|X| \neq |Y|$. Rad procesa f određuje se kao

$$\alpha(f) = \frac{1}{|X|} \left\{ \sum_{i=1}^{n} |X_i| \log_2 |X_i| \right\}, \qquad (5.2)$$

što odgovara gubitku informacije pri unosu, ako je f nepovratno, i gubitku poretka, ako je f povratno. U tom prilazu, za razliku od Helermana ne razlikuju se identična pretvaranja, tj. prenos podataka.

Izvore informacija i njihove entropije razmatrao je i Rozvadovski, kod koga se obim računarskog rada određuje kao sumarno smanjenje entropije svih informacionih izvora, koji obrazuju računarski proces. Npr. za proces, koji odgovara pojedinačnoj mašinskoj komandi dodjeljuju se tri informaciona izvora:

S_1 - bira tekuću i određuje sljedeću komandu,
S_2 - određuje vrijednost operanada,
S_3 - određuje vrijednost rezultata.

Tada se obim računarskog rada određuje kao:

$$U = H(S_1) + H(S_2) + H(S_3), \qquad (5.3)$$

gdje je $H(S_i)$ entropija izvora S_i.

Svi ovi prilazi su veoma apstraktni sa stanovišta ciljeva postavljenih u ovom radu. Za njih nije bio nađen nijedan prihvatljiv odraz na karakteristike tehničkih performansi sistema. Kod svih ovih prilaza korišćen je denotacioni prilaz za opis dinamike programa.

Predstavljanjem programa u formi djelimično-uređenog skupa obraćanja logičkim resursima ne zavisi od unutrašnjih karakteristika računarske sredine i u tom smislu ta forma može služiti kanoničnom predstavom realizacije algoritma. Dalje, program određuje samo slijed korišćenja logičkih resursa, koji obezbjeđuje računarska sredina. Obim posla ma koje aktivnosti računarskog sistema, uključujući i korišćeni logički

resurs, konačno je određen skupom elementarnih operacija ili komandi, koje treba da izvrši fizička sredina, da bi ispunila aktivnost. Taj skup komandi određuje linkovanje i realizacija logičkog resursa, koji zavise od kompajlera i sistemsko-softverske sredine. Fizička sredina odražava (preslikava) niz aktivnosti programa na osu vremena, dajući svakoj aktivnosti interval vremena potreban za njegovo izvršavanje. Ako se nađe metod koji će biti dovoljno efektivan u smislu vremenske složenosti i dovoljno tačan u smislu prognoziranja vremena i koji će održavati slijed obraćanja k logičkim resursima u terminima elementarnih operacija fizičke sredine, onda će biti postignuti ciljevi analize.

Neka je data istorija izvršavanja programa u obliku uređenog skupa obraćanja k takvim logičkim resursima, čija je realizacija niz komandi, koje izvršava jedan procesor. Ovi resursi su označeni kao sekvencijalni logički resursi.

Ovdje je pre svega od interesa analiza statičkih logičkih resursa. Dinamički logički resursi, saglasno opredeljenju pripadaju logičkoj sredini, koju po uslovu izvornog zadatka mi poznajemo samo toliko, da možemo izračunati vrijeme zadovoljenja zahtjeva za bilo koji dinamički logički resurs. Povezivanje obraćanja k dinamičkom logičkom resursu vrši se metodom interpretacije i ne zavisi od sadržaja obraćanja. Vrijeme zadovoljenja zahtjeva za korišćenjem logičkog resursa zavisi ne samo od realizacije tog resursa, nego i od kašnjenja sinhronizacije pri pojavi više obraćanja k njemu. Takođe utiče i arbitraža konflikta pri jednovremenom obraćanju k njemu. Ta vremenska kašnjenja se mogu izračunati pomoću tehnike imitacionog modeliranja, simulirajući ponašanje konkurentnih procesa u vremenu [74], [79], [80].

Složeniji je slučaj sa statičkim logičkim resursom, čije je privezivanje osnovna briga ovog poglavlja. U kontekstu razmatranog zadatka obraćanja k statističkom logičkom resursu je, u suštini, narudžba za izvršavanje operatora prisvajanja i izračunavanje izraza na njegovoj desnoj strani. Da li je taj izraz aritmetički, logički, indeksni i sl. sada nije važno.

Problem sada ima nekoliko aspekata. Prvi je optimizacija broja obraćanja k logičkom resursu. Može se podvrgnuti program na jeziku programiranja optimizaciji, kao rezultat čega su iz tijela ciklusa izneti neki operatori prisvajanja. Tim samim će biti skraćen broj obraćanja ka nadležnom logičkom resursu u ponašanju programa. Drugi aspekt je optimizacija samog obraćanja. Optimizacija, kako u prvom tako i u drugom slučaju ne zavisi od osobenosti fizičke sredine. Njeni metodi su dobro razvijeni i šire poznati.

Treći aspekt je zavisnost povezivanja statičkog logičkog resursa od karakteristika kompajlera, tačnije generatora koda korišćenog kompajlera. Posebna pažnja biće posvećena razmatranju ovog aspekta problema.

Neka je data istorija izvršavanja programa u obliku uređenog skupa obraćanja k logičkim resursima. Pretpostavlja se:

- da su svi logički resursi sekvencijalni,
- da se razmatra obraćanje samo ka statičkim resursima,
- da istorija izvršavanja programa ne sadrži beskorisna obraćanja logičkim resursima. (obraćanje k logičkim resursima biće beskorisno ako njegovo udaljenje iz istorije izvršavanja programa ne mijenja rezultate programa),
- da bilo koji izraz u istoriji izvršavanja programa ne sadrži opšte podizraze.

Izvršioca karakterišu:

- skup tipova komandi i vremena izvršavanja komandi svakog tipa,
- broj programski-dostupnih registara procesora (registra opšte namjene),
- kapacitet operativne memorije i njen adresni prostor,
- maksimalna veličina stek memorije.

Potrebno je odrediti vrijeme izvršenja niza zahtjeva za statičkim logičkim resursima u zadatoj istoriji izvršavanja programa.

Jednom tipu komandi pripadaju komande sa istim kodom operacije i vremenom izvršavanja. tako, npr. komande sa istim kodom operacije, ali raznim oblicima adresacije operanada, po pravilu, imaju razna vremena izvršavanja, i kako slijedi pripadaju raznim tipovima.

Poznato je da vrijeme izvršavanja komande ne zavisi samo od koda operacije i korišćenog načina adresacije nego i od vrijednosti operanada. Prilikom analize može se zanemariti uticaj vrijednosti operanada na vrijeme izvršavanja komandi. Sa druge strane mogu se razbiti svi skupovi vrijednosti operanada na klase ekvivalentnosti, pri čemu istoj klasi ekvivalentnosti pripadaju te vrijednosti operanada kod kojih se razlike vremena izvršavanja mogu zanemariti. Kada se konkretizuje pojam tipa komandi ostaje se u granicama razmatranja, ukoliko ono zavisi od potrebne tačnosti procjene.

Neka je a - aktivnost. Težinom aktivnosti a naziva se vektor

$$(n_1, n_2, \cdots, n_k) \qquad (5.4)$$

gdje k označava tip komande u sistemu komandi posmatranog procesora, a n_i je broj komandi i-tog tipa, korišćenih pri izvršavanju aktivnosti a.

Neka je Ω skup vektora oblika (5.4), $A(P)$ - skup aktivnosti u ponašanju programa P, a $\Omega(P)$ je skup vektora oblika (5.4), koji odgovaraju aktivnostima iz $A(P)$.

Težina aktivnosti može biti izražena na sljedeći način

$$\omega : A(P) \to \Omega \qquad (5.5)$$

pri čemu se pretpostavlja da je $\omega(\varnothing) = (0, 0, \cdots, 0)$, tj. odsustvo aktivnosti odgovara nul-vektoru.

Neka su a_1, a_2, ..., a_j aktivnosti i neka je $\prec$ operacija sekvencijalnog redanja aktivnosti. Tada je $a_1 \prec a_2 \prec \ldots \prec a_j$ sastavna aktivnost, koju obrazuje sekvencijalno izvršavanje ukazanih aktivnosti u ukazanom poretku jedne te iste fizičke sredine aktivnosti. Funkcija $\omega(\mathbf{a})$ je aditivna, tj.

$$\omega(a_1 \prec a_2 \prec \cdots \prec a_q) = \sum_{i=1}^{q} \omega(a_i). \qquad (5.6)$$

Ako preslikavanje $\omega(a)$ nije jednoznačno, onda taj uslov označava sljedeće. Među vrijednostima $\omega(a_i)$ za svako $a_i \in \{a_1, a_2, \cdots a_q\}$ uvijek je moguće naći takvo, da uslov bude ispunjen.

Bitan doprinos ovakvog određivanja težine izračunavanja je što ono daje efektivno izračunavanje vremena izvršavanja aktivnosti a. Naime neka je $p_t = (t_1, \cdots, t_k)$ vektor tehničke produktivnosti procesora, tj. elementi toga vektora su vremena izvršavanja komandi sa uzimanjem u obzir korišćenog načina adresacije operanada.

Tada se vrijeme izvršavanja aktivnosti a, uređajem sa tehničkom produktivnošću p_t može definisati kao funkcija

$$T(a) = (\omega(a), p_t) = \sum_{i=1}^{q} n_i t_i. \qquad (5.7)$$

Može se pokazati, da ova funkcija posjeduje sljedeća svojstva [79]:

1. $\forall(a_1, a_2) : T(a_1) \le T(a_2)$, ili $T(a_1) \ge T(a_2)$, tj. aktivnosti se mogu upoređivati.
2. $T(\varnothing) = 0$.
3. $\forall a : a \neq \varnothing \Rightarrow T(a) > 0$.
4. $\forall(a_1, a_2) : T(a_1 \prec a_2) = T(a_1) + T(a_2)$.
5. $T(a)$ je norma u vektorskom prostoru Ω obrazovanom sa $\omega(a)$ pri svim mogućim a.

Gore predstavljena razmatranja dokazuju tačnost tvrdnje da ako su zadati tehnička produktivnost izvršioca p_t, istorija izvršavanja programa $H(P)$ i jednoznačna preslikavanja $\omega(a_i)$, i=1 ,..., q, pri čemu je zadovoljena relacija (5.6) i u sredini izvršavanja aktivnosti a_i nema dinamičkih logičkih resursa, onda se za ma koji niz aktivnosti iz $H(P)$ može odrediti vrijeme njenog izvršavanja uz pomoć relacije (5.7).

U praksi uslov jednoznačnosti $\omega(a_i)$ nije ispunjen. Suština se sastoji u tome, da se pri statičkom povezivanju preslikavanje ω jedne iste aktivnosti a_i može predstaviti sa različitim vektorima $\{(n_1, \cdots, n_k)_j\}$. Generatori koda raznih kompajlera mogu izgrađivati razne težinske vektore za jednu istu aktivnost a_i.

Pri izgradnji koda za izračunavanje izraza osnovni problemi koji se mogu desiti su: koje se komande koriste, u kojem poretku i gdje se smještaju međurezultati. Količina različitih kodova, kojima se može izračunati isti izraz je beskonačna. Čak ako se ograničimo samo na "razumne" kodove njihov broj raste eksponencijalno sa rastom dužine izraza. Isto tako je važno da se pri izgradnji generatora koda, teži izabrati algoritam, koji obezbjeđuje izgradnju optimalnog koda. Kriterijum optimizacije, po pravilu, orijentisan je tako, da je on bilo direktno bilo indirektno, usmjeren na generisanje koda sa minimalnim vremenom izvršavanja.

Da bi odstranili zavisnost funkcije $\omega(a_i)$ od osobenosti generatora koda konkretnog kompajlera potrebno je izgraditi jednoznačno preslikavanje

$$\omega_{opt} : \{a_i \mid a_i \in A(P)\} \to \{(n_1, \cdots, n_k)\} \qquad (5.8)$$

optimalno za dati set komandi procesora C_{cpu}, tj.:

$$\forall \omega : \omega_{opt}, \forall a_i : a_i \in A(P) \Rightarrow (\omega_{opt}(a_i), p_t(C_{cpu})) \le (\omega(a_i), p_t(C_{cpu})) \qquad (5.9)$$

gdje su $C_{cpu} = \{(k_i, t_i)\}$ karakteristike izvršioca, odnosno procesora, pri čemu je k_i tip komande a t_i vrijeme njenog izvršavanja.

Zadatak traženja ω_{opt}, kako je formulisan gore, mora se rješavati ponovo za svaki skup komandi C_{cpu}. Uzrok tome leži u činjenici da se kao kriterijum optimizacije koristi vrijeme. Takav kriterijum je uvijek sistemski zavistan, tako da se zadatak optimizacije mora rješavati svaki put ponovo. Koristeći osobinu razmatrane klase izvršioca (procesora) po kojemu su, dužina koda i vrijeme njegovog izvršavanja direktno proporcionalni. Ovo je pokazano u radovima u kojima preovladava upravo ovaj

kriterijum optimizacije. Kao posledica ovog može se uzeti za kriterijum optimizacije dužina koda tj.

$$l(\omega(a_i)) = \sum_i n_i . \qquad (5.10)$$

Sada se može formulisati zadatak optimizacije na sledeći način: dati su istorija izvršavanja programa i karakteristike izvršioca $C_{cpu} = \{(k_i, t_i)\}$, treba naći jednoznačno preslikavanje $\omega_{opt}(a_i)$ takvo da je zadovoljen sledeći uslov

$$\forall \omega : \omega \neq \omega_{opt}, \forall a_i : a_i \in A(P) \Rightarrow l(\omega_{opt}(a_i)) \leq l(\omega(a_i)). \qquad (5.11)$$

Ideja predloženog rešenja sastoji se u izgradnji modela računara koji će odražavati karakteristike realnih računara i postojanje optimalnosti koda. Novina ove ideje se sastoji u primjeni poznate tehnike za postizanje novih ciljeva. Izgraditi algoritam, koji će generisati optimalni kod za izgrađeni model, tj. treba zadati preslikavanje sistema komandi modela u sistem komandi realnog izvršioca. Drugim rečima sistem komandi realnog izvršioca je parametar generatora koda.

Opredeljujući uticaj na efikasnost generisanog koda imaju takve karakteristike izvršioca, kao što su organizacija i korišćenje registarske i operativne memorije, fleksibilnost i raznovrsnost načina adresacije. Kod pomenute klase izvršioca po pravilu koristi se direktni, registarski, indeksni, implicitni i načini adresacije koji su orjentisani na rad sa stekom.

Problem generacije optimalnog koda, čak i kod programa sastavljenih iz čisto linijskih programskih struktura, npr. program koji se sastoji od niza operatora prisvajanja na mašini sa jednim registrom je veoma složen [86], [87], [88]. Međutim, ako se ograniči posmatranje programa bez opštih podizraza, problem će biti bitno uprošćen. To se postiže simetričnim operacijama registar-registar, bez složenih načina adresiranja. U literaturnim izvorima: [89], [90], [91] dokazana je optimalnost algoritma generacije koda za slučaj izraza drvenaste strukture pri raznim kriterijumima optimizacije, uključujući i kriterijum dužine generisanog koda.

Poseban doprinos u izgradnji algoritama za generaciju optimalnog koda ima rad [93], gdje je u klasi algoritama dinamičkog programiranja bio izgrađen algoritam linearne vremenske složenosti, koji generiše optimalni kod. Ovaj kod je optimiziran za registarski model izvršioca, kod koga je moguće i indirektno adresiranje. Ovaj algoritam generiše optimalni kod izraza, zadatog u obliku drveta T, povezujući optimalne nizove komandi poddrveća iz T.

U radu [89] određeni su neophodni i dovoljni uslovi optimalnosti koda, koji izračunava vrijednost izraza po drvetu T, koji predstavlja taj izraz. U tom radu je takođe dokazano, da svaki kod koji zadovoljava ove uslove se može predstaviti u normalnoj formi. Uslovi optimalnosti i pojmovi uvedeni u ovom radu su slični pojmovima i uslovima optimalnosti distribuirane memorije, detaljno opisanim u [101].

Kod ovog prilaza drvo T se posmatra odozgo-naniže i dijeli na nizove poddrveća. Ova poddrveća, sa izuzetkom poslednjeg, predstavljaju podizraze iz T. Pri optimalnom izračunavanju T, ove podizraze treba izračunavati tako, da se sačuvaju njihove vrijednosti u operativnoj memoriji. Na kraju se izgrađuje odgovarajući kod programa za izračunavanje T.

Prethodno opisani modeli izvršioca ne obuhvataju stek izvršioce, kod njih nema indeksne adresacije, što usložnjava modeliranje rada sa strukturama. Ovu prazninu su u određenom smislu ispunili rezultati istraživanja objavljeni u publikaciji [92]. Tamo su bili dobijeni rezultati analogni kao u [77], ali primjenjeno kod stek izvršioca sa stekom ograničene dubine. Kriterijum optimizacije u ova dva rada je isti, to je dužina koda. U [92] je pokazano da kod stek izvršioca ovaj kriterijum optimizacije je jednak minimizaciji broja operacija istovara steka u memoriju.

Na osnovu gore ukazanih rezultata bio je izgraden model, koji je dobio naziv univerzalni model izvršioca. Njegov detaljni opis je dat u knjizi [94]. Osnovne karakteristike mašine koja se modelira ovim modelom su:

- ✓ S jednakih redno adresiranih ćelija memorije,
- ✓ N registara opšte namjene,
- ✓ M ćelija memorije,određenih za stek.

Stekova može biti nekoliko. Svaki stek je snabdeven pokazivačem, koji može biti smješten u registru. Ukupna dužina steka ne prevazilazi M. U mašini se koristi tri načina adresacije: registarska, direktna i indeksna sa korišćenjem ne više od p indeksnih registara.

Univerzalni model izvršioca izvršava sledeće operacije:

- $E \to r$,
- $r \to m$,
- $Push(m,r): r+1 \to r, \quad m \to M(r)$,
- $Pop(m,r): M(r) \to m, \quad r-1 \to r$,
- $Op(\varphi,r): r-p+1 \to r, \quad \varphi(M[r],\ldots,M[r+p-1]) \to M[r]$.

ovdje je: $r \in N$, $m \in S$, $M[r] \in M$, E je izraz sa strukturom drveta koji sadrži operacije iz konačnog skupa θ, a φ je p-narna operacija iz θ.

Algoritam generacije koda dobija na ulaz program, u obliku niza izraza u obliku drveta. Svako drvo, u zavisnosti od tipa operanda, ima i raskrsnicu odlučivanja režima (stek ili registarski) i specijalnu labelu koja pokazuje početak ili kraj ciklusa. U zavisnosti od ukazanog režima generiše se kod. Iz algebarske osobine operacije u izrazu se uzima u stek režimu i komutativnost i asocijativnost, a u registarskom samo komutativnost. Kao ulazni parametri algoritam dobija osnovne parametre modela: broj ćelija operativne memorije S, broj registara N, dubina steka M, maksimalno mogući broj indeks registara u komandi p, dozvola ili zabrana korišćenja neposrednih operanada u komandi. Za testiranje algoritma generacije koda za model univerzalnog izvršioca bilo je razmotreno nekoliko programa.

Za svaki program bili su dobijeni kodovi modela univerzalnog izvršioca pri različitim ulaznim parametrima, koji su se zatim prevodili u asembler odgovarajućeg računara. Dobijeno vrijeme izvršavanja programa upoređivalo se sa vremenom, koje se dobija pri korišćenju kompilatora. Pri tome razlika u vremenu izvršavanja programa, izračunatog pomoću algoritma, i vremena izvršavanja programa, dobijenog uz pomoć kompajlera nije prevazilazio ±3%, što je zadovoljavajuća procjena tačnosti prognoziranja vremena izračunavanja.

5.6 Sredstva opisa ponašanja programa

Prve primjene operacionog prilaza za opis ponašanja programa bili su u terminima regularnih izraza. Ova primjena je korišćena za analizu performansi i pripremu radnog opterećenja [29], [30], [32], [68].

Jednim od prvih primjena operacionog prilaza za opis ponašanja programa javljaju se E-izrazi, opisani u [74]. Kod ove metode opisa ponašanja programa koristi se metodaređanja, koja pretpostavlja postojanje centralnog upravljanja i jedinstvenog mjerenja vremena. Ove pretpostavke, u opštem slučaju, nisu tačne i ne mogu se primijeniti kod distribuiranih računarskih sistema.

Drugim slabim mjestom regularnih izraza u postojećem obliku javljaju se nedovoljnost i neadekvatnost sredstava za opisivanje nedeterminizma u ponašanju programa. U suštini, jedinstvenim odgovarajućim sredstvom javlja se operacija izbora. Medutim, u postojećem obliku ni ona ne dozvoljava da se razdvoje unutrašnji i spoljašnji nedeterminizam u ponašanju distribuiranih programa.

5.7 Zaključci

Postojeća sredstva posmatranja i mjerenja karakteristika funkcionisanja aplikativnih programa:

- ne dozvoljavaju rad sa distribuiranim događajima,
- sva ova sredstva i metode su orijentisane na fiksni skup događaja,
- poznati tehnički načini mjerenja karakteristika događaja nijesu adekvatni svojstvima ponašanja razmatrane klase programa.

Analizom sredstava modeliranja distribuiranih računarskih sistema sa tačke gledišta tipičnih ciljeva istraživanja i ocjene njihove funkcionalnosti pokazao je, da izabrano eksperimentalno sredstvo mora da obezbijedi kako količinsku, tako i algoritamsku analizu distribuiranih računarskih sistema.

Definisanjem nezavisne mjere ponašanja programa omogućeno je rješavanje tri zadatka:

- izbor aktivnosti za opis ponašanja programa,
- prognoziranje ponašanja u novoj sredini i
- procjena vremena aktivnosti.

Za rješenje prvog zadatka uvodi se pojam logičkog resursa kao osnovnog sredstva obrade podataka i upravljanja izračunavanjima. Upravo te njegove osobine obezbjeđuju to što niz korišćenja logičkih resursa zatvara međusobne veze po informaciji i upravljanju u programu. Važno je i to, da nivo detaljizacije pojma logičkog resursa nije fiksiran i dozvoljava razmatranje ponašanja programa sa raznim stepenima detaljizacije.

Predloženo rješenje drugog zadatka zasniva se na klasifikaciji ponašanja programa. Slijedi da je ta klasifikacija bila izgrađena na detaljnoj analizi pojma nedeterminizma i posljedica koje on izaziva. Takođe je bilo pokazano da za rješenje izlaznog zadatka nije potrebno imati nezavisno ponašanje cijelog programa, dovoljno je imati nezavisno ponašanje samo određenog njenog dijela. Međutim, van granica sprovedenog istraživanja ostao je uticaj na ponašanje programa različitih oblika

paralelizma. Takvo istraživanje zahtijeva izgradnju modela samog programa, njegovog izvršioca, formalizaciju oblika paralelizma i određivanje uslova pod kojim u zavisnosti od korišćenog oblika paralelizma ponašanje programa je neizmijenjeno.

Određivanje mjere računarskog rada ne bi trebala da zavisi od karakteristika kompajlera konkretne računarske sredine ili bilo kakvog sredstva pripreme programa za izvršavanje. Ovo rješenje se može dobiti u klasi algoritama dinamičkog programiranja, kada se kao kriterijum optimizacije uzima aditivna funkcija od količinskih karakteristika generisanog koda. Ovo je moguće pri sljedećim uslovima:

- izrazi nemaju opštih podizraza,
- u nizu obraćanja logičkim resursima nema nekorisnih obraćanja i
- pri generaciji podrazumijevaju se komutativnost i asocijativnost operacija, koji se susreću u izrazima.

Iz predhodnih razmatranja a na osnovu rezultata istraživanja karakteristika ponašanja distribuiranih aplikativnih programa i funkcionisanja distribuiranih računarskih sistema može se zaključiti da se može izgraditi matematički model funkcionisanja distribuiranog sistema koji će odražavati osobenosti interakcije aplikativnog programa sa sistemsko-softverskom i fizičkom sredinom, hijerarhijsku prirodu i distribuiranost ovih sredina. Pri tom je važno napomenuti, da za rješavanje zadatka analize produktivnosti distribuiranog računarskog sistema nije nužno izgrađivati cijelu invarijantu ponašanja programa. Izabrani put rešenja dozvoljava da se u datoj postavci tog zadatka ograniči na određeni dio invarijante ponašanja programa. Ne obazirući se na to, ***ovdje je bio cilj da se pokaže, da se uz izvesna ograničenja koja su gore navedena, može govoriti o postojanju sistemski-nezavisne mjere računarskog rada ili invarijante ponašanja programa***.

GLAVA

6

6. MATEMATIČKI MODEL FUNKCIONISANJA DISTRIBUIRANOG RAČUNARSKOG SISTEMA

6.1 Uvod

6.2 Osnovne pretpostavke izgradnje matematičkog modela

6.3 Glavne komponente matematičkog modela

6.4 Opis matematičkog modela

6.5 Zaključci

6.1 Uvod

Sada je potrebno definisati osnovna pitanja modeliranja distribuiranih računarskih sistema i zahtjeve za sredstvima modeliranja, koja bi mogla zadovoljiti uslove zadatka formulisanog u poglavlju 3.2. Pri tome veliki značaj ima izbor matematičkog modela funkcionisanja distribuiranog računarskog sistema. On zavisi od cilja modeliranja i osobina objekta modeliranja koji je od interesa za analizu. Model je manje-više apstraktna, uprošćena predstava razmatrane pojave tj. spoljašnji odraz suštine predmeta i procesa sa ciljem prognoziranja njihovih svojstava. U datom slučaju pod pojavom se podrazumijeva proces korišćenja resursa logičke i fizičke sredine distribuiranog sistema od strane distribuiranog aplikativnog programa.

Pod *matematičkim modelom* distribuiranog sistema, podrazumijeva se sistem matematičkih pojmova i relacija među njima, koji adekvatno opisuju karakteristike funkcionisanja distribuiranog sistema. Kako je to već naglašeno u poglavlju 3.4, izgradnja matematičkog modela distribuiranog sistema, predstavlja postupak izražavanja pokazatelja performansi sistema u zavisnosti od njegovih unutrašnjih parametara. Time se analiza realnog distribuiranog sistema, zamjenjuje se ispitivanjem njegovog matematičkog modela. Istraživanje distribuiranog računarskog sistema se svodi na rješavanje konačnog broja logičkih i računskih operacija nad brojevima uz pomoć odgovarajućih numeričkih metoda.

Prednost izgradnje matematičkog modela sastoji se u tome što on dozvoljava da se relativno brzo, uz minimalne troškove, dobiju vrijednosti pokazatelja performansi distribuiranog računarskog sistema u širokom dijapazonu izmijene parametara sistema i parametara radnog opterećenja.

6.2 Osnovne pretpostavke izgradnje matematičkog modela

Osnovne osobine koje karakterišu posmatranu klasu distribuiranih računarskih sistema, a koje su bitne za izgradnju modela, date su u prethodnim poglavljima. Ovdje će one biti ponovljene, uz izvjesna dodatna ograničenja koja prate analizu i koja neće bitnije umanjiti univerzalnost niti tačnost metode. Osnovne pretpostavke za izgradnju ovdje predloženog modela distribuiranog računarskog sistema, sastoje su sljedećem:

1. Upravljanje u sistemu je decentralizovano. To znači da je svaki računar u potpunosti autonoman i radi pod svojim operativnim sistemom;
2. Sistem karakteriše odsustvo jedinstvenog vremena, tj. svaki računar posjeduje svoj sistem računanja vremena. S tim u vezi, uvodi se pojam logičkog vremena, odnosno mora se voditi računa o sinhronizaciji aplikativnih procesa;
3. Sistem sadrži djeljive resurse, kako hardverske tako i softverske odnosno logičke, pa u interakciji između distribuiranih aplikativnih procesa postoje konflikti koji se moraju rješavati na zadovoljavajući način, a u cilju obezbjeđenja korektnosti distribuiranih programa;

4. Autonomni računari su sa jednim centralnim procesorom ili se posmatraju kao jednoprocesorski;
5. Memorija u sistemu je distribuirana tj. razdijeljena je po autonomnim računarima i ne postoji jedinstvene memorije u sistemu;
6. Posmatrani sistemi ne rade u realnom vremenu, odnosno ponašanja programa ne zavisi od vremena;
7. U sistemu je prisutan unutrašnji i spoljašnji nedeterminizam u ponašanju programa.
8. Konkurentne procese u sistemu karakteriše postojanje dva oblika paralelizma;

Funkcionisanje distribuiranog računarskog sistema određuje uzajamno djelovanje procesa aplikativnih programa, procesa sistemsko-softverske strukture i hardvera. Ovdje se pod aplikativnim programima podrazumijevaju distribuirani aplikativni programi, koji su već opisani u poglavlju 2.9 i čiji se procesi odvijaju paralelno na odvojenim autonomnim računarima. Uz izvesne aproksimacije, a u cilju pojednostavljenja analize, dinamika izvršavanja distribuiranog aplikativnog programa, se posmatra kao skup *elementarnih procesa*. Nivo elementarnih procesa određen je nivoom detaljizacije modela. Pretpostavlja se, da se elementarni procesi ne mogu prekidati i da se izvršavaju samo sekvencijalno, tj. na jednom istom hardverskom izvršiocu. Pod hardverskim izvršiocem može se posmatrati: procesor, kontroler, mrežni adaptere ili računar u cjelini. Dok se ne završi jedan elementarni proces drugi ne može započeti. Isto tako praktično nema elementarnih procesa čije izvršavanje protiče nezavisno na jednom istom hardverskom izvršiocu. Izvršavanje jednog može biti zaustavljeno ili usporeno izvršavanjem drugog. Kao što je već rečeno, to je jedna od formi paralelizma. Ona se sastoji u tom da se na vremenskoj osi izvršioca, događaji koji odgovaraju jednom procesu mogu smjenjivati i ređati u proizvoljnom poretku sa događajima drugog procesa.

Karakteristike ponašanja programa u distribuiranoj sredini, već su razmatrane u prethodnim poglavljima, ovdje treba još napomenuti da je jedna od bitnih osobina njihovog ponašanja pojava dva oblika neodređenosti tog ponašanja, odnosno dva oblika nedeterminizma: unutrašnjeg i spoljašnjeg.

6.3 Glavne komponente matematičkog modela

Uzajamno djelovanje aplikativnih procesa, sistemskih procesa i hardvera može se šematski opisati na sljedeći način. Svaki aplikativni proces, koji se izvršava na određenom autonomnom računaru u sklopu distribuiranog programa, određuje logički niz akcija. Ove akcije se sastoje u obraćanju logičkim resursima u čijem se okruženju izvršava aplikativni proces, a preko njih hardveru, tačnije hardverskim resursima. Aplikativni procesi, za obavljanje svojih aktivnosti, koriste obe ove klase resursa.

Kao što je već navedeno u poglavlju 5.5.1, rutine odnosno procesi operativnog sistema, koji stoje na usluzi aplikativnim procesima, mogu se posmatrati kao dinamički logički resursi. Statički logički resursi odražavaju direktno obraćanje aplikativnog procesa hardverskim resursima distribuiranog sistema, zato se za njihovo obilježavanje najčešće koristi hardverska terminologija. S druge strane, i sami procesi operativnog

sistema se obraćaju hardveru računara radi zadovoljenja zahtjeva aplikativnog procesa i obezbjeđenja pravilnosti rada čitavog sistema. Sa stanovišta istraživača svaki dinamički resurs je softverski proces, pa samim tim i on ima određeno ponašanjem analogno ponašanju aplikativnog procesa, što se i te kako mora uzeti u obzir pri izgradnji matematičkog modela.

Na osnovu prethodnog izlaganja, osnovu matematičkog modela treba da čine sljedeće komponente:

a) *elementarni aplikativni procesi*, koji karakterišu ponašanje distribuiranog aplikativnog programa;
b) *logički resursi*, u čijem se okruženju izvršava aplikativni proces i kome se isti obraća radi zadovoljenja svojih zahtjeva;
c) *izvršioci*, koji predstavljaju fizičku sredinu posmatranog distribuiranog sistema, odnosno hardverske resurse na kojima se izvršavaju aplikativni i sistemski procesi;
d) *posmatrač*, odnosno instrument evidencije ili monitor.

Logički resursi, pomažu aplikativnim procesima u njihovom izvršavanju i čine neku vrstu interfejsa između njih i izvršioca odnosno hardverske strukture distribuiranog računarskog sistema. Sa stanovišta potrebnog nivoa tačnosti procjene performansi sistema i tipa analize tj. ugla posmatranja, izvršioci se dijele na sekvencijalne i distribuirane. Sekvencijalni izvršilac u jedno isto vrijeme izvršava aktivnosti zahtijevane samo jednim sekvencijalnim procesom. Pod sekvencijalnim izvršiocem može se npr. posmatrati autonomni računar, procesor autonomnog računara, mrežni adapter i tome slično. Jedan sekvencijalni izvršilac može u režimu podjele vremena tj. kvazi-paralelno, opsluživati nekoliko sekvencijalnih procesa. Sekvencijalni izvršioci su povezani međusobno komunikacionim kanalima, odnosno računarskom mrežom. Taj skup obrazuje distribuiranog izvršioca. Preko komunikacionih kanala procesi razdijeljeni po raznim izvršiocima mogu razmjenjivati poruke. Komunikacioni kanali ne predstavljaju izvršioce jer oni ne mogu raditi po sopstvenom programu. Komunikacioni kanali mogu samo propuštati poruke od jednog izvršioca do drugog bez bilo kakvih izmijena. Pri tome se unosi izvjesno kašnjenje između momenta njihove otpreme i momenta njihovog prijema. Veličina ovog kašnjenja predstavlja postojanu karakteristiku datog kanala.

Svaki sekvencijalni izvršilac je povezan sa određenim skupom drugih sekvencijalnih izvršioca. Njihov skup je ograničen, fiksan i ne mijenja se tokom rada aplikativnog programa. Takođe treba imati u vidu da je svaki sekvencijalni proces programa povezan sa određenim drugim procesima. Njihov skup je takođe fiksan i ne mijenja se sa vremenom, tokom jedne seanse eksperimenata.

Uopšteno gledano, na jednom izvršiocu procesi se mogu izvršavati kako sekvencijalno tako i paralelno. Suština je u tome da se termin paralelizma posmatra sa dva stanovišta tj. procesi se mogu odvijati istovremeno uz međusobnu kooperaciju ili nezavisno jedan od drugog. U drugom slučaju, sa stanovišta analize, nije važno odvijaju li se procesi u isto vrijeme ili ne.

Obraćanje elementarnih aplikativnih procesa, logičkim resursima najčešće se ostvaruje prenosom poruka odgovarajućeg tipa. One sadrže parametre, kojima se aplikativni proces, obraća logičkim resursima, odnosno procesima operativnog sistema. Tip poruke je standardizovan i pripada nekom određenom skupu dozvoljenih poruka. Pri

izgradnji modela, teži se da, prenos poruka i njihovo upravljanje bude u nadležnosti onog dijela modela koji je označen kao posmatrač. On je nezavistan od aplikativnog procesa i izvršioca i nalazi se van njih. Na nivou posmatrača i u odnosu na njega, određuje se i stepen transparentnosti i detaljizacije procesa, ekvivalentnost procesa, tipovi poruka i sl.

Prenos upravljanja, između dijelova distribuiranog programa tj. između aplikativnih procesa, može nastati na inicijativu samog procesa, a može nastati kao rezultat spoljašnjeg uticaja (npr. prekida). Na primjer od strane drugog izvršioca pod uticajem nekog procesa koji se na njemu odvija. Jednom izvršiocu može biti predstavljeno nekoliko procesa. Njihova količina je ograničena veličinom baferskog prostora izvršioca.

Prenos upravljanja od procesa "A" ka procesu "B" označava da resurse izvršioca i logičke resurse počinje koristiti proces "B". Izvršilac izvršava unutrašnje aktivnosti, koje su određene programom koji rađa dati proces. Te aktivnosti su nevidljive za posmatrača. Na vremenskoj osi izvršioca označavaju se događaji koji odgovaraju aktivnostima procesa "B". Granični nivo detaljizacije procesa za posmatrača su elementarni procesi, tj. neke uslovno nedjeljive elementarne aktivnosti, koje može izvršavati izvršilac. Primjer elementarnog procesa mogu biti komande mikroprocesora, pri čemu se uzima u obzir da je izvršilac sam taj mikroprocesor. Rezultatom izvršavanja ovih aktivnosti sa tačke gledišta posmatrača javlja se poruka određenog tipa i identifikacija procesa kojem je posmatrač dužan predati upravljanje i tu poruku.

Na ovaj način se, uz pomoć objekata kao što su: elementarni procesi, logički resursi, izvršioci i posmatrač, na formalnom nivou, izgrađuje matematički model posmatrane klase distribuiranog računarskog sistema, koji će detaljnije biti opisan u daljem tekstu.

6.4 Opis matematičkog modela

Ako se sa P_{uk} označi konačan skup svih procesa, koji su u trenutku posmatranja aktivni u sistemu, sa P_s skup aktivnih sistemskih procesa, a sa P_a skup procesa distribuiranog aplikativnog programa. Tada se pretpostavlja da važe sljedeće relacije :

$$P_{uk} = P_s \cup P_a \quad \text{i} \quad P_s \cap P_a = 0 \qquad (6.1)$$

6.4.1 Pojam koraka

Ponašanje procesa se može opisati kao struktura sastavljena od mnoštva ***koraka***. Korak (engl. *step*) čine tri pojma: *pobuda, reakcija i kompleksnost.* Pobuda je dobijanje poruke i prijenosa upravljanja od drugog procesa. Primljena poruka je tipa α. Reakcija je slanje poruke tipa β i prijenosa upravljanja procesu $p \in P_{uk}$. Kompleksnost je niz unutrašnjih aktivnosti koraka $q_1, q_2, \ldots q_n$ čiji se skup označava sa Q. Tada se sa Q^* označava skup svih mogućih lanaca konačne dužine sastavljenih uz pomoć simbola $q_i \in Q^*$. Pod lancem se podrazumijeva niz grana, u grafu bez petlji, koji se nadovezuju jedna na drugu bez obzira na njihovu orijentaciju.

Korak s se sada može predstaviti izrazom oblika :

$$\alpha \xrightarrow{q} \beta \circ p \qquad (6.2)$$

Veličina q u izrazu (6.2) je elemenat skupa Q^* i karakteriše kompleksnost koraka, a veličine α, i β predstavljaju poruke i elementi su konačnog skupa poruka M.

Tako se sada, uz pomoć dekartovog proizvoda, može predstaviti skup koraka procesa p, kao:

$$S(p) = M \times Q^* \times (P_{uk} \times M). \qquad (6.3)$$

U gornjem izrazu P_{uk} je skup svih aplikativnih i sistemskih procesa, koji se izvršavaju paralelno tj. istovremeno sa procesom p.

Svaki korak s raspolaže sljedećim osobinama:

- uticaj (engl. *influence*) $I(s)$,
- reakcija (engl. *reply*) $R(s)$,
- unutrašnje aktivnosti (engl. *action*) $A(s)$.

Dva koraka su jednaka ako su im uticaji i reakcije jednaki tj.:

$$s_1 = s_2, \text{ ako važi da je } U(s_1) = U(s_2) \text{ i } R(s_1) = R(s_2) \qquad (6.4)$$

Kompleksnost koraka je veličina koja dozvoljava da se odredi vrijeme izvršavanja unutrašnjih aktivnosti koraka. Te aktivnosti se odnose na korišćenje hardverskih i logičkih resursa. Kompleksnosti se na vremenskoj osi prikazuje jednoznačnom funkcijom koja se naziva funkcija kompleksnosti koraka:

$$f: Q^* \to \{(n_1, n_2, \ldots n_k)_j\}. \qquad (6.5)$$

Vektori $(n_1, n_2, \cdots, n_k)$, u gornjem izrazu su težinski vektori, a n_i su elementarne aktivnosti izvršioca, čija su vremena izvršavanja poznata i mogu se dobiti iz dokumentacije o tehničkim performansama izvršioca ili njihovim direktnim mjerenjem. Funkcija kompleksnosti koraka posjeduje sljedeće osobine:

$$f(q_1, q_2, \ldots, q_n) = \sum_{i=1}^{n} f(q_i), \qquad (6.6)$$

pri čemu se pretpostavlja da je $f(\varnothing) = (0, 0, \cdots, 0)$, tj. odsustvo unutrašnjih aktivnosti odgovara nul-vektoru.

Vrijeme prenosa poruke između dva izvršioca procjenjuje se uz pomoć dužine poruke (engl. *size*). Svaki tip poruke u sistemu ima tačno određenu dužinu. Dužina poruke se izražava u jedinicama podataka (bit, bajt, karakter) i može se definisati kao cjelobrojna funkcija, koja zavisi od tipa poruke tj.

$$\Lambda(x) = n, \text{ gdje je: } x \in M, \text{ a } n \text{ je prirodan broj.} \qquad (6.7)$$

Istorijom procesa p naziva se neprazan, konačan niz koraka toga procesa koji je zatvoren sa lijeva. To znači da ako je $h(p) = \{s_1, s_2, \ldots, s_k\}$ istorija procesa p, onda je za $\forall i : i \le k$ i $h'(p) = \{s_1, s_2, \ldots, s_i\}$ takođe istorija procesa p.

Broj koraka istorije $h(p)$ naziva se dužinom istorije i označava se sa $|h(p)|$. Za dvije istorije $h_1(p)$ i $h_2(p)$ kažemo da su jednake ako su im dužine jednake i svi odgovarajući koraci jednaki tj. ako je $|h_1(p)| = |h_2(p)|$ i za $\forall i : 1 \le i \le |h(p)|$ važi da je $s_{i_1} = s_{i_2}$.

6.4.2 Ponašanje procesa

Ponašanje procesa p opisuje skup $H(p)$ svih mogućih istorija toga procesa. U cilju ispitivanja osobina kojim raspolaže struktura $H(p)$ nad skupom $S^*(p)$ koraka procesa p i njegovih nizova uvode se dvije operacije:

- Operacija sljeđenja, koja se označava sa $\prec$. Oznaka $x \prec y$ označava da je x ispred y;
- Operacija izbora, koja se označava sa $\vee$.

Ako za lanac l važi da je $l = l_1 \prec l_2$ onda je dužina lanca l jednaka zbiru dužina lanca l_1 i lanca l_2, tj. $|l| = |l_1| + |l_2|$. Može se reći da je operacija sljeđenja ustvari povezivanje lanaca, koji se mogu posmatrati kao nizovi simbola u alfabetu $S(p)$. Izraz oblika $h(p) = h_1(p) \prec h_2(p)$ označava, da se proces p prvo ponaša u saglasnosti sa istorijom $h_1(p)$, a zatim u saglasnosti sa istorijom $h_2(p)$. Lako se pokazuje da je opracija sljeđenja ($\prec$) asocijativna, ali ne i komutativna.

Izraz oblika $h(p) = h_1(p) \vee h_2(p)$ opisuje ponašanje procesa p, koji se može razvijati bilo u saglasnosti sa istorijom $h_1(p)$, bilo u saglasnosti sa istorijom $h_2(p)$. Pravilo za izbor konkretne istorije ovom operacijom nije definisano. Operacija izbora ($\vee$) je komutativna, asocijativna i distributivna zdesna u odnosu na operaciju sljeđenja ($\prec$). To znači da važi:

a) $l_1 \vee l_2 = l_2 \vee l_1$,

b) $(l_1 \vee l_2) \vee l_3 = l_1 \vee (l_2 \vee l_3)$,

c) $l_1 \prec (l_2 \vee l_3) = l_1 \prec l_2 \vee l_1 \prec l_3$.

Tvrdnja 6.1. Istorija $h(p)$ svakog procesa p može se predstaviti u obliku:

$$h(p) = h_1(p) \prec \sim h(p), \quad \text{gdje je: } h_1(p) \in H(p) \qquad (6.8)$$

Dokaz: Tačnost ove tvrdnje slijedi iz pretpostavke da je ma koja istorija zatvorena slijeva i iz osobine operacije sljeđenja.

Tvrdnja 6.2. Neka je $H'(p) = \{ h'(p) | h'(p) = h_0 \prec \sim h'(p) \}$, gdje je $\sim h'(p) \in \sim H'(p)$, pri čemu je $\sim H'(p)$ skup podistorija neke istorije $H'(p)$. Onda važi sledeća relacija:

$$H'(p) = h_0 \prec \sum_{\sim h(p) \in \sim H'(p)} \sim h_i(p) \qquad (6.9)$$

Dokaz: Za dokaz ove tvrdnje dovoljno je iskoristiti distributivnost operacije sljeđenja u odnosu na operaciju izbora, po kojoj je:

$$h_0 \prec (\sim h_1 \vee \sim h_2 \vee \ldots \vee \sim h_i) = (h_0 \prec \sim h_1) \vee (h_0 \prec \sim h_2) \vee \ldots \vee (h_0 \prec \sim h_i)$$

uz uslov da važi sljedeći uslov $\forall i : (h_0 \prec \sim h_i(p)) \in H'(p)$ prethodna tvrdnja je dokazana.

Ako se uvede oznaka, da je $H_0^*(p) = \{h(p) \mid h(p) = h_0 \prec l\}$, gdje je $h(p) \in H(p)$, onda važi sljedeća tvrdnja:

Tvrdnja 6.3. Za ma koje fiksno $h_0 \in H(p)$ izraz $H_0^*(p) \in H(p)$ oblika $h_0 \prec \sum_{\sim h(p) \in \sim H'(p)} \sim h_i(p)$ je jedinstven sa tačnošću do asocijativne permutacije članova.

Dokaz: Ako se pretpostavi da postoji neko $h''(p) \in H_0^*(p)$, ali koje se ne može predstaviti u gornjem vidu, to onda dovodi do protivurečnosti. U tom slučaju u $h''(p)$ neće biti prefiksa h_0.

Iz prethodne tvrdnje zaključuje se, da za neki prefiks $h'(p)$, ma koje istorije iz $H(p)$ može se formirati klasa ekvivalentnosti $H''(p) = \{h(p) \mid h(p) = h'(p) \prec\sim h(p)\}$. Pri čemu za $\forall h'(p) \in H(p)$ i $\forall h''(p) \in H(p)$, uz uslov da je: $h'(p) \neq h''(p)$, važi da je $H''(p) \cap H'''(p) = \varnothing$

Sada se nad skupom $H(p)$ jednoznačno definiše struktura, čija će procedura izgradnje biti opisana u sljedećem tekstu.

Saglasno sa tvrdnjom 6.3 ponašanje ma kog procesa može se predstaviti u obliku $H(p) = \bigcup_{s_i \in S(P)} S_i^*$, gdje je $S(P)$ skup koraka programa P. Ovaj izraz se može uporediti sa grafom čiji je broj čvorova jednak broju koraka s_i, odakle počinju istorije procesa p i početnog čvora s° koji je fiktivni čvor i koji se uvodi u cilju izgradnje grafa. Veza s° sa čvorem s_i, tj. luk $(\overline{s^\circ s_i})$ postoji ako je s_i prefiks bar jedne istorije iz $H(p)$.

U nastavku, takođe u skladu sa tvrdnjom 6.3, može se za svako i pretpostaviti da važi da je: $S_i^* = \bigcup_{s_j \in S(P)} \sim S_j^*$, pri čemu se svaka struktura $\sim S_j^*$ može predstaviti grafom, čiji je početni čvor odgovara koraku s_i iz prethodno konstruisanog grafa. Ako je s_i posljednji korak istorije onda se na njega povezuje luk oblika $(\overline{s^\alpha s_i})$, gdje je s^α specijalni simbol koji označava kraj posmatrane istorije.

Kao rezultat gornje procedure dobija se povezan graf $G_H(p)$, koji predstavlja jednoznačnu sliku skupa istorija $H(p)$. Međusobna jednoznačnost preslikavanja između $G_H(p)$ i $H(p)$ slijedi iz definicije procedure po kojoj je izgrađen graf $G_H(p)$. Lako se dokazuje da je povezan graf $G_H(p)$ - *stablo* [56].

Dakle, skup istorija $H(p)$ se može predstaviti stablom, za koje se uvodi sljedeća oznaka:

$$G_H(p) = (N, S(p), f(N), L_H(p)), \qquad (6.10)$$

ovdje je N skup čvorova grafa, $S(p)$ skup koraka procesa p, $f(N)$: $N \to S(p)$ je funkcija označavanja čvorova, a $L_H(p)$ je skup lukova grafa.

Na taj način se ponašanje procesa p prikazuje stablom, čiji čvorovi predstavljaju korake procesa p, sa kojih počinju njegove istorije. Ponekad je zgodno predstavljati korak u obliku uređenog para čvorova povezanih lukom. Prvi čvor predstavlja pobudu a drugi reakciju.

Jedna od osnovnih karakteristika ponašanja distribuiranih programa jeste pojava nedeterminizma. Postoje dva oblika nedeterminizma: spoljašnji i unutrašnji, kako je to

već objašnjeno u poglavlju 5.5.2. Može se reći da izraz $s_i \vee s_j$ opisuje spoljašnji nedeterminizam ako za uticaje ova dva koraka važi da je: $I(s_i) \neq I(s_j)$, a unutrašnji ako je: $I(s_i) = I(s_j)$ i ako su reakcije ovih koraka različite, tj.: $R(s_i) \neq R(s_j)$. Ova definicija dozvoljava da se procijeni stepen nedeterminizma u posmatranom distribuiranom programu.

Na taj način se uz pomoć teorije grafova mogu opisati i istraživati strukture oblika $H(p)$, tj. ponašanje procesa. Na isti način se mogu opisati i relacije među procesima.

6.4.3 Ponašanje distribuiranog aplikativnog programa

Ponašanje distribuiranog aplikativnog programa P se određuje kao skup ponašanja procesa koji ga obrazuju:

$$H(P) = \bigcup_{p_i \in P} H(p_i) \qquad (6.11)$$

Skup istorija programa $H(P)$ se može predstaviti *šumom* stabala $G_H(p)$, koja se označava sa $U_H(P)$. Za skup koraka $S(P)$ se može reći da je djelimično uređen. Ta relacija odražava interakciju između procesa i nizanje koraka u istoriji svakog procesa. Može se ići i detaljnije, pa da ova relacija opisuju i uzročno-posljedične veze u skupu aktivnosti programa. Od njega zavisi semantika paralelizma.

Po definiciji je:

$$s_i \rightarrow s_j \;\Leftrightarrow\; \exists h(p):\ (s_i \succ s_j) \in h(p) \ \text{ i } \ s_i, s_j \in S(p)$$

s_i se naziva *uzrokom*, a s_j *posljedicom*. Ova relacija se označava sa $R \rightarrow (p) = \{(s_i, s_j) \mid s_i \rightarrow s_j\}$. Ona određuje uzročno-posljedične veze u skupu koraka $S(p)$, gdje je $p \in P$.

Relacija $R \rightarrow (p)$ se može proširiti i za slučaj događaja. Događaj je uticaj ili reakcija nekog koraka. Ako se sa e označi događaj, onda po definiciji važi:

$$e_i \rightarrow e_j \;\Leftrightarrow\; \exists s_k:\ (e_i = I(s_k) \,\&\, e_j = R(s_k)),\ \text{ili}$$
$$\exists s_i:\ (e_i \in s_k \,\&\, e_j \in s_i) \ \&\ (s_k, s_i) \in R \rightarrow (p)$$

izrazi oblika $e_i = I(s)$, $e_j = R(s)$ označavaju da se ti događaji odgovaraju uticaju i reakciji koraka s.

Za opis interakcije kako između samih aplikativnih procesa, tako i između aplikativnih procesa i procesa izvršne sredine uvode se pomoćne relacije na skupu koraka programa:

$$\tilde{R}(P) = \{(s_i, s_j) \mid s_i \in S(p) \,\&\, s_j \in S(q) \,\&\, p \in P \,\&\, q \in P \ \Rightarrow\ I(s_j).q = R(s_i)\}$$

tj. reakcija koraka s_i u ponašanju procesa p jednaka je uticaju koraka s_j na procesu q.

Na osnovu gore izloženog, istorija izvršenja programa $H(P)$ se može predstaviti sljedećom trojkom:

$$H(P) = \langle H(p_i), \{\alpha_i\}, \tilde{R}(P) \rangle \qquad (6.12)$$

tj. skupom istorija procesa $p_i \in P$, koji zadovoljavaju relaciju $\tilde{R}(P)$ i skupa $\{\alpha_i\}$ aktivnosti koje određuju početne korake tih procesa.

- Poslije završetka elementarnog procesa on uvijek vraća upravljanje onom procesu koji je inicirao njegovo izvršavanje;
- Kod elementarnih procesa trenutak prenosa poruke-rezultata i trenutak vraćanja upravljanja se podudaraju, drugim riječima ako je upravljanje vraćeno onda je poruka sa rezultatima već raspoloživa.

Funkcija logičkog vremena $C(t)$ prenosi na događaj onaj broj taktova, koji su protekli do momenta njegovog rađanja na datom izvršiocu. Trenutkom rađanja događaja smatra se vrijeme početka izvršavanja prvog elementarnog procesa nadležnog za ovaj događaj. Funkcija logičkog vremena $C(t)$ određena je na skupu svih događaja u procesima, koji odgovaraju datom izvršiocu. Logičko vrijeme u razmatranju distribuiranih sistema uveo je *Lemport* [83], [84], zbog odsustva globalnog sata u sistemu i nemogućnosti globalnog računanja vremena. Oblast vrijednosti ove funkcije je skup prirodnih brojeva. Osobine funkcije logičkog vremena $C(t)$ su sljedeće:

a) $$\forall e_i, e_j : e_i, e_j \in H(p_m) \& e_i \succ e_j \Rightarrow C(e_i) < C(e_j) \qquad (6.14)$$

tj. u jednom procesu uzrok nastaje uvijek prije posljedice;

b) $$\forall e_i, e_j : e_i \in H(p_k) \& e_j \in H(p_m) \& e_i \succ e_j \& p_k \sim p_m \Rightarrow C(e_i) < C(e_j) \qquad (6.15)$$

tj. ako uzrok i posljedica pripadaju različitim procesima i ti se procesi izvršavaju na jednom istom izvršiocu, onda po satu izvršioca uzrok nastaje uvijek prije posljedice.

Ako je $(e_i, e_j) \in R^* \rightarrow (p)$, može se uvesti oznaka: $e_i \rightarrow e_j$, gdje je $R^* \rightarrow (p)$, onda važe sljedeće posljedice:

1. Između dva uzastopna sekvencijalna događaja jednog procesa(recimo između uzroka i posljedice) uvijek prođe samo jedna jedinica vremena.

2. $$\forall e_i, e_j : e_i \rightarrow e_j \Rightarrow C(e_i) < C(e_j), \qquad (6.16)$$

tj. ma koji proces se izvršava sekvencijalno;

3. $$\forall e_j : e_0 \rightarrow e_j \Rightarrow C(e_0) < C(e_j), \qquad (6.17)$$

tj. obraćanje nekom procesu uvijek se dešava prije nego što se desi bilo kakav događaj u tom procesu, gledajući po satu datog izvršioca.

4. $$\forall e_i, e_j : e_i \in p_k \& e_j \in p_m \& p_i \succ p_j \& p_k \sim p_m \Rightarrow (e_i) < C(e_j) \vee C(e_j) < C(e_i) \qquad (6.18)$$

ovo znači da se, svi procesi na jednom izvršiocu izvršavaju se sekvencijalno.

5. $$\forall e_i, e_j : (e_i, e_j) \in R^* \succ (P) \& e_i \in p_k \& e_j \in p_m \& p_k \sim p_m \Rightarrow (e_i) < C(e_j), \qquad (6.19)$$

znači da, ako su dva događaja, koja pripadaju različitim procesima, međusobno povezana uzročno-posljedičnom vezom a izvršavaju se na jednom istom izvršiocu onda uzrok nastaje uvijek prije posljedice, gledajući po satu datog izvršioca.

Tvrdnja 6.4. Logičko vrijeme $C(t)$ ne narušava uzročno-posljedične odnose na skupu procesa, koji se izvršavaju na jednom izvršiocu, što se može predstaviti relacijom:

$$C(e_i) < C(e_j) \& (e_i, e_j) \in R \rightarrow (P) \ , \qquad (6.20)$$

ovdje je e_i – uzrok, a e_j – posljedica.

Dokaz: Pravilnost ove tvrdnje slijedi iz činjenice da je funkcija vremena $C(t)$ monotona. Ovo se javlja i kao logična posljedica prethodnih pet tačaka.

Ako se sa t označi promjenljiva koja pripada skupu T, a čije značenje je jednako astronomskom vremenu, koje se može odrediti uz pomoć klasičnog astronomskog praćenja vremena. Ovo ni na koji način ne zavisi od osobina posmatranog distribuiranog računarskog sistema, ali je važno za korisnike sistema.

Oblast definisanosti funkcije $C(t)$ se još može proširiti na sljedeći način. Neka se sa $C(t)$ označi broj taktova koji su protekli do trenutka t, onda važi:

$$\exists \rho : \forall t : \forall \Delta : \ \rho, t, \Delta \in T \ \& \ 0 < \Delta < \rho \ \Rightarrow \ C(t+\rho) - C(t) = 1 \ \& \ C(t+\Delta) - C(t) = 0 \quad \textbf{(6.21)}$$

tj. logički sat datog izvršioca "otkucava" sa konstantnom brzinom ρ. Ipak, obezbjeđenje jednoznačnog pokazivanja logičkog časovnika izvršioca, tokom različitih serija eksperimenata, moguće je samo izborom dovoljno velikog perioda posmatranja.

Memoriju izvršioca predstavlja setom $(M_{stek}(p), N, R_{eg})$, određuje kapacitet operativne memorije, strukturu registarske i stek memorije izvršioca.

Funkcija $M_{stek}(p)$ određuje potrebnu veličinu stek memorije. Ova funkcija je definisana unutar skupa procesa P_{uk}, a njene vrijednosti mogu biti samo cjelobrojne. To je broj memorijskih jedinica potrebnih za smještaj procesa p.

Maksimalni kapacitet operativne memorije posmatranog izvršioca je N memorijskih jedinica. Na datom izvršiocu može se izvršavati P'_{uk} procesa, pri čemu mora da važi da je: $\sum_{p \in P'_{uk}} M_{stek}(p) \le N$. U graničnom slučaju je $N > max(M_{stek}(a_i))$, gdje je $a_i \in P_a$, tj. izvršilac može smjestiti jednu komandu sa operandima.

Skup parametar koji karakterišu registarsku i stek memoriju izvršioca označen je sa R_{eg}.

Arbitar ϕ je jednoznačni funkcional, koji opisuje arbitražu na ulazu izvršioca. Oblast definisanosti ϕ je funkcija logičkog vremena datog izvršioca i skup karakteristika funkcija izlaza, povezana sa njegovim ulazom. Oblast vrijednosti je skup imena procesa, koji čekaju na ulazu. Po isteku $vt(a_i)$ na datom izvršiocu, ϕ određuje da jedan od procesa koji čekaju na ulaz mogu da dobiju izvršioca.

6.4.5 Funkcionisanje distribuiranog računarskog sistema

Funkcionisanje distribuiranog sistema se posmatra kao interpretacija ponašanja distribuiranog aplikativnog programa i izvršne (sistemsko-softverske) sredine na izvršiocu, tj. kao izgradnja istorije programa sa zadatim: ponašanjem programa, ponašanjem izvršne sredine, izvršioca i početnog stanju.

6.4.6 Instrument evidencije ili posmatrač

Matematička definicija distribuiranog računarskog sistema se može predstaviti sljedećom trojkom:

$$DRS = \langle H(P), S_{hd}, E_{nv} \rangle . \quad \textbf{(6.22)}$$

Ovdje je $H(P)$ ponašanje programa, S_{hd} je funkcija distribucije aplikativnih procesa u izvršnoj sredini a E_{nv} je distribuirana računarska sredina.

Izvršenje programa ***P*** na ***Env*** se evidentira uz pomoć posmatrača. Ova evidencija je označena kao posmatranje ili mjerenje. Posmatrač se zadaje u obliku identičnih algoritama i protokola njihovih interakcija. Ovi algoritmi se nazivaju sekvencijalnim posmatračima ili prosto posmatračima. Svaki od tih algoritama određuje izbor sljedećeg koraka u ponašanju procesa aplikativnog programa i izvršne sredine, koja pripada odgovarajućem izvršiocu. Svaki sekvencijalni izvršilac ima svog sekvencijalnog posmatrača.

Nekom izvršiocu Siz_i odgovara posmatrač Obs_i. U tom slučaju važi relacija:

$$M \times P_{uk} \times \phi_i \rightarrow S(p). \tag{6.23}$$

koji se može opisati kao:

$$Obs_i(s^*, \phi_i) = s. \tag{6.24}$$

gdje je: $s \in H(p)$, korak istorije procesa p.

6.5 Zaključci

Predloženi matematički model daje ocjenu i provjeru projektnih rješenja izgradnje i efikasnosti algoritma posmatranja i mjerenja ponašanja distribuiranih aplikativnih programa, kao i provjeru različitih osobina tih programa. Model opisuje dinamiku interakcije aplikativnih procesa sa softverskim i hardverskim sredinama. U njoj su jasno razgraničeni program i sredina njegovog izvršavanja. Ovo se ogleda u tome da opis ponašanja programa u modelu ne zavisi od vremena konkretnog izvršioca, različiti procesi programa razvijaju se nezavisno a sekvencijalni izvršioci funkcioniraju autonomno jedan od drugog. Model odražava hijerarhijsku strukturu računarskog sistema, kao i distribuiranost fizičke sredine i sistemskog softvera.

U okviru predloženog matematičkog modela:

- Razrađena je matematička apstrakcija za opis ponašanja programa, koja obuhvata zasad u praksi poznate, oblike nedeterminizma.
- Kreiran je matematički aparat za opis strukturnih i funkcionalnih svojstava hardvera računarskog sistema. On dozvoljava da se opiše decentralizacija upravljanja, višeznačnost metričkog vremena i konflikti.
- Izgrađen je instrument evidencije označen kao posmatrač, koji opisuje dinamiku izvršenja programa u distribuiranom računarskom sistemu. Pomoću njega može se istraživati međusobni uticaj dvaju oblika paralelizma.
- Zadatak analize produktivnosti formulisan je u matematičkom obliku.
- Istražen je način uređivanja aktivnosti, označenih u težinskom vektoru, bez uzimanja u razmatranje izvještaja o promjeni stanja i sadržaja registara.
- Pokazano kako se, znajući vremenski profil rada sistema, mogu dobiti različite karakteristike produktivnosti, uspostavljena je veza vremenskog profila sa osnovnim oblicima produktivnosti kao što su: propusna sposobnost sistema i vrijeme odgovora.

Rezultati dobijeni istraživanjem matematičkog modela, koriste se u početnoj fazi razrade distribuiranog računarskog sistema, kao i pri provjeri dokazu pravilnosti izgrađenog imitacionog modela računarskog sistema.

GLAVA

7. IMITACIONO MODELIRANJE DISTRIBUIRANOG SISTEMA

7.1 Opšti pojmovi

7.2 Osnovni zahtjevi koje imitacioni model treba da zadovolji

7.3 Postupak analize sistema imitacionim modeliranjem

7.4 Realizacija imitacionog modela distribuiranog sistema

7.5 Provjera i kalibracija modela

7.6 Zaključci

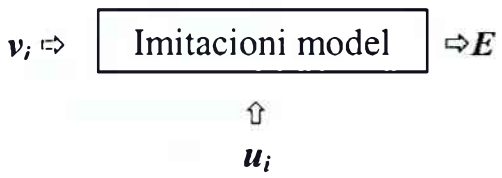
7.1 Opšti pojmovi

Metoda imitacionog modeliranja distribuiranog sistema predstavlja proces konstruisanja modela realnog distribuiranog sistema i numerički eksperiment sa ovim modelom na računaru. Dakle, sam proces imitacionog modeliranja se sastoji iz dva dijela: izgradnje modela i analitičke primjene modela za analizu performansi sistema koji se modelira. Drugim riječima, ako se koncepcijski ili matematički model posmatranog računarskog sistema, pomoću odgovarajućih algoritama i programskog alata implementira na računaru, dobija se ***imitacioni model*** tog sistema. Imitacioni model je programski sistem, koji je strukturno i funkcionalno sličan posmatranom računarskom sistemu i koji odražava evoluciju izlaznih objekata u diskretnom vremenu. Mjera strukturne sličnosti između realnog računarskog sistema, koji se analizira i njegovog imitacionog modela određuje se sljedećim faktorima:

- u imitacionom modelu mora biti sačuvana podjela posmatranog sistema na podsisteme,
- model treba da eksplicitno odražava većinu veza između podsistema.

Osnovni cilj modeliranja je razumijevanje ponašanja sistema i procijene njegovih performansi u granicama ograničenja nametnutih nekim utvrđenim skupom kriterijuma. Sam proces izgradnje i istraživanja imitacionih modela u literaturi se označava i kao *imitaciona metoda analize računarskih sistema*. Jedna od danas najznačajnijih primjena ove metode, jeste u oblasti distribuiranih sistema i računarskih mreža.

U daljem tekstu će biti razmotrena uprošćena struktura imitacionog modela. Iako matematička ili programska struktura modela može biti veoma složena, osnove njene izgradnje su jednostavni. Uprošćeno imitacioni model se može posmatrati kao "crna kutija". Na ulazu postoji skup promjenljivih i parametara, a kao rezultat na izlazu se dobija funkcija koja karakteriše ponašanje distribuiranog sistema, tj.: $E = f(v_i, u_i)$.



Ovdje su sa v_i označeni parametri i promjenljive na koje istraživač može uticati i kojima može upravljati, a sa u_i su označeni parametri sistema i oni parametri radnog opterećenja na koje istraživač ne može da utiče; f je funkcionalna zavisnost između v_i i u_i koja određuje ponašanje, preciznije rečeno performanse sistema E.

Parametri su detaljnije opisani u poglavlju 3.4.3. Bitno je naglasiti da, jednom utvrđena vrijednost parametara, na početku jedne serije imitacionog eksperimenta, ostaje nepromjenjena tokom odvijanja te serije eksperimenata. Npr. u modelu komunikacionog uređaja kakav je recimo ruter lokalne računarske mreže parametri mogu biti: brzina prijema, obrada i slanje podataka, veličina baferske memorije, i sl. To su karakteristike

date komponente realnog distribuiranog sistema i jasno je da su one nepromjenjene dok se ne izmjeni ta komponenta (u konkretnom primjeru to je ruter).

Promjenljive, koje učestvuju u izgradnji imitacionog modela distribuiranog sistema, nalaze se u izvjesnim granicama. Granice promjenljivih mogu biti uvedeni od strane istraživača (npr. na osnovu iskustva) ili samom fizičkom prirodom karakteristika sistema. Takva ograničenjima npr. može biti maksimalni i minimalni nivo opterećenja komunikacionog čvora, ili maksimalna veličina baferske memorije. Većina tehničkih normativa ili standarda računarskih sistema predstavljaju skup iskustvenih ograničenja. Uopšteno govoreći, ograničenja promjenljivih i parametara su uslovljena samom prirodom komponenti distribuiranog sistema.

Pri analizi distribuiranih računarskih sistema uz pomoć imitacionog modela mora biti što je moguće preciznije definisana tz. *ciljna funkcija*, tj. mora biti jasna predstava koji su ciljevi i zadaci modeliranja posmatranog sistema i koja su to neophodni kriterijumi za ocjenu ispunjenja tih ciljeva. U praksi se obično izdvajaju dva tipa ciljnih funkcija ili kriterijuma modeliranja: "podrška" postojećem distribuiranom sistemu i procjena prednosti (nedostataka) dodavanja novih ili izmjene postojećih resursa nekog posmatranog distribuiranog sistema. Prvi cilj je povezan sa održavanjem postojećeg distribuiranog sistema ili stanja, time što će se modeliranjem procijeniti njegova produktivnost. Ovaj se cilj podudara sa zadatkom ovog rada formulisanom u poglavlju 3.2. Drugi cilj se odnosi na procjenu pokazatelja performansi, kada se izmjeni ili doda neki od hardverskih ili softverskih resursa sistema. Ciljna funkcija mora biti jednoznačno određena ciljem ili zadatkom analize i mora odgovarati prihvaćenim rješenjima. Ona je obično sastavni dio modela i cio proces manipulisanja sa modelom usmjeren je na optimizaciju ili zadovoljenje tog cilja i zadatog kriterijuma.

7.2 Osnovni zahtjevi koje imitacioni model treba da zadovolji

Sa tačke gledišta ciljeva postavljenih u ovom radu, odnosno sa stanovišta izgradnje jednog principijelno novog modela, od posebnog su značaja sljedeće osobine distribuiranih računarskih sistema:

- paralelnost u radu praktično svih komponenti sistema,
- asinhronost odvijanja procesa u sistemu,
- postojanje hijerarhijskih nivoa interakcije komponenti sistema,
- složenost protokola komunikacije između procesa u sistemu,
- aktivna primjena tehnologije klijent/server.

Na osnovu gore izloženog i osobenosti funkcionisanja distribuiranih računarskih sistema, može se zaključiti, da je kod njih bitna ne samo količinska ocjena njihove efikasnosti, nego i algoritamska analiza ponašanja aplikativnih programa u distribuiranoj sredini [79]. Karakteristično za metode, primijenjene kod algoritamske analize je, da kod njih nema detaljne vremenske analize ponašanja sistema. One koriste formalni opis sistema uz pomoć termina nekog algoritamskog modela. Primjerom takvih modela javljaju se konačni automati, različite algoritamske šeme poznate kod algoritamskog programiranja, mreže Petri i slično [95], [96].

Pod količinskim karakteristikama računarskog sistema podrazumijevaju se svi oni parametri sistema koje se mogu količinski izmjeriti i koji za svoje određivanje, zahtijevaju vremensku analizu dinamike funkcionisanja sistema. Ovdje se podrazumijeva da je vrijeme metrička veličina, a ovaj dio analize označen je kao količinska analiza. Za sprovođenje količinske analize distribuiranih računarskih sistema tradicionalno su se primjenjivale analitičke metode kao što je teorija masovnog opsluživanja ili teorija čekanja (engl. *queuing theory*) [113], [115], [118]. Da bi količinska analiza distribuiranog sistema bila sprovedeno analitičkom metodom neophodno je formalno uprostiti strukturu sistema koji se modelira, budući da realističnije jednačine čine analizu vanredno složenom. Na taj način u nekim slučajevima nije moguće sprovesti ni približno tačne količinske proračune pokazatelja performansi na osnovu metode teorije masovnog opsluživanja. Čak i manje modifikacije fizičke strukture sistema zahtijevaju korenitu preradu analitičkog modela. Međutim, analitički modeli su često osnova za razumne aproksimacije pokazatelja performansi, a takođe dozvoljavaju dobijanje korisnih kvalitativnih rezultata.

Jedan sveobuhvatan i perspektivan sistem modeliranja treba da dozvoli analizu kako količinskih tako i algoritamskih svojstava ponašanja programa. Postojeći sistemi modeliranja, kako specijalizovani, tako i oni koji su zasnovani na univerzalnim jezicima programiranja nijesu dozvoljavali sprovođenje obe ove analize istovremeno. Po pravilu ove dvije analize su sprovođene odvojeno. Korišćene su različite metode, koje je teško bilo formalno povezati. To je u značajnoj mjeri usložnjavalo analizu i projektovanje distribuiranih računarskih sistema, a takođe je predstavljalo potencijalni izvor grešaka pri prelazu od jedne forme predstavljanja u drugu. Na taj način, aktuelan je zadatak izgradnje modela koji će jedinstvenim jezičkim sredstvima opisati i objediniti količinsku i algoritamsku analizu distribuiranih sistema.

Ne manje važnim pitanjem od opisa ponašanja samog sistema, javlja se problem opisa radnog opterećenja. Po pravilu sredstva samog jezika opisa nijesu dovoljna, ili nijesu sasvim udobna za rješavanje ovog zadatka. Zato je sasvim prirodno da u sistemu modeliranja budu izgrađena i posebna sredstva za opis radnog opterećenja sistema.

Pri razmatranju sistema modeliranja treba, imati u vidu, da složenost računarskog sistema povlači za sobom i složenost njegovog modela. Složenost modela je između ostalog određena:

- koncepcijom opisa dinamike programa,
- načinom i oblikom opisa ponašanja programa,
- načinom opisa izvršne i fizičke sredine,
- karakteristikom i vrstom uzajamnih hardverskih i softverskih veza,
- osnovnim oblicima interakcija između procesa u sistemu,
- izborom odgovarajuće matematičke apstrakcija itd.

Drugim riječima složenost imitacionog modela određuje logički model funkcioniranja distribuiranog računarskog sistema.

Da bi se donekle riješio problem složenosti, sistem modeliranja treba da zadovolji princip struktuiranosti kao osnovnog sredstva u borbi sa složenošću i sredstva podrške izmijeni nivoa detaljizacije modela. Primjena principa struktuiranosti utiče na rješenje mnogih ranije zanemarenih problema. Tako npr., metode analize moraju biti primjenljive na raznim nivoima agregacije sistema. To znači da treba izgraditi niz modela, različitog

stepena detaljizacije. Prirodno je, da se to mora odraziti i na sredstva i načine izgradnje radnog opterećenja i njegove korekcije pri prelazu sa jednog nivoa detaljizacije modela na drugi. Struktuiranost je važna i kao mogući način rješavanja problema adekvatnosti modela. Međutim, izgradnjom niza modela različitog stepena detaljizacije, javlja se opasnost dodatnog povećanja broja parametara veza između njih, što čini sistem modeliranja nepreglednim i složenijim za analizu sa svim posljedicama koje iz toga proizilaze.

Teorijski, model koji se izgrađuje mora da obezbijedi sredstvo stroge provjere i dokaz korektnosti predloženog prilaza, ocjenu i provjeru projektnih rješenja, konstrukcija i analize efikasnosti algoritama posmatranja i mjerenja ponašanja sistema, kao i provjeru različitih osobina distribuiranih programa. Praktično, model treba da izgradi osnovu za razradu eksperimentalnih sredstava, npr.: sistema imitacionog modeliranja, sistema debagiranja i sl.

Ukratko, osnovni zahtjevi koje imitacioni model treba da zadovolji mogu se formulisati na sljedeći način:

- model mora odražavati dinamiku interakcija programskih procesa;
- u modelu moraju biti jasno razdvojeni program i sredina njegovog izvršavanja;
- model mora odražavati hijerarhijsku strukturu, koja pripada programu, fizičkoj i logičkoj strukturi;
- model mora uzeti u obzir distribuiranost softverske i hardverske strukture: prva se ogleda u relativnoj nezavisnosti izvršavanja programskih procesa između tačaka sinhronizacije, a druga u nezavisnosti autonomnih računara;
- model mora odraziti različite načine povezivanja procesa programa na resurse softverske i fizičke sredine.

7.3 Postupak analize sistema imitacionim modeliranjem

Kako je već naglašeno u poglavlju 4.4 za rješavanje zadatka analize distribuiranih računarskih sistema posmatrane klase, koji je i primarni zadatak ovog rada, kao optimalna izabrana je nešto modifikovana tehnika imitacionog modeliranja. Saglasno ovom usmjerenju moraju se odrediti konkretna metodološka sredstva i procedure potrebne za izgradnju imitacionog modela i sprovođenje istraživanja uz njegovu pomoć. U okviru izgradnje sredstava i procedura imitacionog modeliranja distribuiranog sistema potrebno je obezbijediti sprovođenje sljedećeg niza koraka:

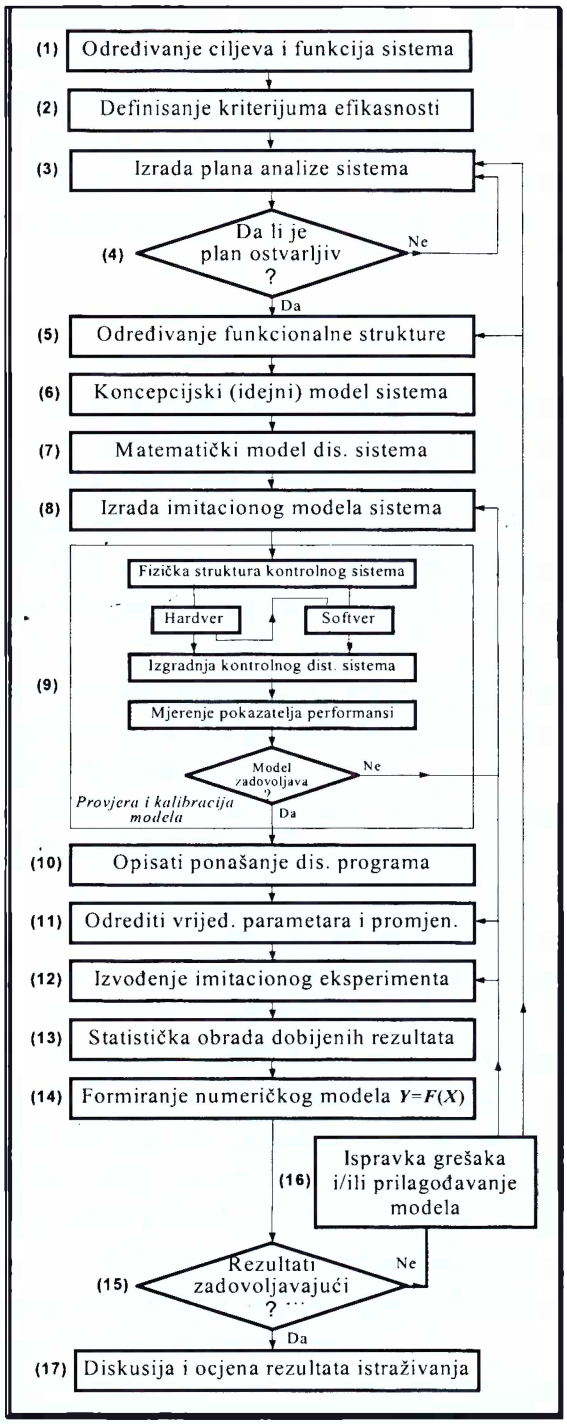
- odrediti osnovne karakteristike distribuiranog računarskog sistema koji je predmet analize;
- odrediti specifične osobine pojedinih komponenti, posmatrane klase distribuiranih sistema, relevantne za modeliranje;
- ustanoviti granice, ograničenja i instrumente mjerenja efikasnosti sistema, koji se posmatra;
- izvršiti izbor metodologije i napraviti plan analize i plan procesa imitacionog modeliranja;
- formulisati koncepcijski model, tj. ostvariti prelaz od realnog sistema ka nekoj logičkoj strukturi ili izvršiti apstrakciju realnog distribuiranog sistema;

- razraditi metode i sredstva izgradnje i opisa ponašanja programa;
- izgraditi matematički model sistema, kao teorijsku osnovu i dokaz ispravnosti metode imitacionog modeliranja na računaru;
- analizirati izgrađeni matematički model sa ciljem:
  - ✓ postavke zadatka analize produktivnosti distribuiranih računarskih sistema i razrade metoda i rješenja kroz izgradnju vremenskog dijagrama za dobijanje empirijskih modela produktivnosti,
  - ✓ odrediti uticaj oblika paralelizma i unutrašnjeg i spoljašnjeg nedeterminizma na ponašanje distribuiranih aplikativnih programa;
- izgraditi i implementirati algoritam imitacionog modela na računaru. drugim riječima opisati model na nekom programskom jeziku i pripremiti ga za izvršavanje na računaru;
- pripremiti relevantne podatke (npr. vrijednosti parametara i promjenljivih) neophodne za izgradnju i eksploataciju modela, i predstaviti ih u odgovarajućoj formi;
- procijeniti adekvatnost modela, odnosno, povećati do prihvatljivog nivoa stepen uvjerenosti, sa kog se može suditi u odnosu na korektnost zaključka o realnom sistemu, dobijenom na osnovu obraćanja modelu;
- razraditi specifičnosti metoda i sredstva imitacionog modeliranja konkretnog distribuiranih računarskog sistema;
- isplanirati strategiju eksperimenta sa modelom, koja treba da izdvoji potrebne informacije o realnom distribuiranom sistemu;
- izvršiti taktičko planiranje tj. odrediti način sprovođenja svake serije ispitivanja, predviđenih planom eksperimenta;
- izvršiti seriju eksperimenata sa modelom kao objektom ispitivanja, koja se sastoji od ostvarivanja imitacije sa ciljem dobijanja željenih podataka i analize osjetljivosti;
- interpretacija rezultata , koji su dobijeni putem imitacije;
- praktično korišćenje (interpretacija) modela ili rezultata modeliranja;
- dokumentacija, odnosno, registracija toka ostvarivanja projekta i njegovih rezultata, a takođe dokumentovanje procesa izgradnje i korišćenja modela.

Na osnovu gore izloženog može zaključuje se da imitaciono modeliranje složenog sistema, kakav je distribuirani sistem posmatrane klase, obuhvata više faza, pa se postupak modeliranja može predstaviti algoritmom kao na slici 7.1. Analiza distribuiranog sistema, prije svega, počinje sa eksplicitnim određivanjem ciljeva i zadataka koje dati sistem treba da riješi (1), što podrazumijeva i pripremanje informacija za izgradnju njegovog funkcionalnog (logičkog) modela i definisanje kriterijuma njegove efikasnosti (2). Slijedi izrada plana analize i određivanje opšteg pogleda na dati sistem (3), poslije čega treba izvršiti procjenu ostvarljivosti izrađenog plana (engl. *feasibility study*) (4), koja ima svoj tehnički, operacioni i ekonomski aspekt. Na osnovu prethodnih koraka dolazi se do logičke ili funkcionalne strukture distribuiranog sistema (5). Dalje se definiše koncepcijski ili idejni model distribuiranog sistema (6). Sljedeći korak se odnosi na izgradnju matematičkog modela (7) i njegove programske realizacije na računaru, odnosno imitacionog modela (8). Vrši se provjera i kalibracija modela mjerenjem pokazatelja performansi na kontrolnom (eksperimentalnom) sistemu (9).

Opisuje se ponašanje programa u distribuiranoj sredini tj. definiše se model radnog opterećenja (10) i zadaju vrijednosti parametara i promjenljivih (11). Izvođenjem serije imitacionih eksperimenata (12) dobijaju se rezultati čijom se statističkom obradom (13) formira numerički model (14) pomoću kog se dobijaju količinske karakteristike posmatranog sistema. Uz pomoć ovako formiranog modela ocijenjuju se granice efikasnosti datog sistema koji zadovoljava postavljene zahtjeve u pogledu obrade podataka, vremenu kašnjenja u prenosu poruka mrežom, tačnosti, cijeni, složenosti uređaja hardverske strukture, sigurnosti i drugim važnim karakteristikama.

Proces imitacionog modeliranja distribuiranog sistema ima iteracioni karakter. To znači da ako ocjena dobijenih pokazatelja performansi (15) ne daje zadovoljavajuće rezultate, vrši se korekcija ulaznih podataka (16), ili se pristupa povećanju stepena detaljizacije modela sa dijeljenjem i korišćenjem raznih nivoa apstrakcije. Zatim se ponovo ispunjavaju neki od predhodnih koraka. S tim u vezi, poželjno je, kako je to već objašnjeno u prethodnom (7.2) poglavlju, da imitacioni model bude strukturno izgrađen, čime se olakšava prijelaz sa jednog nivoa detaljizacije modela na drugi. Sa povećanim detaljisanjem modela treba biti oprezan, jer sa većim nivoom detaljizacije rastu troškovi modeliranja i složenost sistema modeliranja, pa postojanje velikog broja iteracionih koraka može dovesti da takva varijanta modela bude neadekvatna.



Slika 7.1 Algoritam procesa imitacionog modeliranja.

Postupak analize distribuiranog sistema obično se vrši na taj način što se razradi nekoliko varijanti sistema sa različitom fizičkom strukturom ili različitom sistemsko-softverskom strukturom. Onda se procjenom i upoređivanjem bira optimalno rješenje, s tim što se na svakom koraku mora voditi računa o potrebnom optimalnom nivou detaljizacije modela. U ovom radu se zadržava samo na analizi distribuiranog računarskog sistema kao cjeline, ne ulazeći u detalje njegovog tehničkog projektovanja, saglasno sa konkretno raspoloživim hardverskim i softverskim mogućnostima.

Na kraju se na osnovu rezultata modeliranja izrađuje potrebna dokumentacija i procijenjuje nivo zadovoljenja postavljenih ciljeva i kriterijuma modeliranja (17).

7.4 Realizacija imitacionog modela distribuiranog sistema

Kako je već naglašeno, imitacioni model je programski sistem, koji mora biti strukturno i funkcionalno sličan posmatranom distribuiranom računarskom sistemu. Realizuje se preslikavanjem odgovarajuće funkcionalne strukture ili koncepcijskog modela u programsku strukturu prilagođenu za izvršavanje na računaru. Ovdje predložen imitacioni model realizovan je na programskom jeziku C^{++} i označen je kao *imitacioni model sa porukama*. Model sa porukama dozvoljava da se predstave svi osnovni mehanizmi paralelizma, sinhronizacije i uzajamne konkurencije i kooperacije procesa distribuiranih aplikativnih programa.

7.4.1 Imitacioni model sa porukama

Imitacionog modela sa porukama je programski sistem. Njega čini skup nezavisno razvijenih programskih modula. Sistem programiranja zasnovan je na objektno orijentisanom prilazu. Ovakav prilaz ima niz prednosti kod izgradnje imitacionog modela distribuiranog sistema a najbitnije su:

- fleksibilnost variranja skupova i nivoa korišćenih objekata, čime se obezbjeđuje struktuiranost modela i omogućava jednostavna izmjena nivoa detaljizacije modela, tokom različitih serija imitacionih eksperimenata;
- nikakve interakcije među objektima modela ne mogu proizaći mimo operativnog sistema, tj. svi oblici interakcija objekata usresređeni su u strogo određenim tačkama odvijanja programa, čime je olakšano debagiranje programa;
- pristup objektima se može posmatrati, zaštiti i kontrolisati, zahvaljujući obezbjeđenju mehanizama zaštite, pomoću kojih objektno-orijentisani sistem opisuje svaki entitet, koju on razmatra kao objekat.

Teorijsku osnovu imitacionog modela sa porukama čini matematički model predstavljen u poglavlju 6.4. Kako je navedeno, osnovne komponente ovog matematičkog modela su: *aplikativni procesi, logički resursi, izvršioci i posmatrač*. S obzirom da je ideologija sistema programiranja modela sa porukama, zasnovana na objektno orijentisanom prilazu, prilikom konkretne izrade imitacionog modela, u zavisnosti od nivoa detaljizacije modela, obično se ide na objedinjenje nekih od komponenti definisanih u matematičkom modelu. Ovo se može obrazložiti: npr. činjenicom da sistemski i aplikativni procesi imaju istu prirodu, ili činjenicom da su hardverski elementi sistema ponekad nerazdvojivi od njihovih upravljačkih programa (engl. *drivers*). Obično se logički resursi i izvršioci predstavljaju jednim tipom objekata. Tako se u okviru predloženog modela uglavnom izgrađuju tri vrste objekata:

1. *Dinamički* objekti sistema, kojima je predstavljena funkcionalnost aplikativnih programa. Oni odražavaju dinamičko ponašanje aplikativnog programa ili pojedinih programskih elemenata i logički opisuju aplikativne programe tokom njihovog izvršavanja.
2. *Pasivni* objekti sistema predstavljaju hardverske i sistemsko-softverske elemente ili resurse sistema tj. izvršioce na kojima se izvršava dati program. Oni obezbjeđuju izvršavanje funkcija koje su definisane datim aplikativnim programima.
3. *Posmatrački* (monitorski, instrumentalni) objekti.

Dinamički objekti su matematički objekti, koji sadrži podatke o funkcionalnosti programskih elemenata sistema. To je logička predstava aplikativnih programa, opisana modelom. Takvi objekti se sreću u literaturnim izvorima pod različitim nazivima: istorija programa, operaciono-logička istorija, termalna istorija, termalno-logička istorija programa i slično. Osnovne razlike ovdje predložene matematičke apstrakcije procesa od onih u literaturi sastoji se u sljedećem:

- razvoj aplikativnog procesa se opisuje u vidu djelimično-uređenog niza koraka, a ne prosto događaja;
- na svakom koraku razlikuju se događaji koji se sastoje u prenosu dejstva na odgovarajući proces (npr. prenos podataka tom procesu), od događaja, kada sam proces pokušava djelovati na nekog;
- svaki korak ima atribut, označen kao složenost, koji dozvoljava razmatranje i razvoj procesa u vremenu;
- opis uzima u obzir kako unutrašnji tako i spoljašnji nedeterminizam u ponašanju programa;
- dozvoljava se prijelaz ka raznim oblicima paralelizma koji postoje u distribuiranim računarskim sistemima.

Na taj način se dinamičkim objektima opisuju sve osobenosti ponašanja aplikativnih procesa u distribuiranom sistemu. Važna prednost ovdje predloženog načina opisa ponašanja programskih elemenata, sastoji se i u tome, što on dozvoljava da se lociraju tačke nedeterminizma u ponašanju programa i da se taj nedeterminizam kontroliše.

Pomoću pasivnih objekata sistema predstavljaju se hardverski i logički resursi sistema i izgrađuje model hardverske i sistemsko-softverske strukture. Oni mogu sadržati sljedeće podatke:

- brzinske karakteristike, koje određuju vremenski interval koji je neophodan uređaju za izvršavanje dejstva,
- raspoređivanje funkcija i zadataka između autonomnih računara u sistemu,
- strukturu veza između autonomnih računara,
- tehničke performanse autonomnih računara,
- sistem upravljanja distribuiranog sistema,
- način komunikacije (sinhroni ili asinhroni) među računarima i sl.

Posmatrač je objekat (ili programski modul) koji evidentira uticaje ponašanja aplikativnog programa tj. radnog opterećenja na izvršiocima. Drugim riječima posmatrač dijagnosticira ponašanja aplikativnog programa pri ograničenjima, određenim distribuiranom računarskom sredinom. Posmatrač takođe skuplja, statistički obrađuje i prezentira rezultate modeliranja.

Svi ovi objekti imaju jedinstvenu identifikaciju i tip u okvirima objektno-orijentisanog programa imitacionog modela. Objekat dozvoljava da se inkapsuliraju svi podaci i metode njihove obrade. Interakcije između objekata u sistemu, kao što su npr. zahtjevi za logičkim resursima, obavlja se razmjenom poruka. Svaki objekat u saglasnosti sa svojim tipom može obrazovati složene hijerarhijske strukture, tj. komponentama objekta mogu biti objekti drugih tipova itd. Tip objekata određuje njegovu unutrašnju strukturu i skup operacija, koje su dopuštene nad tim objektom.

Zahvaljujući takvoj organizaciji programa sistem modeliranja koji podržava objektno orijentisani, dozvoljava da se detaljno razdjele gubici koji potiču od samog programa od gubitaka koji odgovaraju sistemu obezbjeđujući izvršavanje, posmatranje mjerenje i skupljanje podataka. Mogućnost rađanja tipova objekata po želji korisnika, jedinstvenost mehanizama upravljanja i zaštite omogućuje fleksibilnost uspostavljanja nivoa posmatranja i mjerenja u sistemu.

7.4.2 Sintaksa programske realizacije modela

Prilikom izgradnje programskih procesa modela korišćena je terminologija koraka (poglavlje 6.4.1). Procesi modela sarađuju jedan sa drugim, šaljući poruke. Poruka je apstraktni objekat koji ima tip. Broj tipova u modelu sa porukama je konačan. Svaki tip ima jedinstveno ime. Poruka može imati strukturu. Sve poruke istog tipa imaju jedinstvenu strukturu. Elementom strukture mogu biti poruke drugih tipova.

Opis procesa programa imitacionog modela sastoji se iz sledećih konstrukcija: bafer *box* i *tijelo procesa* i ima sljedeću sintaksu:

<proces programa> ::= **process** <ime>; <box>;
<tjelo procesa>.

Bafer box određuje memoriju procesa i ima sljedeću sintaksu:

<box> ::= **box**[<spisak tipova>];
<spisak tipova> ::= <tip>: <spisak imena>;
<spisak imena> ::= <ime> [,<ime>];
<tip> ::= private | own | external | source ;

Svaki box ima jedinstveno ime i predstavlja bafer poruke sa neograničenom dužinom. Svaki bafer-box može čuvati rezervaciju procesa za dobijanje poruke ili red čekanja za poruke koje su poslane tome procesu. Veličina tog reda i tip smještenih poruka nije fiksiran. Disciplina opsluživanja toga reda za proces nije determinisana.

Tjelo procesa se opisuje uz pomoć operatora čija je sintaksa:

<operator> ::= **set** <ime poruke> | **send** <ime boxa> |
receive <ime boxa> | **complex** <kod> |
delay <kod> | **stop** |.

Tjelo procesa predstavlja niz koraka i ima sljedeću sintaksu:

<tjelo procesa> ::= **begin** <korak> [; korak] end.
<korak> ::= **reseive** < ime boxa tipa own>; <reakcija>;

7.4.3 Bibliotečni prilaz izgradnje imitacionog modela

Bibliotečni prilaz izgradnje modela distribuiranog sistema, sastoji se u dijeljenju tehnološkog ciklusa izgradnje i korišćenja imitacionog modela na niz etapa. Na taj način se ostvaruje struktuiranost pri izgradnji modela, čime se omogućava da se složeni posao modeliranja distribuiranih sistema podjeli između sebe više istraživača. Prva etapa modeliranja, kod ovog prilaza, se sastoji u izgradnji biblioteke modela raspoloživih hardverskih i sistemsko-softverskih komponenti posmatranog distribuiranog sistema. Pod komponentama sistema podrazumijevaju se uređaji, hardverski i sistemsko softverski resursi ili grupe tih resursa koji obezbjeđuju neku određenu funkciju distribuiranog sistema. Prva etapa obuhvata dekompoziciju posmatranog distribuiranog

sistema na sastavne komponente, njihovu specifikaciju i na kraju njihovu apstrakciju. Ovu etapu realizuje *istraživač-projektant* modela. Zadatak istraživača-projektanta modela je da pripremi biblioteku modela komponenti izgrađujući apstraktne komponente sistema i pripremi podataka za njihovu implementaciju u model distribuiranog sistema. Pod podacima se ovdje podrazumijeva izbor najvažnijih karakteristika komponenti sistema i što je moguće tačnije određivanje vrijednosti njihovih relevantnih parametara. *Istraživač-eksploatator* modela, sa druge strane, može koristiti već pripremljenu biblioteku modela komponenti za izgradnju imitacionog modela distribuiranog sistema.

Pri izgradnji biblioteke modela sastavnih komponenti distribuiranog sistema treba težiti da ista može biti primijenjena na najširu moguću klasu distribuiranih sistema. Postojanje, na tržištu, raznovrsnih mrežnih i komunikacionih uređaja, kao i personalnih računara različite konfiguracije koji komuniciraju različitim protokolima i pod različitim operativnim sistemima, zahtijeva pripremu i izgradnju biblioteke modela, koja bi obuhvatili čitavu lepezu raznih komponenti distribuiranog sistema. Pojava novog uređaja na tržištu zahtijeva izgradnju novog člana biblioteke, koji će modelirati dati uređaj. Pri izgradnji takve komponente istraživaču moraju biti poznate čitav skup funkcija i protokola koji se koriste pri radu datog uređaja.

7.4.4 Model radnog opterećenje sistema

Kao radno opterećenje, ovdje posmatranog, distribuiranog sistema obično se upotrebljavaju odabrani ili za tu namjenu pripremljeni distribuirani aplikativni programi. Pri analizi distribuiranih sistema, radno opterećenje se uglavnom koriste na dva načina, i to:

1. Za opterećenje ili "punjenje" istraživanih distribuiranih računarskih konfiguracija i sistema, u slučajevima kada su eksperimentalno mjere njihove performanse ili kada se vrši provjera i kalibracija imitacionog modela distribuiranog sistema ili modela neke od njegovih komponenti.
2. kao ulazni podaci za imitacioni model posmatranog sistema tj. sistema koji se analizira.

U ovom radu se, kao osnova za izgradnju modela radnog opterećenja, uzima model ponašanje aplikativnih programa u distribuiranoj sredini. Ovo ponašanje je okarakterisano nizovima korisničkih zahtjeva, kojima se distribuirani aplikativni program obraća logičkim i fizičkim resursima sistema, kao i vremenom i specifičnostima tih obraćanja. Tokom imitacionog eksperimenta, model radnog opterećenja upravlja imitacionim modelom računarskog sistema, pa se on mora formulisati istovremeno i u skladu sa modelom samog sistema. Osnovne funkcije modela radnog opterećenja mogu se formulisati na sljedeći način:

- predstaviti korektne, adekvatne i zastupničke karakteristike distribuiranog aplikativnog programa, koje su potrebne za prognoziranje njegovog ponašanja na klasi distribuiranih računarskih sistema toliko širokoj koliko je to moguće,
- obezbijediti upravljačke i reproduktivne eksperimente pri procjeni pokazatelja performansi posmatranog sistema,
- skratiti toliko, koliko je to moguće količinu analizirajućih podataka bez bitnijeg smanjenja tačnosti analize,
- predstaviti podatke u formi, pogodnoj za njihovo korišćenje od strane modela sistema.

Prilikom izgradnje modela ponašanja distribuiranih aplikativnih programa ili modela radnog opterećenja, određeni parametri i/ili promjenljive, mogu biti zadati:

- *deterministički* na osnovu iskustva ili mjerenja na realnom sistemu i
- *stohastički*, uz pomoć slučajnih brojeva i ponavljanja, sa ciljem dobijanja srednjih vrijednosti nekih karakteristika sistema ili nekih endogenih promjenljivih.

S obzirom da model ponašanja programa upravlja odvijanjem eksperimenta sa imitacionim modelom sistema, zavisno od načina definisanja parametara i promjenljivih radnog opterećenja razlikuju se dva tipa imitacionog modela distribuiranog računarskog sistema:

1. Modeli kojim se upravlja trasama, odnosno, uređenim i unaprijed tačno definisanim skupovima ulaznih promjenljivih i
2. Modeli kojim se upravlja raspodjelama slučajnih brojeva, odnosno modeli, upravljani nizovima slučajnih događaja, koji na kompleksan i algoritmom programa definisan način utiču jedan na drugog.

Kod imitacionog modela sa porukama mogu se primijeniti obe metode, a moguće su i kombinacije ove dvije metode modeliranja radnog opterećenja. Izbor modela radnog opterećenja usko je povezan sa strukturom distribuiranog aplikativnog programa i stepenom detaljizacije modela. Podrazumijeva se da bilo koja izmijena u opisu radnog opterećenja distribuiranog sistema zahtijeva dublju i globalniju izmijenu opšte strukture modelirajućeg programa, a takođe i njegovih odvojenih djelova. Međutim, fleksibilniji sistem modeliranja minimizira te izmijene.

7.5 Provjera i kalibracija modela

Imitaciono modeliranje se, prije svega, definiše kao *metoda procjene* pokazatelja performansi posmatranog distribuiranog sistema. Rezultatima dobijenim pri modeliranju ne može se vjerovati ako model nije dovoljno tačan.

7.5.1 Adekvatnost modela

Osnovno pitanje, koje proizilazi pri korišćenju imitacionog modela, sastoji se u tome da se odgovori: koliko dobro model predstavlja odgovarajući realni računarski sistem? Kako dokazati da model nekog komunikacionog uređaja u distribuiranom sistemu funkcioniše isto tako kao i sam uređaj sa potrebnom tačnošću u zadatom dijapazonu? To su pitanja ***adekvatnosti*** modela. Imitacioni model sa porukama se izgrađuje za postizanje konkretnog cilja, tj. za analizu jedne određene klase distribuiranih računarskih sistema i njegova adekvatnost ili opravdanost ocijenjuje se u terminima tog cilja.

Provjera adekvatnosti modela predstavlja proces, u toku kog se postiže prihvatljiv nivo uverenosti korisnika u to da je bilo koji zaključak o ponašanju sistema dobijen na osnovu modeliranja pravilan. Nemoguće je dokazati da se ova ili ona imitacija javlja pravilnim ili "istinskim odrazom" realnog distribuiranog sistema, zato se problem dokazivanja adekvatnosti modela uglavnom svodi na pokazivanje da je procedura izgradnje modela korektna i da su struktura i funkcionalnost modela pravilni tj. da dovoljno dobro odražavaju strukturu i funkcionalnost realnog sistema. Ovo se posebno

odnosi na distribuirane računarske sisteme koje karakterišu nedeterminizma u ponašanju aplikativnih programa.

Provjera modela je izuzetno važna etapa imitacionog modeliranja. Odgovornost za provjeru adekvatnosti modela, u toku prve etape modeliranja, tj. tokom formiranja biblioteke modela pojedinih komponenti sistema, leži na istraživaču-projektantu. Istraživač-eksploatator je odgovoran da li je model sistema, u cjelini, adekvatan realnom distribuiranom sistemu. Korišćeni imitacioni model može izazvati utisak realnosti i pravilnosti, kod istraživača-projektanta modela i kod istraživača-eksploatatora modela, koji su upoznati sa detaljima njegove realizacije i pristupaju k njemu sa povjerenjem. Međutim, za slučajnog posmatrača, a ponekad i za iskusnog specijalistu, obično su sakrivene izvorne pretpostavke na osnovu kojih je model izgrađen. Zato nepažljiva provjera modela, izvršena površno, može dovesti do netačnih zaključaka o performansama posmatranog sistema i do negativnih posljedica ako se na osnovu pogrešnih rezultata modeliranja izgradi i instalira realan distribuirani sistem.

U ovom radu korišćen je *racionalistički prilaz* za izgradnju adekvatnog imitacionog modela. Naime, u poglavlju 6.4, matematičkim putem tj. uz pomoć matematičkih izraza izgrađena je hipoteza relativnih odnosa i korelacija između aplikativnih procesa, logičkih resursa i hardverskih izvršilaca posmatranog distribuiranog sistema. Korišćene su metode formalne logike za dobijanje određenih zaključaka i informacija o realnom sistemu. U ovom slučaju model sa porukama se dobija kao skup pravila logičke dedukcije, koje polaze od pretpostavki koje se mogu (ili ne) empirijski provjeriti i dolazi se do objektivnih zaključaka, koji se implementiraju u model.

7.5.2 Tačnost modela

Prilikom projektovanja modela, teži se izgradnji takvog modela koji će posjedovati karakteristike bliske karakteristikama realnog sistema koji se analizira. Ocijeniti kvalitet modela znači ocijeniti stepen uvjerenosti u to da, rezultati dobijeni uz pomoć modela, odgovaraju realnom sistemu. Pojam dokaza pravilnosti modela nije binarna promjenljiva i ne može se rešiti po "crno-bijelom" principu. Obično izgradnja apsolutno adekvatnog modela nije jasna čak ni teorijski.

Rezultatima dobijenim pri modeliranju ne može se vjerovati ako model nije dovoljno tačan. Problem ***tačnosti*** modela ogleda se u činjenici, da su rezultati modeliranja tačni samo u tom slučaju kada se identični procesi dešavaju u modelu i u realnom sistemu. Ako između tih procesa postoji neko razmimoilaženje, onda je rezultatima modeliranja svojstvena netačnost, koja se ne može ustanoviti povećanjem broja ponavljanja imitacionog eksperimenta ili primjenom različitih metoda statističke analize. Ako se rezultati takvog, nedovoljno tačnog ili nedovoljno provjerenog modela, ipak koriste za analizu, istraživači moraju uzeti u obzir eventualne greške modeliranja i voditi računa o ograničenoj primjeni tako dobijenih rezultata.

Tačnost modela se definiše uz pomoć pokazatelja performansi, izabranih za procjenu sistema. Model je dovoljno tačan ako se razlika između vrijednosti pokazatelja performansi, dobijenih pri modeliranju distribuiranog sistema i vrijednosti izmjerenih na realnom sistemu nalazi u granicama dopuštene greške. Na taj način, tačnost modela se mjeri razlikom između pokazatelja performansi realnog sistema i njegovog modela. Izbor pokazatelja performansi i veličina maksimalne dopuštene greške zavise od tipa

potrebne analize, odnosno istraživanja. Tačnost modela se može predstaviti u obliku realnog broja na skali od 0 do 1, gdje je sa 0 označen apsolutno netačan model, a sa 1 apsolutno tačan. Sa porastom tačnosti modela raste i cijena modeliranja, ali raste i njegova vrijednost za istraživače i korisnike, koji primjenjuju dobijene rezultate.

Na osnovu već izloženog, tokom procesa procjene tačnosti imitacionog modela razmatraju se: unutrašnja stanje modela, njegovo podudaranje sa realnim sistemom i pravilnost interpretacije rezultata. Postupak procijene se može sprovesti u tri etape.

1. Provjera opšte pravilnosti rada modela,
2. Procjena tačnosti i podudarnosti ponašanja modela i ponašanja realnog sistema,
3. Analiza i interpretacija podataka dobijenih kao rezultat imitacionog eksperimenta.

7.5.3 Kalibracija modela

Ako tačnost modela nije zadovoljavajuća, onda model mora biti izmijenjan, a proces provjere ponovljen. Ta operacija se naziva ***kalibracija*** modela. Pojam kalibracije modela takođe se odnosi i na način usklađivanja biblioteke modela komponenti sa karakteristikama konkretnog uređaja ili resursa distribuiranog sistema. Onda se pod kalibracijom podrazumijeva izbor parametara i metoda usklađivanja biblioteke modela komponenti sa konkretnim realnim uređajima.

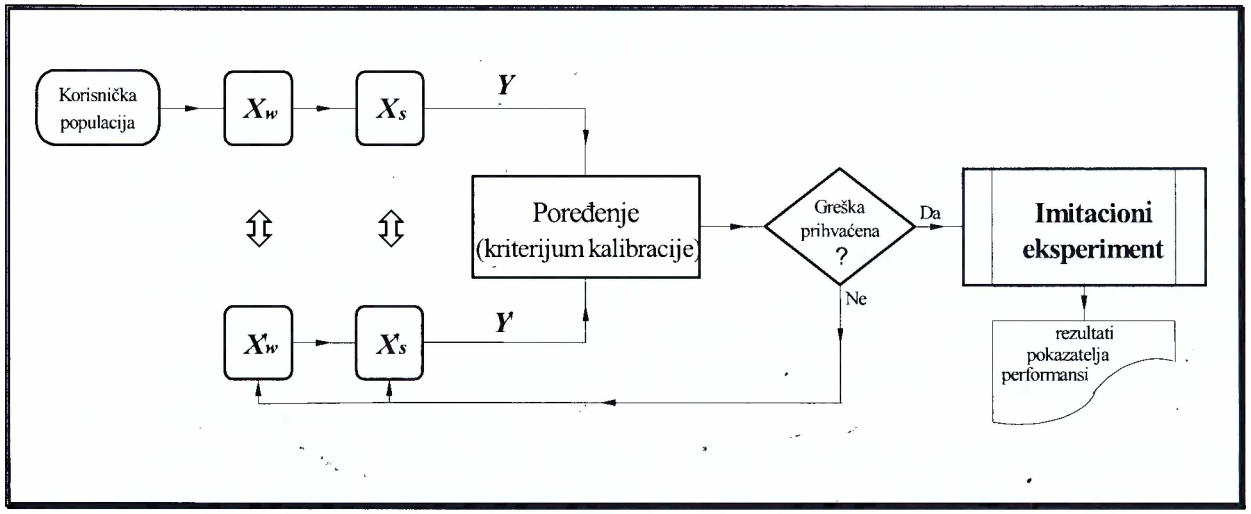
Cilj kalibracije se sastoji u smanjenju ili eliminisanju netačnosti koje mogu nastati pri formiranju modela. Međutim, i tačan model može davati neispravne rezultate čak i u procesu kalibracije, ako je metoda rješavanja, ili vrijednosti ulaznih parametara, ili i jedno i drugo neispravni. Na taj način su istraživači ili korisnici modela dužni izbjeći pripisivanje ponašanju modela, greške prouzrokovane netačnim metodama rješavanja, ili netačnim vrijednostima parametara.

Algoritam procesa kalibracije imitacionog modela predstavljen je na slici 7.2. Distribuirani računarski sistem sa skupom sistemskih parametara ili konfiguracijom X_s, pri radnom opterećenju X_w ima performanse Y. Radno opterećenje sistema X_w generiše se van posmatranog sistema od strane korisničke populacije i obično je opisano tekstom distribuiranog aplikativnog programa. Pokazatelji performansi, čiji je skup vrijednosti označen sa Y, mogu biti predstavljene kao: skalarne veličine, skalarne funkcije parametara i/ili promjenljivih, funkcije raspodjela slučajnih brojeva ili trasa parametara sistema. Pokazatelji performansi takođe mogu biti posmatrane kao slučajne veličine ili opisane uz pomoć srednje vrijednosti i disperzije.

Model distribuiranog sistema sa konfiguracijom X_s je X'_s. Ako sa ovim modelom upravlja radno opterećenje X'_w, koje modelira X_w, dobijeni pokazatelji performansi su Y'. Upoređivanje vrijednosti istoimenih pokazatelja performansi sistema iz skupova Y i Y' daje mjeru tačnosti modela distribuiranog sistema i modela njegovog radnog opterećenja. Postoje tri osnovne klase grešaka u modelu koje utiču na rezultate dobijene modeliranjem:

- *greške zadavanja parametara i promjenljivih*, koje su prouzrokovane korišćenjem netačnih vrijednosti parametara sistema.
- *greške formiranja*, koje su prouzrokovane pogrešnim, nepotpunim, ili nedovoljno detaljnim modelom,
- *greške rješenja*, koje su prouzrokovane nekorektnom, ili vanredno uprošćenom metodom rješavanja modela,

Ako se javi neprihvatljivo neslaganje između performansi realnog distribuiranog sistema i performansi dobijenih modeliranjem vrši se usklađivanje i ispravka parametara i promjenljivih modela, pa se postupak ponavlja. Kada se vrijednosti parametara i promjenljivih usaglase sa realnim vrijednostima i njihove greške svedu na minimum, a model upoređivanjem veličina iz skupova Y i Y' i dalje daje netačne rezultate, potrebno je prekomponovati čitav model. Iterativna procedura kalibracije modela se završava samo kada tačnost modela postane zadovoljavajuća.



Slika 7.2 Blok šema iterativne procedure kalibracije modela.

Problemi pri kalibraciji modela nastaju kada distribuirani sistem, koji se analizira, ne postoji fizički ili nije dostupan za eksperiment. To je slučaj kada se metoda modeliranja primjenjuje u projektovanju distribuiranih sistema. Praktično ovaj problem se svodi na to da je potrebno "ubijediti se " da je koncepcijski model korektno preslikan u imitacioni model. Problem je sličan provjeri korektnosti programa. Jedan od načina da se prevaziđe ovakav problem je da modelirajući sistem predstavlja samo novu konfiguraciju već izmjerenog sistema, ili da se za potrebe analize, realno, izgradi eksperimentalni distribuirani računarski sistem, tzv. kontrolni sistem principijelno iste konfiguracije kao i onaj koji se želi projektovati. U praksi se može izabrati niz testnih slučajeva i iskoristiti model za dobijanje vrijednosti pokazatelja performansi, a zatim provjeriti razumnost dobijenih rezultata modeliranja i njihovu prigodnost manuelnim izračunavanjem.

7.6 Zaključci

Zadatak imitacionog modeliranja kompleksnog sistema kakav je distribuirani računarski sistem svodi se na izgradnju unutrašnje strukture modela na osnovu apriorne informacije, predhodnih istraživanja i postojećih teorija. Složeni imitacioni model distribuiranog sistema sastoji se iz više prostih modela (npr. bibliotečni prilaz). Prostim modelima se obično imitiraju procesi ili uređaji koji su dobro poznati i određeni i koji imaju široku primjenu u posmatranoj klasi distribuiranih sistema. Međutim, kad se ti prosti modeli objedine u složenu cjelinu broj mogućnosti međusobnih interakcija čini razumijevanje ponašanja čitavog sistema modeliranja otežanim.

Prilikom izgradnje imitacionog modela sa porukama, za koji se može reći da adekvatno i dovoljno tačno predstavlja distribuirani računarski sistem posmatrane klase, koriste se metode, predstave i hipoteze koje su dobro poznate iz njihovih predhodnih primjena (poglavlje 4.3), a druge se zasnivaju na strogo teorijskoj osnovi izvedenoj iz predhodnih istraživanja i literature (poglavlje 5.5 i 6.4). Metoda ili hipoteza koja se odbaci sa nekom apriornom informacijom (poglavlje 4.4), ne može se koristiti sve dotle dok dopunskim istraživanjem ili iskustvom ne izmjeni se odnos prema njoj. Nema smisla uključiti u razmatranje metodu ili hipotezu, koja se smatra pogrešnom ili neprimenljivom kod analize distribuiranih sistema posmatrane klase. Kada se imitacioni model pažljivo izgrađuje, što je i bio slučaj kod projektovanja modela sa porukama, nivo uvjerenosti u njegovu ispravnost je visok.

Sljedeća etapa izgradnje modela sastoji se u postupku provjere adekvatnosti modela za predskazivanje ponašanja realnog distribuiranog sistema koji se modelira. U ovoj etapi istraživač-projektant se mora sam ubediti i takođe mora ubjediti onog ko koristiti dati model, u to da on radi baš tako kako i treba da radi tj. dokazati da je model primjenljiv i koristan. To je u ovom radu urađeno tako što je pokazano da postoji sistemski nezavisna mjera ponašanja distribuiranog aplikativnog programa i matematičkim opisom tog ponašanja.

Uopšteno govoreći, imitacioni model sa porukama odražava unutrašnju strukturu i unutrašnje veze distribuiranog sistema koji se modelira. U ovom radu pokazano je da se na osnovu ponašanja distribuiranog aplikativnog programa mogu dobiti informacije o radu čitavog sistema. Pravilnost i tačnost teorije izgradnje modela se može provjeriti samo praktično, mjerenjem na realnom sistemu slične ili iste konfiguracije. Sa problemima mjerenja performansi distribuiranog sistema baviće se sljedeće poglavlje ovog rada. S obzirom da se pri izgradnji ma kog modela, pa i modela sa porukama, koji je ovdje predložen, koristi uprošćena apstrakcija realnog sistema, onda se model smatra apsolutno tačnim samo ako se jednoznačno podudara sa realnim distribuiranim sistemom koji se posmatra. Slijedi da postoje različiti stepeni korektnosti i tačnosti modela. Modeliranje distribuiranih sistema uz pomoć modela sa porukama ne bavi se traženjem apsolutno tačnih vrijednosti pokazatelja performansi ovih sistema, već samo njihovih približnih ili srednjih vrijednosti. Ono čak dozvoljava da se preko više serija imitacionih eksperimenata omogući postepeno približavanja istinskim vrijednostima ovih parametara..

Može se na kraju reći da se procesi: izgradnje, provjere i realizacije modela nerazdvojivi.

GLAVA 8

8. MJERENJA U DISTRIBUIRANOJ RAČUNARSKOJ SREDINI

8.1 Uvod

Metode matematičkog i imitacionog modeliranja, iako mogu biti praktične i jeftine u finansijskom i vremenskom pogledu, prvenstveno se koristi kao metode *procjene* računarskih performansi. Dobijeni rezultati odstupaju od pravih vrijednosti uvijek kada posmatrani sistem nepotpuno zadovoljava uslove pretpostavljene u matematičkom ili imitacionom modelu. Sa druge strane, direktnim mjerenjem se dobijaju približno stvarni pokazatelji performansi realnog distribuiranog sistema. Glavne prednosti mjerenja performansi na stvarnom sistemu su eliminisanje grešaka prouzrokovanih zanemarivanjem nekih relevantnih sistemskih karakteristika koje imaju efekat na performanse sistema. Ukoliko se prilikom mjerenja pokazatelja performansi realnog distribuiranog sistema koriste njegovi resursi, mjerenje može smetati radu sistema i imati znatan uticaj na izmjerene veličine. Smanjenje performansi i netačnost rezultata mjerenja, izazvani unošenjem nekog instrumenta u sistem, predstavljaju konstantan problem za istraživače i mogu umanjiti značaj podataka dobijenih mjerenjem [29], [30], [32], [44].

8.2 Koncepcija posmatrača

Rezultati mjerenja performansi realnog računarskog sistema određene klase, obezbjeđuju osnovu za analizu takvih sistema. Mjerenje se sprovodi u procesu "posmatranja" sistema. *Posmatrač* ili instrument mjerenja (engl. *monitor*) se definiše kao nešto što prati pojave koje se dešavaju u sistemu i ako su s njegove tačke gledišta, u sistemu dogodila bilo kakva izmijena mjerenjem se ocijenjuje njihova količinska vrijednost. Posmatrač, dakle, sarađuje sa sistemom čije performanse mjeri. On bi trebalo da bude pasivni posrednik pri prenosu određene informacije iz sistema, ali to nažalost, nije uvijek moguće ostvariti. Ovom prilikom se ne ulazi u prirodu, odnosno, suštinu samog posmatrača.

Posmatrač može biti bilo spoljašnji bilo unutrašnji. *Spoljašnji posmatrač* razmatra sistem kao "crnu kutiju", koja sadrži ograničeni broj poznatih funkcija. Posmatranje se u suštini svodi na mjerenje izmijena u reakciji sistema pri kontrolirajućim izmjenama radnog opterećenja. Taj prilaz se obično koristi pri uporednoj ocjeni sistema, kada se kao kvalitativna mjera produktivnosti bira propusna sposobnost sistema ili vrijeme odziva. *Unutrašnji posmatrač* obezbjeđuje mjerenje i kontrolu izmijena, koje se dešavaju unutar sistema. Ciljevi mjerenja mogu biti različiti: dijagnostika aparature, debagiranje programa, analiza produktivnosti sistema i mnogi drugi.

Prilikom izbora posmatrača treba voditi računa o sledećem:

- Kakva je informacija o radu sistema nužna za postizanje postavljenih ciljeva?
- Gdje i u kojim tačkama sistema se ona može dobiti?
- Kako i pomoću kakvih sredstava je moguće nju izdvojiti iz sistema?

Posmatrano ponašanje distribuiranog sistema je niz izmijena posmatranih stanja sistema. Posmatrano stanje odražava ponašanje sistema čak na najnižem nivou sistema, a to je stanje svih memorijskih elemenata u sistemu: registarske i operativne memorije, spoljašnje memorije, bafera perifernih uređaja i sl. Međutim, pri analizi produktivnosti, po pravilu, se radi sa uprošćenim modelom sistema, za analizu su prije svega od interesa informacioni tokovi u sistemu ili tokovi podataka i upravljanja, koji se javljaju pri uzajamnom djelovanju programa. Zato se obično pod pojmom stanja sistema, podrazumijeva samo memorija koja odražava stanje objekata u programu. U njoj se ne odražavaju sve izmijene stanja sistema.

Jednim od ključnih pitanja pri rješavanju zadatka mjerenja i analize performansi distribuiranog računarskog sistema javlja se to kako i u kojim terminima opisati ponašanje aplikativnih programa u distribuiranoj sredini, odnosno, kako opisati radno opterećenje posmatranog sistema. Na pojmu stanja direktno ili indirektno baziraju se sve metode opisa ponašanja programa [51], koji se posmatra kao niz prelaza iz stanja u stanje. Pri tome se, kako je već naglašeno u poglavlju 5.4, koriste dva principijelno različita prilaza: denotacioni i operacioni [54], [63], [64].

U *denotacionom prilazu* program se posmatra kao preslikavanje $\Pi : X \rightarrow Y$, gdje je X ulazni podaci, a Y rezultati (tj. akcenat se stavlja na transformaciji podataka programom). Ovo preslikavanje se sastoji iz niza preslikavanja $\{\Pi_1, \Pi_2, ..., \Pi_i\}$, realizovanih operatorima programa, koji mijenjaju vrijednosti promjenljivih. Na taj način, ponašanje programa se opisuje neposredno u terminima vrijednosti njegovih memorijskih elemenata. Time se u njemu direktno odražavaju samo te aktivnosti programa (npr. izvršavanje operatora prisvajanja), koje izazivaju izmijene promjenljivih programa u memoriji. Sva ostala dejstva (npr. prenos upravljanja) uspostavljaju se jednoznačno samo na osnovi uzastopnog determinističkog karaktera izračunavanja.

Kod *operacionog prilaza* dinamika programa se razmatra kao niz događaja, podrazumijevajući prije svega događaj kao promjenu stanja, povezujući ga prije svega s tim aktivnostima čiji početak ili kraj izaziva te promjene. U takvom prilazu može se značajno proširiti pojam stanja, uključivši u njega ne samo neposredno promjenljive programa, nego i druge oblike memorije elemente u sistemu. Osnovno je da, aktivnosti koje mijenjaju vrijednosti promjenljivih, budu različite za posmatrača. Takve aktivnosti mogu biti izvršavanje mikrokomandi, komandi, funkcija, sistemskih poziva, slanja poruka itd. Njihov nivo zavisi od nivoa na kojem se želi analizirati opterećenje sistema [68].

U prvom slučaju, kod denotacionog prilaza, akcenat se stavlja na transformaciju ili obradu podataka programom, tj. program se razmatra kao funkcija. U drugom slučaju akcenat se stavlja na operacionu prirodu distribuiranih programa. U suštini ovo su dva različita prilaza opisa jedne iste pojave. Kako distribuirani programi sadrže visok stepen paralelizma u sebi, a paralelizam ima operacionu koncepciju, ovdje se podrobnije razmatra operacioni prilaz.

Podaci, dobijeni tokom mjerenja performansi računarskih sistema, mogu se podijeliti po formi u četiri kategorije:

- trase,
- relativna aktivnost,
- frekventne karakteristike aktivnosti i
- statističke ili prosječne karakteristike aktivnosti.

1. Trase – kod njih se aktivnost A_k opisuje trojkom (A_k, t_i, T_i), gdje je:

A_k - oznaka ili šifra date aktivnosti ili njemu odgovarajućeg događaja,

t_i - vrijeme početka i-te pojave aktivnosti,

T_i - dužina trajanja i-te aktivnosti.

Korišćenjem operacionog prilaza, potpuno je logično, da aktivnost karakterišu tri događaja: početak, trajanje i kraj aktivnosti.

2. Relativna aktivnost pokazuje, koji dio vremena od ukupnog vremena rada programa se troši na ispunjavanje aktivnosti A_k:

$$r_k = \frac{1}{t - t_0} \int_{t_0}^{t} a_k(\tau) d\tau, \quad \text{gdje je} \tag{8.1}$$

$a_k(\tau) = 1$, ako se sistem nalazi u stanju, koje odgovara izvršavanju aktivnosti A_k i $a_k(\tau) = 0$ u suprotnom slučaju,

t i t_0 su momenti početka i kraja mjerenja, pri čemu je $t \geq t_0$.

3. Frekventna karakteristika aktivnosti. Osnovom ove karakteristike javlja se frekvencija izvršavanja ili odvijanja posmatrane aktivnosti. Frekventna karakteristika aktivnosti c_k mjeri se brojem događaja e_k, koji iniciraju aktivnost A_k :

$$c_k = \frac{1}{t - t_0} \sum_{t_n} e_k(\tau); \quad \text{gdje je} \tag{8.2}$$

$t \geq t_n \geq t_0$, i $e_k(\tau) = 1$, ako je $\tau = n$ i $e_k(\tau) = 0$, u suprotnom slučaju,

t_n je vrijeme pojave aktivnosti A_k.

4. Statistička (prosječna) karakteristika aktivnosti. U ovom slučaju aktivnost karakteriše vremenska raspodjela njegovog izvršavanja, ako se ono mijenja od jednog njegovog izvršavanja do drugog. Neka je $f_k^n(T)$ funkcija vremenske raspodjele izvršavanja (odvijanja) aktivnosti A_k u momentu završetka njegovog n-og pojavljivanja, tada je:

$$f_k^n(T) = \frac{1}{n} \sum_{i=1}^{n} g(T, T_k^i); \quad \text{gdje je} \tag{8.3}$$

T - vrijeme trajanja i-tog pojavljivanja A_k

$g(T_1, T_2) = 1$, ako je $T_1 = T_2$ a nula u suprotnom slučaju.

8.3 Tipovi posmatrača

Sva sredstva posmatranja ili mjerenja informacionih tokova u računarskim sistemima mogu se podijeliti na:

- programska,
- mikroprogramska i
- instrumentalna.

8.3.1 Programski posmatrači

Programski posmatrač je specijalni program ili kompleks programa, instaliran u računarski sistem čije se karakteristike mjere. Saglasno, predloženoj koncepciji, posmatrač postaje posrednik između komponenti koje posmatra i samog računarskog sistema. Posmatrati to znači međusobno djelovati. Kao posljedica toga se javlja konkurencija za resursima sistema između posmatrača i programa koji čine radno opterećenje. Na taj način, programski posmatrač može izobličiti ili promijeniti ponašanje sistema koji mjeri. Međutim sam po sebi taj fakt nije opasan. Opasna su samo izobličenja koja se ne mogu detektovati i izmjeriti.

U toku perioda mjerenja programski posmatrač periodično prekida rad računarskog sistema da bi izvršio mjerenje i skupljanje podataka. To radi ili pomoću programskih prekida [65] ili pomoću povremenih prekida [66]. U prvom slučaju djeluju tako, da svaki događaj koji se mjeri prati prekid. Pri obradi toga prekida skupljaju se i fiksiraju svi neophodni podaci o stanju sistema. Pri korišćenju tehnike povremenih prekida dešava se sljedeće: umesto toga, da se fiksira svaka izmijena u sistemu posmatrač fiksira njeno stanje kroz određeni ili slučajni interval vremena. Rad sistema se prekida i u spoljašnjoj memoriji se zapisuje (fiksira) njeno stanje bez obzira da li je u toku tog intervala bilo kakvih izmijena ili ne. Kako u prvom, tako i u drugom slučaju fiksirani tj. zapisani podaci, po pravilu, odnose se na punjenje instrumentalnog nivoa sistema a rijetko njegovog sistemsko-softverskog nivoa [29], [30], [33].

Pomoću *tehnike programskih prekida* dostupna je bilo koja od četiri oblika mjerenja: trase, relativna aktivnost, frekvencija i raspodjela. Međutim, njihova primjena je povezana sa ozbiljnim teškoćama. Prvo, to je unošenje ili uvođenje u sistem i mijenjanje postojeće softvera. Izmijena oblasti programskih prekida može narušiti cjelovitost sistema. Zato, da bi takvo uvođenje prošlo uspješno, potrebno je najintimnije poznavanje sistema i pažljivo testiranje sistema poslije unesenih izmijena. Ovdje se javlja još jedan problem, naime, proizvođači računarskih sistema se staraju da takve informacije i sistemu postanu nedostupne korisnicima.

Implementacija prekida može biti konstantna tj. fiksirana još prilikom kompilacije programskog posmatrača, a može biti generisana po potrebi. U prvom slučaju obično se primjenjuje neki prekidač, koji dozvoljava da se tokom svakog konkretnog izvršavanja maskiraju neželjeni prekidi. Taj prekidač može biti kako programski tako i hardverski. U drugom slučaju, prekidi se implementiraju samo za vrijeme izvršavanja i samo ti, koji odgovaraju događajima interesantnim za istraživanje.

Tehnika *povremenih prekida* primjenjuje se za mjerenja relativne aktivnosti i frekvencije događaja. Tačnost mjerenja određuje se frekvencijom prekida u vremenu i karakteristikama procesa koji se mjeri. Što je duži interval mjerenja, tim su ređi povremeni prekidi, i tim manji će biti dodatni rashodi mjerenja. Na taj način, povremeni prekidi ne samo da zahtijevaju neznatne izmijene programske podrške sistema, nego pri dovoljno dugom intervalu mjerenja imaju male dodatne gubitke. Mogućnost ovog tehničkog prilaza je ograničena: on može dati samo statistički srednje podatke, tako da može dopustiti različite fluktuacije u ponašanje sistema, koje mogu nastati između dvije uzastopne tačke povremenih prekida.

Jedan od nedostataka programskih posmatrača, koji se koriste u distribuiranim računarskim sredinama, ogleda se baš u primjeni tehnike prekida. Ova tehnika, za

pravilnu registraciju distribuiranih aktivnosti, zahtijeva centralizaciju upravljanja, što je teško sprovodljivo ili je nemoguće kod razmatrane klase računarskih sistema. Može se izdvojiti još jedan nedostatak ovog načina posmatranja rada programa, a to je značajne teškoće pri izmijeni skupa posmatranih aktivnosti.

Za izgradnju distribuiranih posmatrača obično se koristi druga tehnika. Ovdje je primijenjena i isprobana tehnika zasnovana na prevlađivanju barijera zaštite sistema ili mjerenja pri prolasku kroz nivo zaštite sistema. Njegova primjena ne zahtijeva toliko detaljne informacije o sistemu, kao u slučaju programskih prekida, ali on čuva sva dostignuća ovog prilaza. Idealno za njegovu primjenu zgodna je metodologija objektno-orjentisanog prilaza.

8.3.2 Mikroprogramski posmatrači

Drugim moćnim sredstvom realizacije posmatrača javlja se mikroprogramiranje. S mikroprogramskog nivoa dostupni su takvi indikatori aparature, koji su sa gornjih nivoa nedostupni. Međutim, primjena tehnike mikroprogrmiranja ima dva ozbiljna nedostatka:

- prvi problem se ogleda u korišćenju specijalne i skupe aparature,
- drugi problem se ogleda u tome da je događaje, koji nastaju u sistemu na vrlo niskom nivou, teško prenijeti u događaje višeg nivoa, na kojem obično rade programeri. Ovdje je situacije slična onoj koja se javlja pri realizaciji jezika programiranja; kako retranslirati greške koje su proizašle na nivou komandi mašine na nivo operatora tog novog jezika.

Kod ove tehnike postoje i druga slaba mjesta, ali ona ima značajnih prednosti opisanih u literaturi [67].

8.3.3 Hardverski posmatrači

Ovi tipovi posmatrača se mogu podijeliti na unutrašnje i spoljašnje. Tako npr. hardver mnogih savremenih računarskih mašina ima upravljačke i signalne linije, u okviru glavne magistrale, pomoću kojih se kontroliše ponašanje sistema [68], [69]. Spoljašnji hardverski posmatrač priključuje se na određene tačke sistema, detektuje signale na njenim linijama, obrađuje ih i zapisuje kod sebe, izvan mjerenog sistema. Hardverski posmatrač predstavlja savršeno autonoman sistem, koji ne zahtijeva nikakvu pomoć od sistema koji se mjeri. On se praktično ne miješa u njen rad, a pokušava da ne izmijeni njeno ponašanje.

Tehnika posmatranja može biti izgrađena bilo na neposrednom mjerenju električnih parametara odgovarajućih linija, bilo priključenjem uz pomoć interfejsa na rezervne linije, ili na primjeni unutrašnjih hardverskih elemenata [21], [69]. Obrada skupljenih informacija može biti programska ili hardverska, može biti prikupljena poslije događaja a može se snimati i u realnom vremenu.

8.4 Karakteristike mjernih mogućnosti posmatrača

Mjerne mogućnosti ma kog posmatrača određuju tehnika mjerenja i korišćeni posmatrač. Oni su ograničeni sa jedne strane zakonima fizike, a sa druge strane postojanjem povratne sprege između rada mjerenog sistema i rada posmatrača. Mjerne mogućnosti posmatrača moguće je okarakterisati sljedećim karakteristikama:

- *Domen posmatranja* se definiše klasom aktivnosti koje je posmatrač sposoban fiksirati i izmjeriti. Tu karakteristiku umnogome određuje brzina procesora, ali ne samo ona. Domen posmatranja zavisi od nivoa, na kojem se izvodi mjerenje. Taj nivo se određuje sljedećim razmatranjem. Pojavu distribuiranosti u računarskom sistemu karakterišu: pojava konflikta u dostupu djeljivim resursima i kašnjenja sinhronizacije. To dovodi do nelinearnog sniženja produktivnosti. Odavde slijedi, da nivo distribucije u sistemu određuje taj najniži nivo, počev od kojeg se javljaju gubici produktivnosti zbog ovih uzroka. U ovom slučaju ti gubici proizilaze na nivou sistema veza fizičke sredine, arbitražnih algoritama, distribucije resursa u logičkoj sredini i toka zahtjeva prema računarskoj sredini od strane programa. Nivo distribuiranosti u konkretnom računarskom sistemu određuje nivo analize računarskog sistema pri ocjeni produktivnosti. Zato će u razmatranoj klasi računarskih sistema od posebnog značaja biti nivo programskih procesa i nivo sistema veza u fizičkoj sredini. Ovi nivoi su dostupni samo programskom posmatraču.
- *Brzina mjerenja* je maksimalna frekvencija sa kojom je posmatrač sposoban da raspozna i izmjeri događaje. Brzina mjerenja posmatrača je od posebnog značaja pri mjerenju. U distribuiranom računarskom sistemu čak vrlo brz posmatrač, ali sekvencijalan, tj. onaj koji ima mogućnost fiksiranja jednog mjerenja za jedan put, ne može fiksirati jednovremenu pojavu događaja. Pokušaji imitacije distribuiranosti na sekvencijalnim mašinama, uz primjenu tehnike dijeljenja vremena i sl., sa ciljem izbjegavanja jednovremenih događaja, oštro snižava efikasnost mjerenja i opasni su zbog uticaja sistemskog softvera eksperimentalnog sistema. Ovo može uticati na izobličenje skupljenih podataka i neadekvatno predstavljanje paralelizma. Osim toga, postojanje u sistemu nekoliko nezavisnih tokova upravljanja (prostorno-vremenskih koordinatnih sistema), pretpostavlja i nezavisnost posmatranja.
- *Kapacitet memorije posmatrača* je veličina bafera rezervisanog za smještaj skupljenih podataka. To je vrlo važna karakteristika, tako da od nje indirektno zavisi mogućnost obrade skupljenih podataka. Kod distribuiranog sistema očekuje se velika količina podataka, rezultata mjerenja. Zbog toga, da bi se smanjili gubici resursa sistema na rad posmatrača, a tim samim oslabilo miješanje u rad sistema, distribuiranom posmatraču je potreban određeni obim memorije, podržan adekvatnim mehanizmom prenosa njegovog sadržaja na uređaje spoljašnje memorije, van tog dijela sistema koji se koristi programom.
- *Dopuštena tačnost*, se određuje maksimalnim intervalom između taktova časovnika posmatrača, od kojih on dobija informaciju o vremenu kad je nastupilo vrijeme tog ili onog događaja, koliko se vremena dešavala ta ili ona aktivnost. Upravo te karakteristike ograničavaju dostignutu tačnost u mjerenjima, gdje je neophodno vrijeme. Nju određuju mogućnosti tajmera, dostupnog posmatraču.

8.5 Poređenje hardverskih i softverskih posmatrača

Kao što je već naglašeno, sva sredstva posmatranja kod računarskih sistema mogu se podijeliti na hardverska i softverska. Samo pomoću programskih ili samo pomoću hardverskih posmatrača mjerenja na svim nivoima računarskog sistema nijesu moguća. U nekim slučajevima ovi posmatrači dopunjuju jedan drugog a drugim su konkurenti jedan drugom.

8.5.1 Domen posmatranja.

Mnoge aktivnosti u sistemu se mogu posmatrati kako pomoću programskih tako i pomoću hardverskih sredstava. U vezi sa tim njihovi domeni se bitno razlikuju.

Programski posmatrač ima pristup samo toj memoriji, koja može biti adresirana pomoću komandi. Taj posmatrač može posmatrati samo događaje nivoa sistema komandi i višeg, koji se sprovode bilo prenosom upravljanja (tj. prekidima) po poznatoj adresi, bilo pamćenjem identifikacione informacije u programski dostupnu memoriju, čija se sadržina kasnije može obraditi. Sve ovo se može primijeniti i kod mikroprogramskih posmatrača. Isto tako čitav niz podataka za identifikaciju u sistemu, takvih kao što su imena procesa, skupovi podataka i sl. dostupni su samo programskim posmatračima. Ovi podaci su neophodni za identifikaciju mjerenih objekata, praćenja aktivnosti programske podrške, pogotovu zato što programi po pravilu nemaju stalno fiksirano mjesto u operativnoj memoriji sistema.

Hardverski posmatrači imaju pristup do ma kog elementa unutrašnje registarske memorije sistema, bez bilo kakvih pomoćnih aktivnosti od strane mjerenog sistema. Međutim, pristup sadržaju operativne memorije on ima samo tada, kada se on prenosi između memorije i procesora. Strogo govoreći to je tako i kod programskih posmatrača, ali ovi mogu uzeti informaciju u isto vrijeme dok ista može proći pored davača hardverskog posmatrača. U tom smislu hardverski posmatrač predstavlja pasivnog posmatrača. On može razlikovati događaje, povezane sa radom programske podrške samo u tom slučaju ako oni sprovode prenos upravljanja po apsolutnim tz. fizičkim adresama ili uz pomoć prekida.

8.5.2 Brzina mjerenja.

Brzina sa kojom hardverski posmatrač može primati događaje je vrlo velika i ograničena je samo fizičkim mogućnostima davača i elektronskih kola. Brzina programskih posmatrača određuje se brzinom procesora, na kojem on radi, i veličinom dodatnih gubitaka za posmatranje. Na taj način najmanji događaj koji on može posmatrati je izvršavanje odvojene komande. Maksimalna brzina mjerenja ograničena je maksimalnim vremenom izvršavanja komandi. U isto vrijeme, što je veća brzina mjerenja, veći su i dodatni troškovi mjerenja.

Postojeći programski posmatrači mogu primati i obrađivati događaje samo sekvencijalno, kako slijede, oni mogu zaustaviti sva izračunavanja za vrijeme neophodno za prikupljanje podataka. Isto tako, veliki uticaj na sakupljene podatke mogu ispoljiti strategije raspodjele resursa u sistemu, npr. procesora [76].

Hardverski posmatrači mogu posmatrati istovremeno nekoliko događaja, ali su oni suštinski ograničeni u širini primljene informacije, tj, brojem bita koje mjerni uređaj može zapamtiti u istom trenutku.

8.5.3 Dopuštena tačnost

Kako kod programskih tako i kod hardverskih posmatrače dozvoljena sposobnost je ograničena samo fizičkim karakteristikama tajmera, koji koriste u radu. Apsolutne prevage ni jedna od metoda nema. Zato izbor tipa posmatrača opredeljuju ciljevi i nivo mjerenja, koji je značajan za određenu konkretnu primjenu. Dopuštenu tačnost programskog posmatrača u distribuiranoj sredini određuje najsporiji računar u sistemu.

8.6 Zaključci

Ma koliko izmjerene performanse mogu biti vrijedne, uopšteno gledano, ovom metodom se ne može doći do fleksibilnosti koja je tipična za imitacioni metod. Na primjer, imitacioni model omogućava lako da se posmatraju efekti izmijene različitih parametara konfiguracije (npr. broj autonomnih računara i njihove brzine, propusna moć računarske mreže, kapacitet memorije i tome slično), dok takve izmijene za vrijeme mjerenja na realnom sistemu nijesu moguće, ili su skupe, teško sprovodljive i vremenski dugotrajne. Kombinovanjem mjerenja na realnom distribuiranom sistemu ili nekom kontrolnom (eksperimentalnom) sistemu, specijalno izgrađenim za potrebe istraživanja, sa imitacionim modeliranjem dobijaju se najbolji rezultati. Postoji više situacija gdje imitacioni model može biti dopuna ili proširenje rezultata dobijenih mjerenjem.

GLAVA

9. PRIMJENA IMITACIONOG MODELIRANJA ZA ANALIZU JEDNOG DISTRIBUIRANOG SISTEMA

9.1 Predmet analize

Za analizu produktivnosti specificirane klase distribuiranih računarskih sistema (poglavlje 2.1), u ovom radu je predložen i razrađen metod imitacionog modeliranja, označen kao imitacioni model sa porukama. Metod je iskorišćen za analizu hipotetičkog distribuiranog računarskog sistema Prirodno-matematičkog fakulteta u Podgorici. Ovaj se distribuirani sistem sastoji od autonomnih personalnih računara, međusobno povezanih lokalnom računarskom mrežom. Autonomni računari mogu biti različite konfiguracije, performansi i mogu imati različite uloge u sistemu, tako da to mogu biti: radne stanice, fajl-servere, servere baza podataka, komunikacione servere, print-servere i sl. U sistemu može biti istovremeno prisutno i više mrežnih i lokalnih operativnih sistemi kao što su: Windows, Unix, Linux ili Novell NetWare, pa se za komunikaciju i razmjenu podataka mrežom, mogu koristiti različiti skupovi protokola npr.: NetBEUI, TCP/IP ili IPX/SPX.

Cilj analize ovog hipotetičkog distribuiranog sistema je razrada, izgradnja i istraživanje različitih konfiguracija njegove hardverske i sistemsko-softverske strukture, kao i opšta ocjena kvaliteta primijenjenih projektnih rješenja distribuiranih aplikativnih programa i korišćene računarske mreže. Za postizanje ovog cilja potrebno je:

1. Razraditi odgovarajući ***imitacioni model sa porukama***, koji će na adekvatan način predstaviti distribuiranu obradu podataka u ovom sistemu i hardversko-softverska sredstava za podršku komunikacija;
2. Obezbijediti ***modularni princip*** izgradnje imitacionog modela i strukturni način procjene dobijenih rezultata, kako bi se olakšao sam proces imitacionog modeliranja i omogućio njegov razvitak;
3. Izgraditi eksperimentalni fragment hipotetičkog distribuiranog sistema, koji će na reprezentativan način predstavljati ovaj sistem i koji će poslužiti kao ***kontrolni distribuirani sistem*** za provjeru adekvatnosti i pravilnosti modela;
4. Razraditi i proanalizirati ***sredstva ocjene efikasnosti*** funkcionisanja ovog hipotetičkog sistema i iskoristiti ih kao ulazne parametre imitacionog modela sa porukama.

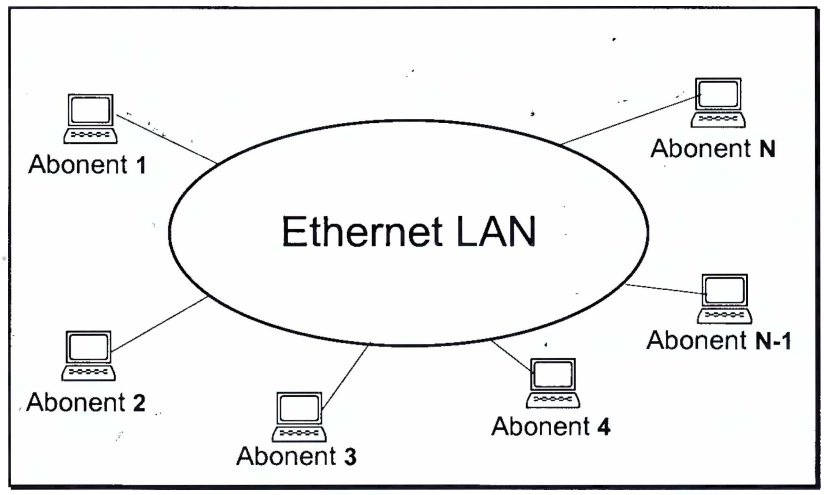
Osnovna pitanja i zadaci, na koje analiza ovog distribuiranog računarskog sistema, treba da da odgovore su:

- ocjena produktivnost ovog hipotetičkog distribuiranog sistema,
- ocjena opšte propusne sposobnosti komunikacione sredine,
- ocjena efikasnosti korišćenja komunikacionih kanala,
- određivanje optimalne distribucije podataka i aplikativnih procesa u sistemu,
- verifikacija protokola međudejstava distribuiranih aplikativnih procesa,
- ocjena opterećenja pojedinih uređaja, koji ulaze u sastav sistema,
- identifikacija zastoja i konflikata u radu algoritama i sistema u cjelini,
- određivanje "uskih" mjesta u sistemu,
- određivanje redundantnosti u sistemu itd.

9.2 Arhitektura kontrolnog distribuiranog sistema

Za potrebe istraživanja u ovom radu izgrađen je reprezentativni fragment hipotetičkog distribuiranog sistema Prirodno-matematičkog fakulteta u Podgorici, koji je označen kao *kontrolni distribuirani sistem*. Arhitekturu kontrolnog distribuiranog sistema (slika 9.1) čini kompleks teritorijalno distribuiranih strukturnih elemenata tj. *abonenata* sistema koje predstavljaju autonomni personalnih računara različite konfiguracije, povezanih jedinstvenom računarskom mrežom za prenos podataka. Kontrolni distribuirani sistem karakteriše:

- postojanje određenog konačnog skupa hardverskih i softverskih komunikacionih sredstava (LAN mreža) i abonenata (autonomnih računara),
- raznovrsnost korišćenih softverskih i tehničkih sredstava,
- distribuirani karakter čuvanja i obrade informacija u sistemu,
- jedinstvo arhitekturnih rješenja svih sastavnih komponenti,
- mogućnost rekonfiguracije pojedinih komponenti ili sistema u cijelini.



Slika 9.1 Arhitektura kontrolnog distribuiranog sistema.

Kontrolni distribuirani sistem predstavlja kompleks hardverskih i softverskih sredstava, specijalno razrađenih za:

- obezbeđenje distribuirane sredine u kojoj se mogu izvršavati eksperimentalni distribuirani aplikativni programi,
- posmatranje, mjerenje i skupljanje podataka o ponašanju ovih distribuiranih programa,
- evidentiranje i mjerenje dodatnih gubitaka i smanjenja performansi, prouzrokovanih unošenjem hardvera i softvera mjernog instrumenta ili posmatrača u sistem.

9.2.1 Otvorena arhitektura kontrolnog sistema

Od posebne je važnosti činjenica, da je arhitektura kontrolnog distribuiranog sistema otvorena za proširenja i izmjene konfiguracije sistema. To znači da su hardverske-softverske komponente tog sistema i način njihovog povezivanja bile izabrane tako, da je bila moguća:

- Prosta izmijena strukture i konfiguracije hardvera i sistemskog softvera;

- Izmijene strategije upravljanja računarskom mrežom i sistemom u cjelini i načina distribuciju logičkih resursa, korišćenih u programskom obezbeđenju njegovih podsistema;
- maksimalna mogućnost, izbora optimalnog nivoa detaljizacije, sa tačke gledišta analize efikasnosti funkcionisanja sistema, izmjerenih karakteristika programa i tačnosti mjerenja,.

Zahtjevi koji se tiču jednostavnosti izmijene strukture i cjelovitosti mjerenja informacionih procesa u sistemu, odnose se na razlike realnog kontrolnog sistema od hipotetičkog distribuiranog sistema koji se posmatra. Kontrolni sistem mora biti otvoren za sva razumna proširenja i eksperimente. Razlozi ovome su što instrument eksperimentatora ne treba da bude sakriven od objekta eksperimenta. S tim u vezi radne stanice i serveri imaju mogućnost izmjene: kapaciteta i brzine operativne memorije, procesora (u određenom dijapazonu), kontrolera diskova i samih diskova, mrežnih kartica, protokola razmjene podataka, upravljačkih programi i sl.

9.2.2 Osnovni hardver kontrolnog sistema

Kod izgradnje kontrolnog distribuiranog sistema, kao procesori su korišćeni Pentium procesori druge generacije 133, 166 i 200 MHz firme Intel, sa PCI magistralom. Za disk podsisteme radnih stanica korišćen je IDE standard, a na serverima su obično bili instalirani SCSI diskovi. Abonenti hipotetičkog kao i kontrolnog sisťema su bili povezani lokalnom računarskom mrežom Ethernet tipa sa propusnim opsegom 10 Mbps i 100 Mbps. Taj izbor se opravdava širokom rasprostranjenošću ovog tipa lokalnih mreža, i postojanjem široke lepeze serijskih modula za ovaj tip računarskih mreža. Zahvaljujući magistralno modularnom principu organizacije kontrolnog distribuiranog sistema otvorena je mogućnost za:

- jeftino i prosto priključenje dodatnih komponenti i perifernih uređaja,
- fleksibilnu rekonfiguracije sistema,
- jednostavnost izmjene broja abonenata, a samim tim i opterećenja sistema,
- fleksibilnu izmijena strategija arbitraže opsluživanja u klijent/server okruženju,
- otvorenost sa tačke gledišta mjerenja informacionih tokova u sistemu.

9.2.3 Sistemsko-softverska sredina kontrolnog sistema

Operativni sistemi radnih stanica i servera izgrađuju izvršnu distribuiranu sredinu kontrolnog sistema. Svaka lokalna operaciona sredina je instalirana na odvojenom autonomnom računarima. Tokom jedne serije eksperimenata svi lokalni operativni sistemi i protokoli razmjene podataka su bili identični. Izuzetak je bio mrežni operativni sistemi instaliran na serveru (serverima). Izvršna distribuirani sredina uključuje u sebi sve funkcije mrežnih i lokalnih operativnih sistema. Nju, kako je navedeno u poglavlju 5.5.1, čine logički resursi, kojima se procesi aplikativnog programa obraćaju radi zadovoljenja svojih potreba.

Tokom nekoliko serija eksperimenata, ovog istraživanja, na serverima i radnim stanicama uglavnom su bili instalirani operativni sistemi: Windows 95/98, Windows NT/2000 i Novell NetWare. Osnovno opredeljenje za izbor ovih operativnih sistema odredili su ciljeve istraživanja, ali i sljedeće činjenice:

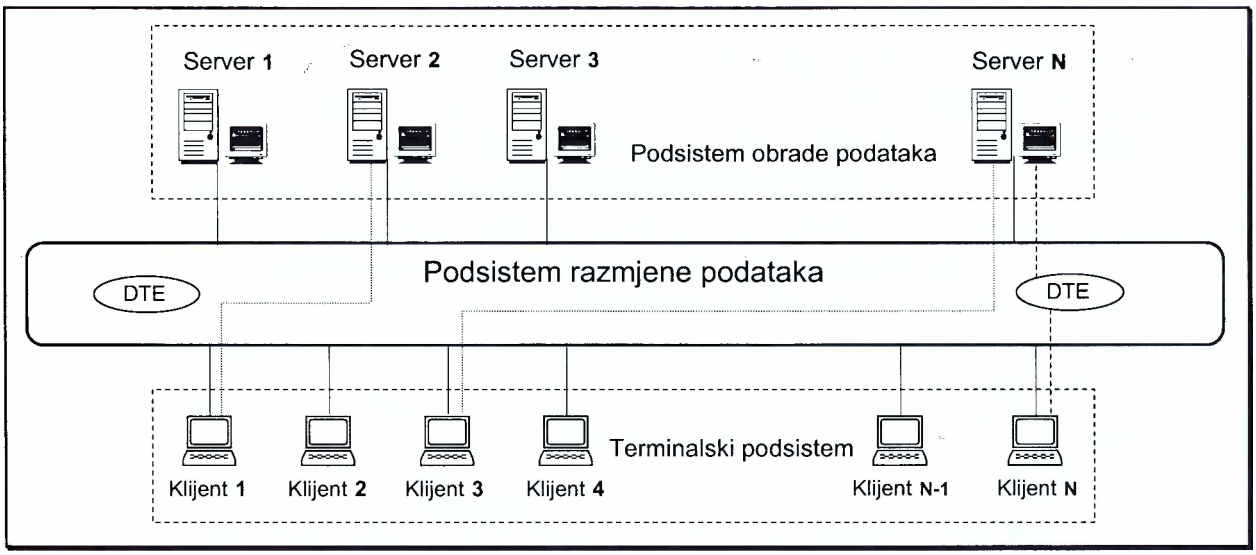
- mogućnost da se u njihovom okruženju izvršavaju distribuirani aplikativni programi, napisani na proceduralno-orijentisanim jezicima,
- komforan način posmatranja, mjerenja i sakupljanja podataka o izvršavanju distribuiranih programa,
- relativna nezavisnost ponašana distribuiranog aplikativnog programa od manjih izmjena konfiguracije i karakteristika fizičke sredine,
- široke mogućnosti debagiranja aplikativnih programa koji se izvršavaju u ovom okruženju.

9.3 Koncepcijski (idejni) model sistema

Hipotetički distribuirani računarski sistem Prirodno-matematičkog fakulteta u Podgorici mora obezbijediti korisnicima, odnosno abonentima, širok spektar usluga i dozvoliti izgradnju čitavog niza automatizovanih podsistema distribuirane obrade informacija, među kojima treba izdvojiti:

- distribuirane baza podataka,
- prenos datoteka između različitih abonenata mreže,
- elektronsku poštu sa razmjenom informacija između korisnika,
- interaktivni rad u grupi većeg broja korisnika,
- sisteme distribuiranog izračunavanja i distribuirane obrade podataka uopšte itd.

Radi obezbjeđenja gornjih zahtijeva funkcionalna ili *logička struktura* ovog distribuiranog računarskog sistema (slika 9.2) mora uključiti u sebi tri klase logičkih modula:



Slika 9.2 Funkcionalna struktura posmatranog distribuiranog sistema.

1. Serverski moduli, koji imaju mogućnost obrade podataka (informacija), realizujući glavnu ciljnu funkciju sistema tj. servisiranje (zadovoljenje) zahtjeva korisnika, kao što su: izračunavanja, pretraživanja, čuvanje i obrada podataka;
2. Klijentski moduli, obezbjeđuju korisniku tj. aplikativnom procesu formiranje zahtijeva za obradu, pristup resursima serverskih modula i prezentaciju rezultata obrade u za korisnika pogodnom i komfornom obliku;

3. Komunikacioni moduli ili moduli interakcije i povezivanja, koji obezbjeđuju povezivanje klijentskih i serverskih modula a takođe i klijentskih modula između sebe, po odgovarajućoj proceduri razmjene podataka.

Skupovi serverskih modula, klijentskih modula i komunikacionih modula obrazuju odgovarajuće logičke podsistema: podsistem obrade podataka, terminalski podsistem i podsistem razmjene podataka.

Principijelnim funkcionalnim zahtjevom javlja se potreba za neograničenim pristupom ma kojeg klijentskog modula komunikacionom podsistemu i podsistemu za obradu podataka. Ograničenja mogu odrediti samo specifičnost tehnologije i tehnički i organizacione karakteristike distribuiranog sistema. Npr., administrator sistema može ograničiti određenim grupama korisnika, recimo studentima, pristup nekim od servera.

Predstavljena na slici 9.2 funkcionalna struktura može se smatrati univerzalnom za posmatranu klasu distribuiranih sistema. Međutim njegova fizička struktura može imati veći broj modifikacija u zavisnosti od načina distribucije funkcija između određenih tehničkih sredstava, oblika i mogućnosti primjene tehničkih sredstava, specifičnih karakteristika, korišćenih pri projektovanju konkretnog sistema.

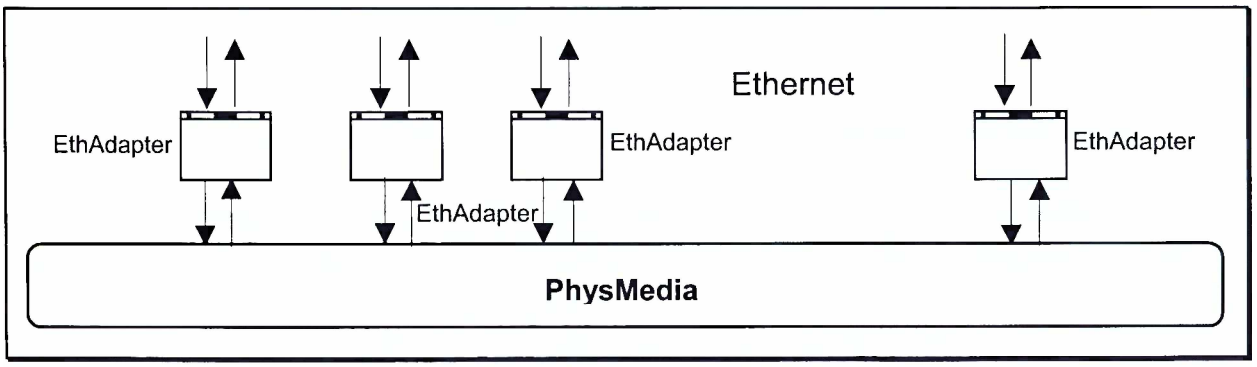
9.4 Primjeri izgradnje biblioteke modela

Tokom realizacije modela posmatranog sistema, neophodno je bilo najprije izgraditi imitacione modele tipičnih komponenti sistema. Ovdje je kao primjer dat model Ethernet protokola IEEE 802.3, kao i modeli aplikativnih procesa klijenta i servera.

9.4.1 Model protokola fizičkog nivoa standarda IEEE 802.3

Standard IEEE 802.3 određuje protokol fizičkog nivoa (nivo 1 modela ISO OSI) i podsloja MAC kanalskog nivoa (nivo 2 modela ISO OSI). Određuje se procedura pristupa fizičkom medijumu i prijem/slanje nizova bitova u računarsku mrežu, magistralne topologiji i sa načinom pristupa medijumu CSMA/CD.

Model Ethernet protokola standarda IEEE 802.3 predstavlja distribuirani program sa ***n*** ulaznih i ***n*** izlaznih bafera i ima strukturu prikazanu na slici 9.3.



Slika 9.3 Model protokola IEEE 802.3.

Protokol fizičkog nivoa obezbjeđuje kontrolu ispravnosti pristiglih ili poslatih okvira uz pomoć kontrolne sume (metoda CRC32) i konstatuje stanja fizičkog medijuma za prenos, koji može biti: slobodan, da je prenos u toku tj. medijum je zauzet i kolizija

kada je došlo do miješanja okvira. Brzina prenosa je zavisila od tipa mrežnog kontolera radne stanice i kretala se od 1 Mbps do 100 Mbps. Protokol podsloja MAC kanalskog nivoa obezbjeđuje prenos okvira na fizički medijum sa ponovnim slanje u slučaju kolizije ili grešaka tokom prenosa, prijem okvira i obavještava više slojeve o rezultatima prenosa.

Proces ***PhysMedia***, na slici 9.3, imitira protokol fizičkog nivoa standarda IEEE 802.3 i medijum za prenos podataka sa magistralnom topologijom. Proces ostvaruje sekvencijalni prenos nizova bita tj. prenos okvira u kanal sa zadatom brzinom. Istovremeno informiše proces ***EthAdapter*** o stanju medijuma za prenos podataka, koje može biti: *slobodan*, *prenos* (medijum je zauzet), *kolizija* ili *kvar*.

Proces ***PhysMedia*** radi u režimu obrade zahtijeva, tj. proces obrađuje poruke pristigle u ulazni bafer, ako takvih poruka nema proces čeka njihov dolazak. Obrada poruke ostvaruje se po sljedećem algoritmu:

1. Čekanje i prijem poruke iz ulaznog bafera;
2. Ako u ulaznom baferu ima još poruka prelazi se na tačku 4;
3. Stanje "*prenos*". Izračunava se vrijeme prenosa podataka po formuli: $T_{tr} = (L_{data} + L_{hdr})/V_{tr} + T_{set}$ i vrši se slanje službene poruke *BUSY* sa vremenom oslobađanja sredine koje je jednako zbiru T_{tr} i tekućeg vremena. Ostvaruje se kašnjenje u trajanju T_{tr} i prelazi se na tačku 5;
4. Stanje "*kolizija*". Vrši se izračunavanje vremena rješavanja kolizije po formuli: $T_{cl} = L_{jam}/V_{tr} + T_{set}$ i šalje se službeno saopštenje *COLLISION* sa vrijednošću vremena oslobađanja medijuma za prenos, koji je jednak zbiru T_{cl} i tekućeg vremena. Ostvaruje se kašnjenje u trajanju T_{cl} i prelazi se na tačku 5;
5. Stanje "*medijum slobodan*". Vrši se čišćenje bafera od zaostalih poruka i šalje se poruka *FREE* da je fizički medijum slobodan a zatim se prelazi na tačku 1.

Proces ***EthAdapter*** imitira rad uređaja za prenos podataka koji se naziva mrežni ili Ethernet adapter, saglasno standardu IEEE 802.3 i IEEE 802.2. Proces ostvaruje prenos okvira protokola višeg nivoa i prijem okvira sa podacima od protokola nižeg nivoa. Takođe vrši provjeru CRC sume paketa i odbacuje ga ako su podaci preneseni sa greškom.

Za vrijeme rada proces ***EthAdapter*** se može naći u stanjima "*slobodan*" i "*prenos*". Početno stanje je "*slobodan*".

Algoritam rada procesa je sljedeći:

(**A**) Proces se nalazi u stanju "*slobodan*":

1. Čekanje i prijem poruke od procesa ***PhysMedia*** (osluškivanje fizičke sredine);
2. Analiza dolazne poruke: ako je poruka *FREE*, prelazi se na tačku 4.
3. Ako je došla poruka *BUSY* ili *COLLISION*, čeka se vrijeme oslobađanja medijuma i prelazi na tačku 1.
4. Došla je poruka tipa *FRAME* (okvir sa podacima) ista se prenosi protokolu višeg nivoa.
5. Ako još poruka od procesa ***PhysMedia*** čeka onda se prelazi na tačku 1.
6. Provjera postojanja poruke od procesa višeg nivoa. Ako ima poruke briše se brojač broja propitivanja i predaje se okvir na fizički sredinu i prelazi se u stanje "*prenos*".

7. Prelaz na tačku 1.

(B) Proces se nalazi u stanju "*prenos*":

1. Čekanje i prijem poruke od procesa ***PhyMedia*** (osluškivanje fizičke sredine);
2. Analiza dolazne poruke:
   a. ako je poruka *FREE*, vrši se prenos okvira na fizičku sredinu;
   b. ako je poruka *BUSY* čeka se vrijeme oslobođenja medijuma i prelazi na tačku 1;
   c. ako je poruka *COLLISION*, čeka se određeno vrijeme da se oslobodi medijum, inkrementira se brojač propitivanja za 1, ako broj propitivanja N_p ne prelazi maksimalni broj propitivanja izračunava se vrijeme kašnjenje T_w [ms] po formuli: $T_w = \begin{cases} rand(0,2^{N_p}).N_p \le 10 \\ rand(0,2^{10}).N_p > 10 \end{cases}$ i prelazi se na tačku 1, inače se višem nivou prenosi poruka FAIL i prelazi se u stanje "*slobodan*";
   d. ako je poruka tipa FRAME (okvir podataka) ako to nije sopstveni okvir on se prenosi procesu višeg nivoa i prelazi se na tačku 1, inače se prelazi u stanje "*slobodan*".

Proces ***EthAdapter*** modelira i oštećenje podataka pri prenosu po fizičkom kanalu. Ako je vjerovatnoća slučajne greške u jednom bitu *PbadBit*, onda se vjerovatnoća pogrešnog prenosa okvira dobija po formuli: $PBadFrame = 1{,}0 - (1{,}0 - PBadBit)^{L_{data}}$.

Ovaj proces je takođe sposoban modelirati sniženje produktivnosti kanala za prenos podataka. Ako je došla poruka tipa SPEED sa poljem *Percent*, onda se izračunava nova brzina prenosa po sljedećoj formuli: $V_{tr} = V_{tr} * (1 - Percent / 100)$.

Tablica 9.1 *Parametri modela protokola standarda IEEE 802.3*

parametar	opis	vrijednost
V_{tr}	Nominalna brzina prenosa podataka [Mbps]	100
T_{set}	Interval vremena između dva okvira [ms]	9,6
L_{data}	Veličina okvira [byte]	64-1518
L_{hdr}	Veličina zaglavlja okvira [bit]	102
L_{jam}	Veličina okvira kojim se obavještava o koliziji [bit]	512
rand(a,b)	Ravnomjerna raspodjela slučajnih brojeva na [a,b]	a-b
PbadBit	Vjerovatnoća pojavljivanja greške u jednom bitu	$10e^{-6}$
Mem	Veličina bafera u adapteru [byte]	1024
Attempt	Maksimalni broj ponavljanja	16

U toku rada proces ***Ethernet*** skuplja sljedeću statističke podatke:

- U datoteku *PhysLayer.dat* sakupljaju se informacije o ukupnom prometu kroz kanal, ukupnom opterećenju u bajtima.
- U datoteku *PhysError.dat* unose se informacije o isporučenim i odbačenim paketima na kanalskom nivou.

9.4.2 Modeli protokola transportnog nivoa TCP

Protokol transportnog nivoa zadužen je za siguran prenos podataka, (sloj 4 modela ISO OSI), koji je orjentisan na uspoastavljanju logičke veze. Sa protokolima nižih slojeva transportni sloj komunicira po dupleksnom kanalu (slika 9.5).

Protokol TCP ostvaruje: prijem korisničke poruke sa višeg nivoa, baferizuje je, dijeli na pakete fiksne dužine, vrši sekvencijalni prenos paketa sa dobijanjem potvrde i obradu *timeout* prenosa. On takođe, vrši prijem paketa od protokola nižeg nivoa sa kontrolom ispravnosti poretka pristiglih paketa, sastavlja poruku od niza paketa i šalje je protokolu višeg nivoa.

Protokolom prenosa podataka (engl. *Transmission Control Protocol, TCP)* ostvaruje se multipleksiranje i demultipleksiranje primljenih/poslatih tokova paketa, tj. podržava istovremeno nekoliko logičkih veza.

Model protokola TCP realizuje se u vidu procesa ***TCP***. Proces ***TCP*** obezbjeđuje ispunjavanje sljedećih aktivnosti:

- Povezivanje;
- Potvrdu uspostavljanja i raskidanja logičke veze;
- Prijem/slanje paketa i potvrda;
- Obradu timeout-a;
- Ponavljanje operacija.

Proces podržava specijalnu strukturu podataka za čuvanje informacija o aktivnim logičkim vezama, koje se čuvaju u bloku kontrole prenosa (Engl. *Transmission Control Block, **TCB***). Svaki od njih ima jedinstven broj (broj porta). Maksimalan broj portova (istovremeno podržanih veza) zadaje se parametrom *TCP_MAX_N_TCB*. Dupleksnu logičku vezu izgrađuje proces za prenos podataka i zatvara se poslije dobijanja odgovarajuće komande od protokola višeg nivoa.

Rad procesa ***TCP*** uključuje u sebi sljedeće aktivnosti po etapama:

a) Uspostavu virtuelnog povezivanja:
   a. Po komandi procesa aplikativnog nivoa proces TCP "otvara" aktivne ili pasivne veze;
   b. Ako je uspostavljena pasivna veza, onda poslije dolaska poruke od procesa aplikativnog nivoa sa brojem porta čvora kojem pripada, proces izgrađuje TCP blok i čeka dolazak poruke o uspostavi veze. Poslije dobijanja takve poruke postojeći TCB blok sa odgovarajućom adresom i brojem porta, udaljene mašine, uspostavlja vezu. Primivši potvrdu o uspostavljanju veze obaveštava aplikativni sloj o uspostavi veze. Tom prilikom se šalje broj porta i adresa udaljenoj mašini;
   c. Ako se uspostavi aktivna veza, onda se formira zahtjev za uspostavljanje veze sa odgovarajućim procesom aplikativnog nivoa, adresom udaljene mašine i brojem porta. Poslije prijema potvrde o uspostavi virtuelnog kanala izgrađuje se TCB blok, koji se jednoznačno identifikuje tu vezu i obavještava se proces aplikativnog nivoa o uspostavi virtuelnog kanala. Ako potvrda nije došla posle isteka timeout-a TCP_MAX_CONN_TIME, izvještava se o tome proces aplikativnog nivoa.

b) Prenos podataka po uspostavljenom virtuelnom kanalu:

a. Ako potvrde za sve dijelove datog prozora ne dođu u toku zadatog timeout intervala vremena, prenos se ponavlja (počinjući od tog broja bajta prozora, koji nije bio potvrđen). Odgovarajući izračunati prozor i timeout, proizilazi zbog toga što je prenos bio neuspješan zbog:
   - *THRESHOLD je jednak polovini prozora prethodnog prenosa;*
   - *veličina prozora datog prenosa postavlja se na minimalnu vrijednost TCP_DEFAULT_WINDOW;*

b. Ako prenos nije ostvaren u toku TCP_N_SEND_ATTEMPTS, obavještava se viši sloj o neuspješnom prenosu;

c. Ako je prenos uspješno završen i vratila se potvrda, širina prozora i timeout se izračunavaju po sljedećim pravilima:
   - *TimeOut=min(TCP_UBOUND,max(TCP_LBOUND,TCP_BETTA*SRTT)),*
     pri čemu je SRTT=TCP_ALPHA*SRTT+(1-TCP_ALPHA)*RTT
     TCP_ALPHA=0,8, TCP_BETTA=2,0 (koeficijenti),
     TCP_UBOUND=60 sec. (maksimalni timeout),
     TCP_LBOUND=1 sec. (minimalni timeout),
     RTT je round-trip vrijeme (vrijeme od opravke prozora do dobijanja potvrde za cijeli prozor),
     Početne vrijednosti su SRTT=0, RTT=TCP_UBOUND.

d. Ako je širina prozora prethodnog prenosa bila manja nego THRESHOLD, povećava se širina prozora dva puta, a ako više nije jednaka THRESHOLD onda se uvećava na *TCP_DEFAULT_WINDOW.*

c) Dobijanje podataka po uspostavljenom virtuelnom kanalu.

a. Pri dobijanju podataka po ispostavljenom virtuelnom kanalu, oni se unose u prijemni bafer veličine *MAX_TCO_RECV_BUF_SIZE* i šalje se potvrda aplikativnom procesu sa brojem tekućeg porta o punjenju bafera. Posljednja primljena poruka s brojem tekućeg porta upisuje se u bafer. Na korisnički zahtjev predaje im se cio bafer prijema koji pripada odgovarajućem TCB bloku. Zajedno sa potvrdom o prijemu podataka šalje se veličina slobodnog bafera prijema, u zavisnosti od koga se umanjuje obim prenošenih podataka.

d) Raskidanje virtuelne veze. Raskid uspostavljene veze može nastati u sljedećim slučajevima:

a. Proces aplikativnog nivoa poslije završetka prenosa saopštava procesu TCP, da je neophodno zatvoriti kanal, tada proces TCP šalje poruku o raskidu veze. Na drugom kraju se to dobija i saopštava se procesu aplikativnog nivoa o raskidu veze. Proces aplikativnog nivoa ako više neće da koristi uspostavljeni virtuelni kanal za prenos podataka takođe šalje poruku o raskidanju virtuelne veze, poslije toga se šalje potvrda o raskidanju veze.

b. U slučaju da istekao timeout TCP_IS_ALIVE_TIME a nije došao ni jedan paket za uspostavom veze, tada se vrši pokušaj provjere mogućeg prenosa podataka po uspostavljenom virtuelnom kanalu, ako u toku TCP_N_SEND_ATTEMPTS pokušaja prenosa probnog paketa potvrda nije došla, veza se smatra raskinutom, o čemu se obavještava proces aplikativnog nivoa.

Tablica 9.2. *Parametri modela protokola* ***TCP***

parametar	opis	Vrijed.
TCP_NET_MAX_SEQ_SIZE	Maksimalna veličina segmenta [byte]	1400
MAX_TCP_SEND_BUF_SIZE	Veličina bafera za slanje [byte]	999999
MAX_TCP_RECV_BUF_SIZE	Veličina bafera prijema [byte]	999999
MSL	Vrijeme po isteku kog se uništava TCB	120
DEFAULT_IP_TTL	Broj čvorova poslije čijeg prolaza se poruka gubi	60
TCP_UBOUND	Maksimalni timeout slanja [sec]	60
TCP_LBOUND	Minimalni timeout slanja [sec]	1
THRESHOLD	Granica, poslije koje se prozor slanja uvećava 2 x	65500
TCP_MAX_CONN_TIME	Vreme za koje se veza mora uspostaviti	100
TCP_NULL_WIN_TIME	Timeout ponovnog slanja poruke [sec]	100
TCP_N_SEND_ATTEMPTS	Broj pokušaja neuspješnog prenosa	3
TCP_IS_ALIVE_TIME	Timeout provjere postojanja	60
TCP_MAX_N_TCB	Maksimalni broj portova	100

U toku rada proces ***TCP*** skuplja sljedeću statističke podatke:

- kašnjenje u uspostavljanju veze;
- kašnjenje u prenosu paketa sa podacima
- kašnjenje indikacije prekida veze;
- slanje poruke o uspostavi veze na pogrešnu adresu;
- broj neregularno pristiglih potvrda o uspostavi veze;
- broj grešaka otkrivenih na drugom nivou u paketima sa podacima;
- broj neplaniranih poslatih paketa za raskid veze;
- broj prekida veze zbog narušenja prenosa podataka;
- broj otkaza slanja potvrde za prikidom veze itd.

9.4.3 Modeli protokola aplikativnog nivoa

Kako je već naglašeno, danas se u praksi za izgradnju distribuiranih programa najčešće koristi tehnologija klijent/server. Najčešći su to zahtjevi kojima se klijentski procesi obraćaju fajl-serverima i serverima baza podataka. Saobraćaj generisan od klijenata je krajnje nehomogen i slučajan, kako po količini podataka koji se prenose, tako i po intenzitetu tokova zahtjeva i vremenu njihovog nastajanja.

Proces ***Client*** modelira rad aplikativnog programa abonenta i odgovara aplikativnom sloju ISO OSI modela. Proces ***Client*** komunicira sa procesom transportnog nivoa po dupleksnom kanalu (Slika 9.5).

Proces u zavisnosti od parametara realizuje sljedeće tipove zahtjeva ili upita:

e) Čitanje iz datoteke na fajl-serveru;
f) Upis u datoteku na fajl-serveru;
g) Prenos datoteke na fajl-server;
h) Prenos datoteke sa fajl-servera;
i) Obraćanje k serveru baza podataka sa ciljem izvršenja sljedećih transakcija:
    a. selektovati zapis (Select);
    b. dodati zapis u bazu podataka (Insert);

c. izmijeniti zapis u bazi podataka (Update);
d. izbrisati zapis iz baze (Delete).

Proces ***Client*** može izgraditi sljedeći slijedeći oblik saobraćaja, tj tok poruka sa zahtjevima koje su usmjerene ka serverima:

1. *Neprekidni*: sljedeći zahtjev se formira neposredno poslije dobijanja potvrde o ispunjenju prethodnog;
2. *Pulsirajući*: sljedeći zahtjev se formira poslije isteka nekog vremena po dobijanju potvrde, pri čemu ovaj interval vremena može biti i slučajna veličina;
3. *Paketski*: koji je predstavlja objedinjenje dva prethodna, naime poslije ispunjavanja niza zahtijeva nastaje prekid, po isteku koga se generiše sljedeće serija zahtjeva.

Proces ***Client*** radi po sljedećoj šemi:

Ako se tekuće vrijeme podudara sa početkom vremena rada procesa prelazi se na tačku2;

1. Šalje se zahtjev za uspostavom aktivnog povezivanja transportnom nivou;
2. Čeka se potvrda o uspostavljenom povezivanju,
3. Šalje se identifikacija korisnika serveru;
4. Čeka se potvrda od servera;
5. Šalje se lozinka serveru;
6. Čeka se potvrda od servera;
7. Ako je tekuće vrijeme sistema nije jednako vremenu završetka rada prelazi se na tačku 9, u suprotnom prelazi se na tačku 11;
8. Formira se zahtjev odgovarajućeg tipa i posredstvom poruke šalje serveru na obradu;
9. Čeka se potvrda od servera i poslije prijema potvrde prelazi na tačku 8;
10. Završetak rada.

Proces ***Client*** tokom svog rada ne sakuplja nikakvu statistiku.

Tablica 9.3. *Parametri modela klijenta*

parametar	opis	vrijednost
Type_Of_Query	Tip zahtjeva (upita) prema serveru	1-5
Begin_Of_Work	Vrijeme početka rada	0
End_Of_Work	Vrijeme završetka rada	1000
Traffic	Tip ostvarenog saobraćaja	1-3

Serverski moduli distribuiranog sistema vrše obradu podataka ili informacija i čuvanje velike količine podataka po zahtjevima klijenata.

Procesi ***FileServer*** i ***DBServer*** modeliraju rad programskog obezbjeđenja fajl-servera i servera baze podataka i i odgovara aplikativnom sloju ISO OSI modela. Procesi ***FileServer*** i ***DBServer*** komuniciraju sa procesom transportnog nivoa po dupleksnom kanalu (Slika 9.5).

Procesi ***FileServer*** i ***DBServer*** rade po sljedećoj šemi:

1. Uspostavlja pasivno povezivanje;
2. Prima ulaznu poruku;
3. Ako je to poruka za uspostavljanje veze, novi klijent se upisuje u spisak, inače prima se porcija podataka sa transportnog nivoa;
4. Ako je poruka neki zahtjev za uslugom prelazi se na tačku 5, inače se ide na tačku 2;
5. U zavisnosti od tipa zahtjeva generiše se kašnjenje;

6. Ako je tokom vremena kašnjenja (obrade prethodnog zahtjeva) došla nova poruka ista se analizira na sljedeći način:
   - Ako je to poruka za uspostavljanje veze, upisuje se novi klijent u spisak i prelazi na tačku 2;
   - Ako je to poruka sa podacima, koji se nalaze u baferu transportnog nivoa, ona se smiješta u rad za čekanje;
7. Ako red nije prazan prima se sljedeća poruka i prelazi na tačku 4.

Procesi ***FileServer*** i ***DBServer*** tokom svog rada ne sakuplja nikakvu statistiku.

9.5 Formiranje modela radnog opterećenje

Kao radno opterećenje distribuiranog sistema, tokom procesa analize, koristio se namjenski razrađen distribuirani aplikativni program. Isti program se koristio i pri realnom eksperimentisanju na kontrolnom sistemu a takođe i pri simulaciji tj. imitacionom eksperimentu sa modelom. Uzrok ovome bila je provjera ispravnosti modela, kako bi se mogli donijeti ispravni zaključci o radu posmatranog sistema. Prilikom izgradnje namjenskog distribuiranog programa vodilo se računa da on bude što je moguće više sistemski nezavistan, jer se samo u tom slučaju pomoću njega može dobiti dobra procjena pokazatelja performansi posmatranog distribuiranog sistema. Namjenski program je bio urađen u klijent/server tehnologiji, pri čemu je:

- serverski dio podržavao procese transakcione obrade u bazi podataka, uključujući i dodatna izračunavanja, kao i razmještaj podataka u okviru distribuirane baze.
- klijentski dio obezbjeđivao udaljeni pristup sistemu i grafičko predstavljanje korisničkog interfejsa u sklopu operativnog sistema Windows.

Kako je već naglašeno u poglavlju 7.4.4, parametri i promjenljive radnog opterećenja koje se prezentira imitacionom modelu, mogu biti generisani stohastički uz pomoć odgovarajućih raspodjela slučajnih brojeva ili mogu biti određene deterministički na osnovu iskustvenih informacija i podataka sakupljenih tokom izvršavanja pomenutog namjenskog programa na kontrolnom sistemu. Tokom procesa analize posmatranog hipotetičkog distribuiranog sistema, a za dobijanje različitih informacija o njegovom funkcionisanju, korišćena su oba ova načina formiranja radnog opterećenja.

9.5.1 Skupljanje trase

Skupovi relevantnih podataka koji karakterišu ponašanje aplikativnih programa, bolje rečeno aplikativnih procesa, u distribuiranoj sredini označeni su kao ***trasa***. Trasa procesa je hronološki uređen niz relevantnih podataka o događajima aktivnostima tog procesa. U trasi se nalaze informacije, koje dozvoljavaju da se izračuna složenost (komleksnost) unutrašnjih aktivnosti procesa tj. aktivnosti koje ne zahtijevaju interakciju sa računarskom sredinom i drugim procesima. Vrijeme početka događaja u hardverskoj sredini se koristi za određivanje integralnih karakteristika dinamike programa.

Trasa je čista ako sistem posmatranja, mjerenja i skupljanja ne kvari trase procesa svojim aktivnostima. Tokom ovog istraživanja maksimalno se težilo čistoći trase. Kad god je to bilo moguće, javno je ukazano na mjesto izobličenja i isti su uzeti u obzir kod procjene rezultata. Za optimalne rezultate je važno imati operativni sistem poznat u svim

svojim aspektima. Međutim, ovo nije lako ostvariti, s obzirom da proizvođači operativnih sistema prikrivaju mnoge detalje njihove arhitekture. Izuzetak čine neke verzije Linux-a, koje su u potpunosti otvorene za istraživača. S druge strane, objektno orijentisan način izgradnje namjenskog distribuiranog programa i bogata sredstva debagiranja sistema programskih kompajlera jezika C^{++}, dozvolile su visok stepen otkrivanja, kontrole i upravljanja izobličenjima. Kontrola nedeterminizma se obavljala u tačkama sinhronizacije i tačkama interakcije procesa. U ovim tačkama vršilo se skupljanje i razmjena informacija prvog nivoa. Instaliranjem specijalizovanih sredstava za skupljanje informacija na drugom, višem nivou, vršena je dinamička interakcije sa njima i sakupljanje njihovih podataka u trasu.

Prilikom sakupljanja trase, kao instrument posmatranja ili mjerenja korišćen je program, zato što ni hardverski ni mikroprogramski posmatrači ne obezbjeđuju skupljanje potrebnih informacija (poglavlje 8.5). Hardverski posmatrač evidentira samo ono što prolazi pored njega, dok se softverski posmatrač može implementira tačno tamo gdje je potrebno. Drugim riječima hardverski posmatrač je pasivan za informacija koje sakuplja, a program je aktivan.

Ovdje realizovani softverski posmatrači predstavljaju konkretizaciju koncepcije posmatrača formulisanog u poglavlju 8.3.1. U svakom slučaju za snimanje trase parametara radnog opterećenja distribuiranog sistema potrebno je bilo pripremiti skup programa koji su se izvršavali istovremeno sa namjenskim distribuiranim programom. Bolje rješenje od ovog je kad se softverski posmatrači integrirašu u same operativne sisteme. Ovo se radi u cilju minimiziranja dodatnih gubitaka rada operativnih sistema i programa, što dalje dovodi do minimizacije konflikata između procesa posmatrača i procesa distribuiranih programa. Praktična realizacija ove ideje bi bila moguća ako bi abonenti kontrolnog sistema radili pod operativnim sistemom Linux, čiji je izvorni kod dostupan i realizovan na programskom jeziku C. U tu svrhu bi se u okviru operativnih sistema abonenata instalirale rutine, koje bi obezbjedile izgradnju trase u "on-line" režimu. Međutim, ovo će biti ostavljeno za neko buduće istraživanje

Sam postupak formiranja modela radnog opterećenja sprovođen je na sljedeći način: namjenski program iz zadatog skupa distribuiranih aplikativnih programa se izvršavao u specijalizovanoj distribuiranoj sredini, koja je označena kao kontrolni distribuirani računarski sistem. Tom prilikom su se mjerile određene karakteristike ponašanja programa na nivou njegove sistemske nezavisnosti. Rezultati mjerenja su se zatim skupljali i obrađivali u specijalizovanom podsistemu koji se nazvan "TRASA". Taj podsistem obezbjeđuje sakupljanje i obradu trase na kontrolnom distribuiranom sistemu i izgradnju na njenoj osnovi modela radnog opterećenja posmatranog distribuiranog sistema. Podsistem "TRASA" izgrađen u okviru sistema imitacionog modeliranja, formira i ažurira bazu podataka o dinamici ponašanja aplikativnog programa. Ona dozvoljava da se:

- dobiju integrirani podaci o dinamici programa,
- vizualizuje i konkretizuje ponašanje distribuiranog programa, npr. sa ciljem nalaženja anomalija, koje se ne potčinjavaju formalizaciji,
- utvrdi odnos između funkcionalnih zahtjeva i stvarnog ponašanja programa,
- obezbijedi imitacionom modelu kvalitativno radno opterećenje.

9.5.2 Stohastički model radnog opterećenja

Stohastički model radnog opterećenja, predstavlja numeričku (matematičku) metodu za opisivanje tokova zahtijeva, koje korisnička populacija prezentira distribuiranom računarskom sistemu na obradu. Izrađena je na osnovu empirijskih podataka iz literature i poznatih relacija iz teorije vjerovatnoće i teorije masovnog opsluživanja. U osnovi ove metode leži procedura, primjenljiva za modeliranje slučajnih veličina i funkcija, koja se naziva metoda statističkih ispitivanja ili *metoda Monte-Carlo*. Ova metoda je ovdje imala, više teorijsku nego praktičnu primjenu. Osnovni razlog leži u činjenici da njena tačnost po pravilu ne prevazilazi 30 procenata. Ona je poslužila za dobijanje nekih preliminarnih rezultata, korišćenih u početnoj fazi istraživanja.

Sa stanovišta teorije masovnog opsluživanja može se reći da je cilj funkcionisanja distribuiranog sistema zadovoljenje klijentskih zahtjeva za uslugom od strane servera. Proces pristizanja zahtjeva u sistem na opsluživanje je slučajni (stohastički) proces, pa on može biti opisan funkcijom $X(t)$ koje određuju broj zahtijeva koji se nalaze na opsluživanju u sistemu, u intervalu vremena $(0, t)$. Značenja funkcije $X(t)$ su slučajne veličine za ma koju vrijednost t. Realno, čak iako se izaberu intervali vremena jednake dužine, malo je vjerovatno da će u tim intervalima vremena doći jednak broj zahtjeva. U sistemu može biti više servera i više ulaznih tokova zahtijeva koji dolaze na opsluživanje. Slučajne veličine, koje opisuje funkcija $X(t)$, mogu poprimiti samo cjelobrojne, konačne vrijednosti $0, 1, 2, ..., k$, gdje je k cijeli broj, pa su tokovi zahtjeva ili ulaznih poruka stohastički procesi sa *diskretnim stanjima*. Funkcija $X(t)$ biće u potpunosti određena ako za ma koji pozitivan interval vremena $t_1, t_2, ..., t_n$, može biti tačno određena njena vrijednost, tj. broj zahtjeva ili poruka koji u tom intervalu pristigne u sistem. Sa druge strane, slučajna veličina x_i se karakteriše odgovarajućim zakonom raspodjele, koji opisuje funkcija raspodjele $F_{x_i}(c)$. Ova funkcija se definiše kao vjerovatnoća toga da je $x_i < c$, gdje je c proizvoljan broj, tj.:

$$F_{x_i}(c) = p\{x_i < c\} \tag{9.1}$$

Metoda Monte-Carlo sastoji se u tome da se zadaje slučajna veličina x, čije je matematičko očekivanje jednako traženoj veličini X, tj.:

$$M[x] = X \tag{9.2}$$

Približna procjena nezavisnog matematičkog očekivanja, koja se poklapa sa traženim rezultatima, nalazi se kao aritmetička sredina rezultata nezavisnih opita. Ostvaruje se n nezavisnih ispitivanja slučajne veličine x: $\{x_1, x_2, ..., x_n\}$ i shodno zakonu velikih brojeva približno se pretpostavlja:

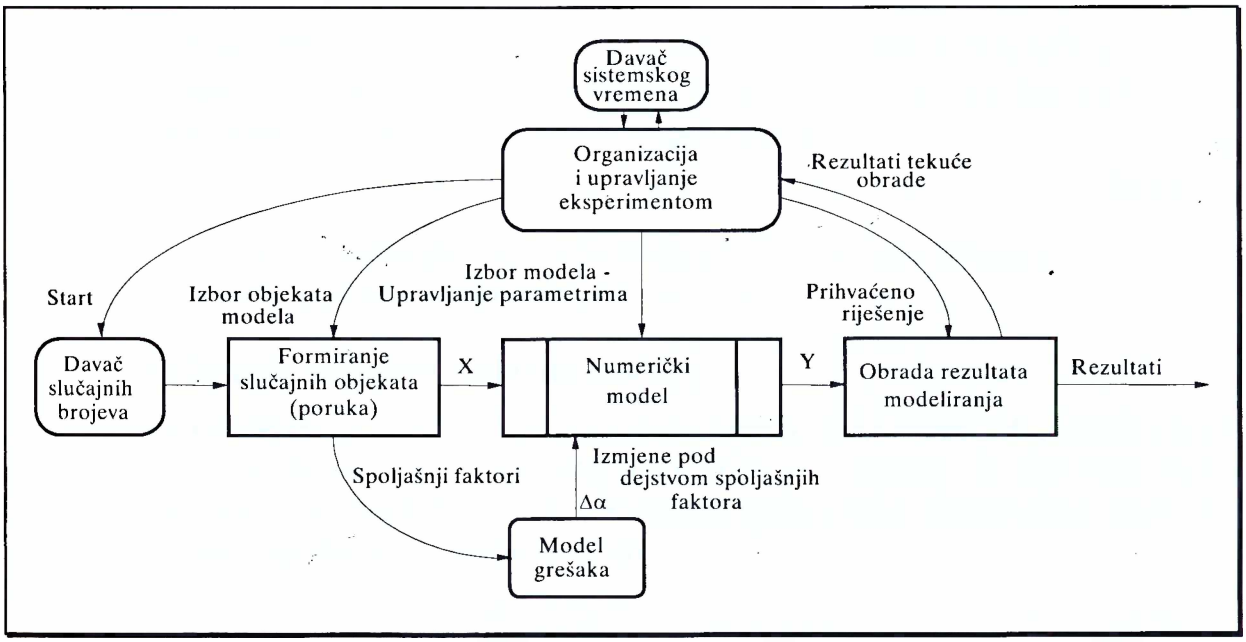
$$\bar{x}_n = \frac{x_1 + x_2 + ... + x_n}{n} \approx X \tag{9.3}$$

Na osnovu (9.2) za ma koje n važi relacija:

$$M[x_n] = M\left[\frac{1}{n}\sum_{i=1}^{n} x_i\right] = \frac{1}{n}\sum_{i=1}^{n} M[x_i] = \frac{nX}{n} = X \tag{9.4}$$

Ispitivanje se sprovodi sve dotle dok se disperzija procjene $\bar{x}_n$ ne snizi do potrebne veličine koja zavisi od dozvoljene greške. Strukturna šema imitacionog eksperimenta upravljanog stohastičkim modelom, zasnovanim na metodi Monte-Carlo prikazana je na slici 9.4.

Zahtjevi za obradom ili rješenjem nekog problema, pristižu u sistem u obliku nizova poruka. Ove poruke sa zahtjevima nose sve relevantne podatke kojima se korisnik ili korisnički program obraća logičkim i fizičkim resursima sistema. To su između ostalog vrijeme nastanka zahtijeva, vjerovatno vrijeme potrebno za zadovoljenje zahtjeva, identifikacija procesa ili resursa kojem je poruka usmjerena i tome slično. Za formiranje ovih slučajnih objekata (*poruka*) u stohastičkom modelu radnog opterećenja koriste se procedure, tzv. davači slučajnih brojeva, koji generišu skup vrijednosti slučajnih veličina.



Slika 9.4 Strukturna šema imitacionog eksperimenta upravljanog stohastičkim modelom radnog opterećenja.

Funkciju uticaja spoljašnjih faktora (npr. smetnji u komunikacionom kanalu), na tok zahtjeva ili poruka pristiglih u sistem, karakterišu faktori izmjene $\Delta\bar{\alpha}$ parametara modela. U ove faktore izmjene mogu se ubrojati pojave grešaka u porukama, upravljačke informacije računarske mreže i tome slično.

9.5.3 Modeliranje slučajnih brojeva

Generisanje na računaru nizova izabranih vrijednosti slučajnih veličina sa zadatim zakonom raspodjele naziva se modeliranjem tih veličina. Slučajni brojevi, koji se povinuju nekom određenom zakonu raspodjele, modeliraju se uz pomoć pretvaranja niza nezavisnih slučajnih brojeva y, ravnomjerno raspoređenih u intervalu [0,1]. Umjesto niza slučajnih, sa dovoljnom tačnošću može se koristiti niz *pseudoslučajnih* brojeva ravnomjerno raspoređenih na intervalu [0,1], dobijen uz pomoć procedure u kojoj se primjenjuje kongruentna metoda. U osnovi te metode nalazi se sljedeća rekurzivna jednačina:

$$y_i = (my_{i-1} + a) mod(2^n), \qquad (9.5)$$

gdje je y_{i-1} poslednji već izgenerisani pseudoslučajni broj, dok su: a, m, n konstante, y_o je početni broj u nizu, koji se naziva i početnom vrijednošću niza. Početna vrijednost određuje sve brojeve u nizu a zadaje je istraživač ili korisnik modela.

Niz brojeva $[y_i]$ je periodičan sa periodom 2^n. Taj period ciklusa će biti dobijen u tom slučaju, ako je a prost broj u odnosu na 2^m, gdje su $m=1+4k$, k - cijeli brojevi. Značenje konstante m zavisi od maksimalno cijelog broja, koji može biti predstavljen na računaru na kome se sprovodi modeliranje.

Procedura dobijanja niza slučajnih brojeva koje imaju funkciju raspodjele $F_X(x)$ se sastoji iz dva koraka:

1. Prvi korak je dobijanje slučajnih brojeva y: $\{y_1, y_2 ... y_n\}$ na intervalu [0,1].
2. U drugom koraku procedure treba riješiti jednačinu $y_i = F_X(x_i)$. Ako je funkcija $F_X(x)$ data u empirijskoj formi kao tablica parova /x, $F_X(x)$/, vrijednosti x_i mogu biti dobijene interpolacijom. Ako je funkcija $F_X(x)$ zadata u analitičkoj formi, a postoji i analitička forma inverzne funkcije, traženi niz pseudo-slučajnih brojeva se može dobiti i bez interpolacije, uz pomoć sljedeće relacije:

$$x_i = F_X^{-1}(y_i). \tag{9.6}$$

Na taj način, za svako značenje y_i jednačina (9.6) daje odgovarajuću vrijednost x_i. Dati metod je poznat pod nazivom *metod obrnute funkcije*.

9.5.4 Modeliranje vremena dolaska poruke

Eksperimentalno je utvrđeno da pri interaktivnom radu abonenata distribuiranog sistema ulazni tok poruka za zahtjevima, koji pristižu u sistem, je dovoljno blizak Poisson-ovom toku. To znači da broj N poruka pristiglih u toku vremenskog intervala t ima Poisson-ovu raspodjelu sa parametrom λt.

$$P_N(t) = \frac{(\lambda t)^N}{N!} e^{-\lambda t}; \qquad (N \geq 0; t \geq 0) \tag{9.7}$$

Funkcija $P_N(t)$, ako se ne uzme u obzir t zavisi samo do λ i predstavlja vjerovatnoću dolazaka tačno N zahtjeva u toku vremenskog intervala $(0, t)$. Tok zahtjeva ovakvog tipa u potpunosti karakteriše veličina λ koja se naziva parametrom toka. Parametar toka λ je jednak matematičkom očekivanju broja zahtjeva, koji u jedinici vremena dolaze u sistem, pa se λ označava i kao srednji intenzitet dolazaka.

$$M_t[N] = \lambda t e^{-\lambda t} e^{\lambda t} \tag{9.8}$$

$$M_t[N] = \lambda \tag{9.9}$$

Poisson-ov tok, označen još i kao prosti tok zahtjeva, posjeduje sljedeća svojstva:

- verovatnoća dolaska određene količine poruka u toku određenog intervala vremena ne zavisi od početka računanja vremena, već zavisi samo od dužine tog vremenskog intervala,
- poruke sa zahtjevima dolaze u sistem u proizvoljnim momentima vremena, koji ne zavise od broja ranije pristiglih poruka i vremena njihovih dolazaka,
- u ma kom trenutku vremena u sistem može doći samo jedan zahtjev.

Poruke dolaze u sistem u slučajnim vremenskim trenucima. Ovi vremenski trenutci dolaska zahtjeva obrazuju konačan ili prebrojiv skup, pa se dobija *proces sa diskretnim vremenom*, ili stohastički niz. Vremenski interval između dolazaka dva uzastopna zahtjeva je slučajna veličina koju karakteriše eksponencijalna funkcija raspodjele.

$$y(t) = 1 - e^{-\lambda t}; \qquad (t \geq 0). \tag{9.10}$$

Srednja vrijednost vremenskog intervala između dolazaka dva zahtjeva je jednaka $1/\lambda$. Vrijednost t se izračunava po metodi obrnute funkcije

$$t = -\frac{1}{\lambda} ln(1 - y); \qquad (0 \leq y < 1). \tag{9.11}$$

9.5.5 Vjerovatnoća vremena opsluživanja zahtjeva

Minimalno vrijeme opsluživanja poruke u sistemu je takođe slučajna veličina. Pod minimalnim vremenom podrazumijeva se čisto procesorsko vrijeme ili čisto vrijeme zadržavanja fizičkog resursa. Eksperimenti su pokazali da ovo minimalno vrijeme ***d(t)*** karakteriše normalna funkcija raspodjele.

$$F_{\mu,\sigma^2}(x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{(u-\mu)^2}{2\sigma^2}} du; \qquad -\infty < x < \infty \tag{9.12}$$

Funkcija $F_{\mu,\sigma^2}(x)$ zavisi od dva faktora: matematičkog očekivanja μ i disperzije σ. Za generisanje normalno raspoređenih pseudoslučajnih brojeva koristi se procedura u osnovi koje leži ideja Marsaglio-Bray [123]. Ova metoda se sastoji u tome da se generišu dva slučajna broja r_1 i r_2 ravnomjerno raspoređena na intervalu [0,1]. Zatim se izračunava:

$$S = V_1^2 + V_2^2, \qquad \text{gdje je: } V_1 = -1 + 2 \cdot r_1 \quad \text{i} \quad V_2 = -1 + 2 \cdot r_2. \tag{9.13}$$

Pri $S \geq 1$ ciklus se ponavlja, a pri $S < 1$ izračunava se:

$$x_1 = V_1 \cdot \sqrt{\frac{-2 ln S}{S}} \qquad \text{i} \qquad x_2 = V_2 \cdot \sqrt{\frac{-2 ln S}{S}}. \tag{9.14}$$

Na taj način se dobija standardno normalna raspodjela sa matematičkim očekivanjem jednakim 0 i disperzijom koja je jednaka 1. Gustina takve raspodjele je data relacijom:

$$f(x) = F'(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}. \tag{9.15}$$

Potrebna normalna raspodjelu minimalnog vremena opsluživanja zahtijeva-poruke dobija se uz korišćenje jednačine:

$$z_1 = \mu + x_1 \cdot \sigma \quad \text{i} \quad z_2 = \mu + x_2 \cdot \sigma, \tag{9.16}$$

gdje je μ matematičko očekivanje, a σ disperzija zahtijevane raspodjele.

9.6 Određivanje dužine odvijanja imitacionog i mjernog eksperimenta

Tokom mjernog eksperimenta u kontrolnom sistemu a isto tako i tokom imitacionog eksperimenta na modelu vrši se evidentiranje podataka o pokazateljima performansi posmatranog distribuiranog sistema. Može se reći da u toku odvijanja jednog takvog mjernog eksperimenta, tj. vremenskog intervala posmatranja sistema, kroz sistem prođe n korisničkih zahtjeva za uslugom. Isto toliko puta treba snimiti odgovarajuće vrijednosti relevantnih parametara x sistema, čime se dobija n skupova vrijednosti svake od ovih veličina.

Kod mjerenja performansi kontrolnog distribuiranog sistema, ili njihovog evidentiranja tokom imitacionog eksperimenta sa stohastičkim modelom radnog opterećenja, mogu se primijeniti poznate relacije iz teorije grešaka. Po ovoj teoriji, sa povećanjem broja n (snimanja ili mjerenja) iznad nekog dovoljno velikog broja, izmjerene veličine teži nekoj srednjoj vrijednosti koja se označava kao *matematičko očekivanje* $\bar{x}$ date veličine. Na taj način se proces snimanja pokazatelja performansi distribuiranog sistema svodi na seriju mjernih ili imitacionih eksperimenata. To znači da se mjerni eksperiment sprovodi m puta. Očevidno, ako se uporede rezultati nekoliko serija eksperimenata jednog istog parametra sistema, najtačnija vrijednost će biti dobijena u seriji, u kojoj kriva raspodjele vrijednosti analiziranog parametra bude najuža. Što je uža kriva raspodjele manja će biti greška $e = x - \bar{x}$ trenutno snimljenog parametra. Tačnost rezultata mjeri se matematičkim očekivanjem kvadrata greške σ^2, koja se naziva disperzijom .

$$\sigma^2 = E(e^2) = \int_{-\infty}^{\infty} (x - \bar{x})^2 p(x) \cdot dx \qquad (9.17)$$

Kako se ovdje radi o konačnom broju posmatranja n, dobijene vrijednosti predstavljaju *slučajni izbor* iz generalnog skupa vrijednosti, pa se veličina $\bar{x}$ naziva izabranom srednjom vrijednošću i može se dobiti po formuli:

$$\bar{x}_n = \left(1/n\right) \sum_{i=1}^{n} x_i \,. \qquad (9.18)$$

Po analogiji sa prethodnim izrazom može se uvesti *izabrana disperzija* S_n^2, koja se može opredijeliti kao srednja vrijednost kvadrata otklona $(x_i - \bar{x}_n)$.

$$S_n^2 = \left[1/(n-1)\right] \sum_{i-1}^{n} \left(x_i - \bar{x}_n\right)^2 \qquad (9.19)$$

Na osnovu gornjih relacija zaključuje se da tačnost rezultata mjerenja zavisi od broja mjernih ili imitacionih eksperimenata n, koji su sprovedeni i uzeti u razmatranje. Sa povećanjem broja n veličina izabrane disperzije, po pravilu se smanjuje, a samim tim se i tačnost rezultata povećava. Ovdje treba naglasiti da su ovi tačni rezultati, tačni samo u to slučaju kada su identični procesi koji se odvijaju u modelu i u realnom sistemu. Ako između tih procesa postoji neko razmimoilaženje, rezultati modeliranja neće biti tačni bez obzira na broj ponavljanja imitacionih eksperimenata. Isti je slučaj ako je softverski

posmatrač, koji mjeri performanse na realnom distribuiranom sistemu, neadekvatan ili daje netačne rezultate

Mjerni ili imitacioni eksperiment za dobijanje nekog pokazatelja performansi sistema x je potrebno sprovoditi sve dotle dok raspodjela veličine x i vrijednost njegove disperzije ne budu ocijenjeni kao dovoljno tačni. U tom cilju, paralelno sa procesom snimanja vrijednosti pokazatelja performansi distribuiranog sistema nužno je i izračunavanje izabrane disperzije tih veličina. Izračunavanjem izabrane disperzije vrši se procjena tačnosti mjerenja dobijenih pokazatelja performansi. Tek kad istraživač zaključi da su model, odnosno posmatrač, adekvatni i da su svi posmatrani pokazatelji performansi dostignu željenu tačnost mjerni, odnosno imitacioni eksperiment se može zaustaviti a rezultati prezentirati.

9.7 Rezultati analize posmatranog distribuiranog računarskog sistema

U toku analize performansi posmatranog distribuiranog sistema sprovedeno je nekoliko serija eksperimenta i to kako realnih na kontrolnom sistemu, tako i imitacionih na izgrađenom modelu. Posmatrano je ponašanje ovog distribuiranog sistema u uslovima različitog opterećenja i topologije.

Rezultati, ovdje izloženi dobijeni su posmatranjem distribuiranog računarskog sistema, koji se sastojao od pet standardnih ethernet segmenata. Svaki segment objedinjavao je do 25 radnih stanica, fajl-server i/ili server baza podataka. Komunikacija je ostvarena uz pomoć protokola IEEE 802.3. Ethernet segmenti i abonenti mreže objedinjeni su u jedinstven sistem uz pomoć dodatnih komunikacionih uređaja (engl. *Data Communication Equipment*, ***DCE***.), kao što su npr.: mostovi, ruteri i habovi (koncentratori). U okviru cijele mreže ostvarivala se potpuna maršrutizacija pri prenosu poruka. Ruteri i mostovi se javljaju pomoćnim uređajima mreže i služe za prenos i maršrutizaciju poruka u mreži.

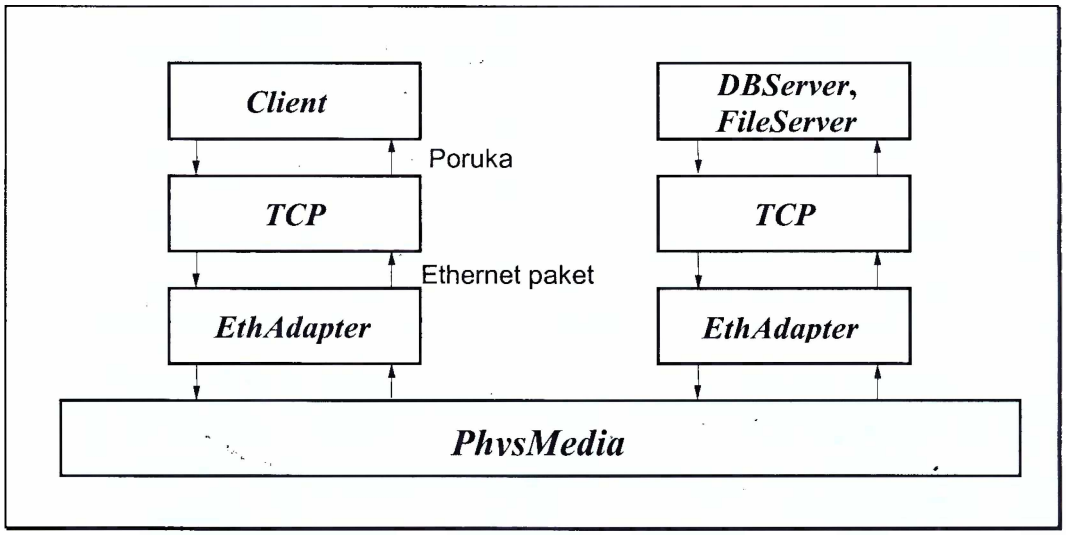
Imitacioni model je razrađen uz pomoć opisanog sistema modeliranja, korišćenjem modela sa porukama. On predstavlja distribuirani program koji imitira rad osnovnih komponenti mrežnog interfejsa i protokola razmjene podataka. Sve komponente modela funkcionišu sinhrono. Uzajamna interakcija se ostvaruje razmjenom poruka, čime se kvalitetno i na prirodan način imitiraju tokovi podataka i službene komunikacije. Bibliotečke komponente ovog modela opisani su poglavlju 9.4.

Strukturno model predstavlja skup procesa koji imaju slojevitu strukturu. Svaki sloj odgovara nekom nivou u modelu mrežne arhitekture OSI ISO. U modelu su realizovani sljedeći slojevi OSI ISO arhitekture (u zagradama su dati nazivi programskih procesa modela, koji realizuju dati sloj):

- aplikacioni ***(Client, DBServer, FileServer)***,
- transportni ***(TCP)***,
- kanalni ***(EthAdapter)***,
- fizički ***(PhysMedia)***.

Ostali slojevi nijesu predstavljeni u modelu, a njihove funkcije su raspoređene između odgovarajućih procesa koji predstavljaju viši ili niži sloj. Izbor slojeva mrežne arhitekture vršen je po principu najvećeg uticaja na kašnjenje poruke kroz mrežu. Strukturna šema jednog ethernet segmenta prikazana je na slici 9.5.

Proces rada modela sastoji se u prenosu poruka između procesa koji modeliraju iste slojeve mrežne arhitekture. Prenos poruka se vršio u saglasnosti sa protokolima TCP/IP i protokolima IEEE 802.3 mreže Ethernet.



Slika 9.5 Strukturna šema jednog ethernet segmenta.

Provjera modela vršila se upoređivanjem rezultata modeliranja sa rezultatima dobijenim mjerenjem na kontrolnom sistemu slične konfiguracije. Osim toga, za dobijanje ulaznih parametra modela, takvih kao što su performanse servera i kašnjenja u mreži, potrebno je bilo izvršiti, određena mjerenja i posmatranja na realnom kontrolnom sistemu iste ili slične konfiguracije. Najbolji rezultati pri analizi se dobijaju kombinacijom metoda modeliranja i mjerenja.

Kao mjera performansi sistema u ovom radu je bilo izabrano vrijeme izvršavanja pojedinih operacija na serverima. Mjerenja se vršena uz pomoć softverskog monitora, koji su bili instalirani na radnim stanicama - klijentima. Kao tajmer mjernog sistema korišćen je sistemski tajmer personalnog računara. Razlika između rezultata dobijenih mjerenjem i modeliranjem nije prelazila petnaest procenata.

Rezultati analize performansi fajl-servera i servera baza podataka dati su u sljedećim tabelama.

Tabela 9.4. *Vrijeme obrade jedne operacije na fajl-serveru.*

	sr. vrjeme obrade zahtj. serv [ms]	sr.vr. obrade zahtj. u mreži [ms]	broj mjerenja	sr. broj prim/pred. pak.	srednja dužina prim./pred. paketa	greška uslov. dispe-rzijom [%]	greška tajmera [%]
Create	10,21	9,97	50	1/1	71,7/91,6	3,68	2,16
Open	8,81	7,56	24	1/1	72,8/90,1	3,72	3,07
Close	6,56	7,43	50	1/1	59,3/54,0	9,29	5,42
Delete	2,32	3,63	50	1/1	71,1/54,0	8,94	8,94

Tabela 9.5. *Brzina čitanja i upisa u datoteku na fajl-serveru.*

	sr. brzina obrade zahtj. serverm [Kbyte/s]	sr.br.. obrade zahtj.u mreži [Kbyte/s]	broj mjerenja	srednja dužina prim./pred. paketa	greška uslov.a disperzijom [%]	greška tajmera [%]
Write	480,55	210,31	45	1521/85	0,52	0,95
Read	560,05	374,65	45	64/1589	0,74	1,35

Tabela 9.6. *Vrijeme obrade operacije nad jednim slogom baze podataka.*

	sr. vrjeme obrade zahtj. serverm [ms]	sr.vr. vrijeme obrade zahtj.u mreži [ms]	broj mj	sr. broj prim/p red. pak.	srednja dužina prim./ pred. paketa	greška uslov. dispe-rzijom [%]	greška tajmera [%]
Insert	1,451	3,422	50	2/2	150/90	2,88	0,10
Select	2,044	2,250	45	2/2	85/155	0,53	0,29
Update	3,154	4,112	45	4/4	112/110	1,62	0,04
Delete	4,703	6,204	40	4/4	83/122	1,07	0,08

Provjera i testiranje rada modela vršeno je imitiranjem niza realnih situacija i sastojala se od sljedećih testova:

1. Provjera pravilnosti funkcionisanja sistema u režimu praznog hoda.
2. Provjera pravilnosti rada sistema dvosegmentne konfiguracije sa simetričnim radnim opterećenjem.
3. Provjera pravilnosti rada sistema pri stohastičkom radnom opterećenju.
4. Provjera funkcionisanja sistema pri otkazu nekih radnih stanica.
5. Provjera funkcionisanja sistema pri otkazu servera.
6. Provjera funkcionisanja sistema pri dvostrukom sniženju performansi jednog servera.

Dobijeni rezultati pokazuju da se dati model može uspješno primijeniti za posmatranje jednog ovakvog distribuiranog sistema.

GLAVA 10

10. ZAKLJUČAK

U ovom radu je izvršena jedna sveobuhvatna numerička analiza uticaja hardverske i sistemsko-softverske strukture na performanse distribuiranog računarskog sistema. Ispitano je ponašanje distribuiranih aplikativnih programa i uticaj koji oni imaju na produktivnost sistema.

Dosadašnje teorije analize distribuiranih sistema, izgrađene su prije svega sa ciljem izgradnje matematičkog aparata za opis ponašanja programa i istraživanja njemu ekvivalentnih transformacija. U njima ili nije bio razdvojen distribuirani aplikativni program i fizička sredina, gdje se on izvršava, ili su fizička sredina i vrijeme kao metrička veličina jednostavno nedostajali. U svim dosadašnjim radovima istraživana je samo jedna osobina paralelizma - redanje. Cilj ovog rada je bio, da se jedinstveno analiziraju količinska i algoritamska svojstva posmatrane klase distribuiranih računarskih sistema. Metode istraživanja distribuiranih računarskih sistema, koje se mogu naći u literaturi, nijesu razmatrale problem analize distribuiranog sistema sa ovog stanovišta. Ovdje predloženi imitacioni model obuhvata ne samo ponašanje distribuiranog programa već i karakteristike hardvera i programskog okruženja, koje obezbjeđuje izvršenje programa.

U radu je:

1. Razrađen matematički model, koji opisuje funkcionisanje distribuiranog računarskog sistema. Taj model se principijelno razlikuje od do danas poznatih modela. Za razliku od njih on obuhvata ne samo algoritamski aspekt ponašanja aplikativnog programa, već i karakteristike hardverske strukture distribuiranog računarskog sistema i jasno uključuje vrijeme kao količinski parametar. U okviru ovog modela istraživan je:
   - načini opisa ponašanja programa na osnovu operacionog prilaza,
   - uzajamna veza vremena i ponašanja distribuiranog aplikativnog programa,
   - različiti aspekata paralelizma unutar distribuiranog računarskog sistema,
   - uticaj nedeterminizma na ponašanje distribuiranih programa,
   - zadatka analize performansi jedne klase distribuiranih računarskih sistema i razrađen metod njegovog rješenja.

2. Formulisana je i istraživana koncepcija kompleksnog prilaza imitacionom modeliranju distribuiranog računarskog sistema, koja dozvoljava da u okviru jedinstvenog sistema istraživanja kako algoritamske, tako i količinske karakteristike posmatranog sistema.
3. Na osnovu izloženih rezultata razrađena je nova metoda analize i procjene performansi distribuiranog računarskog sistema, zasnovana na hipotezi o nezavisnosti ponašanja distribuiranog aplikativnog programa od hardverske strukture sistema.
4. Razrađena je metodologija sredstava mjerenja, opisa i analize ponašanja distribuiranih aplikativnih programa i distribuiranog sistema u cijelini.

Rezultati ovog istraživačkog rada mogu poslužiti kao polazna osnova za razvitak novog perspektivnog naučnog metoda u istraživanju osobina ponašanja distribuiranih sistema. Razmotrene su mogućnosti identifikacije nezavisnosti ponašanja aplikativnih programa, od hardverske strukture i iskorišćenje tih svojstava za analizu sistemskih performansi distribuiranih sistema. Taj problem je, bez obzira na veliki praktični značaj, dosada nedovoljno istraživan.

Važna karakteristika imitacionog modela, koji je ovdje predložen, je mogućnost procjene efikasnosti distribuiranih računarskih sistema u fazi njihovog projektovanja, čime se smanjuju troškovi i vrijeme njihove izgradnje. To povećava kvalitet projekta, dozvoljava provjeru odnosa najavljenih ili zahtijevanih ciljeva projekta i njegovih mogućnosti.

Praktični značaj rada ogleda se u izgradnji jedne matematički korektne metode koja dozvoljava:

- procjenu sistemskih performansi distribuiranog računarskog sistema, bez izgradnje prototipa ili emulatora komandi sistema koji se analizira,
- da se na osnovu ponašanja distribuiranog programa u nekoj kontrolnoj sredini, prognozira njegovo ponašanje u novoj hardverskoj sredini i to na etapi projektovanja,
- da se dovoljno tačne procjene performanse distribuiranog računarskog sistema posmatrane klase.

Dobijeni teorijski rezultati provjereni su na distribuiranom računarskom sistemu zasnovanom na personalnim računarima i lokalnoj računarskoj mreži, koji je dijelom instaliran i testiran na Prirodno-matematičkom fakultetu u Podgorici. Za potrebe istraživanja i provjere predložene metode, u danas veoma aktuelnoj tehnologiji klijent/server, bio je izrađen specijalni distribuirani aplikativni program.

Ova metodologija može biti primijenjena i kod razrade krupnih projekata, kada je na osnovu izgradnje nemoguće skupiti na jednom mjestu sve komponente distribuiranog sistema koji se izgrađuje. Ona se može koristiti i za provjeru saglasnosti dinamičkih karakteristika aplikativnih programa potrebama zahtijevanim u sistemima realnog vremena. Dakle, ***u ovom radu je, za analizu i projektovanje posmatrane klase distribuiranih računarskih sistema, predložena jedna jeftina, dovoljno tačna, efikasna i operativna numerička metoda***.

11. LITERATURA:

[1] Aleksić Ž. Tihomir: *Računari organizacija i arhitektura.* Beograd: Naučna knjiga, 1985.

[2] Майерс Г.: *Архитектура современных ЭВМ.* В 2-х книгах, Москва: Мир, 1985.

[3] Petterson A. David, Hennessy L. John: *Computer Organization & Design.* San Francisco: Morgan Kaufmann Publishers, Inc., 1998

[4] Tanenbaum S. Andrew: *Computer Networks.* New Jersey: Prentice-Hall, Inc., Simon & Schuster Company, 1996.

[5] Bertsekas D., Gallager R.: *Data Networks.* New Jersey: Prentice-Hall, Inc., 1987.

[6] Brumnić A.: *Uvod u računarske komunikacije i mreže.* Beograd: Naučna knjiga, 1985.

[7] Dejvis D.V., Barber D.L.A., Prajs V.L., Solomonides C.M.: *Računarske mreže i protokoli.* Beograd: Naučna knjiga, 1986.

[8] Microsoft Press.: *Osnove umrežavanja.* Beograd: CET, 1998.

[9] Ларионов А.М., Майоров С.А., Новиков Г.И: *Вычислительные комплексы, системы и сети.* Ленинград: Энергоатомиздат, 1987.

[10] Goldman E. James, Rawles T. Phillip: *Applied data communications.* New York, Chichester, Weinheim, Brisbane, Singapore, Toronto: John Wiley & Sons, Inc., 2001.

[11] Coulouris George, Dollimore Jean, Kindberg Tim: *Distributed Systems.* Harlow, England: Addison-Wesley Publishing Company Inc., 1994.

[12] Athey H. Thomas, Zmud W Robert.: *Computers and Information Systems.* llinois, Boston, London: Scott, Foresman and Company, 1988.

[13] Mano M. Morris: *Computer system architecture.* New Jersey: Prentice-Hall, Inc., Englewood Cliffs, 1982.

[14] Glenford J. Myers: *Advances in computer architecture.* New York, Chichester, Brisbane, Toronto, Singapore: A Wiley-Interscience Publication John Wiley & Sons, 1982.

[15] Stallings William: *Operating Systems.* New Jersey: Prentice-Hall, Inc., 1998.

[16] Tanenbaum S. Andrew: *Distributed Operating Systems.* New Jersey: Prentice-Hall, Inc., 1995.

[17] Корнеев В.В.: *Параллельные вычислительные системы.* Москва: Нолидж, 1999.

[18] Orfali Robert, Harkey Dan and Edwards Jeri: *Client/Server Survival Guide.* New York, Chichester, Weinheim, Brisbane, Singapore, Toronto: John Wiley & Sons, Inc., 1999.

[19] Goldman E. James, Rawles T. Phillip, Mariga R. Julie: *Client/Server Information systems.* New York, Chichester, Weinheim, Brisbane, Singapore, Toronto: John Wiley & Sons, Inc., 1999.

[20] Wakerly F. John: *Microcomputer architecture and programming.* New York, Chichester, Brisbane, Toronto: John Wiley & Sons, 1981.

[21] Stojčev K. Mile: *Savremeni 16-bitni mikroprocesori, (deo - I, II i III).* Niš: IRO "Naučna knjiga", Beograd, Ei Niš, RO "Istraživačko-razvojni institut, 1988.

[22] Советов Б. Я., Кутузов О. И., Головин Ю. А., Аветов Ю. В.: *Применение микропроцессорных средств в системах передачи информации.* Москва: Высшая Шк., 1987.

[23] Ribarić S.: *Arhitektura računala pete generacije.* Zagreb: Tehnička knjiga, 1986.

[24] Shannon R.E.: *System simulation.* New Jersey: Prentice-Hall, 1975.

[25] Ладенко И.С.: *Имитационные системы.* Москва: Наука, 1981.

[26] Киндлер Е.: *Языки моделирования.* Москва: Энергоатомиздат, 1985.

[27] Monsef Y. *Modelling and simulation of Complex Systems.* Erlangen: Ghent: Budapest: San Diego: Society for Computer Simulation Int., 1997.

[28] Kleinrock L.: *Queuing systems.* Volume II, computer applications. New York: London: Sydney: Toronto.

[29] Ferrari D.: *Computer systems performance evaluation.* New Jersey: Prentice-Hall, 1978.

[30] Ferrari D., Serazzi G., Zeigner A.: *Measurement and tuning of computer systems.* New Jersey: Prentice-Hall, 1983.

[31] Артамов Г.Т., Брехов О.М.: *Аналитические вероятностные модели функционирования ЭВМ.,* Москва: Энергия, 1979.

[32] Авен О.И., Гурин Н.Н., Коган Я.А.: *Оценка качества и оптимизация вычислительных систем.* Москва: Наука, 1984.

[33] Kobayashi H.: *System performance evaluation methodology* Addison-Wesly Pub., 1979.

[34] Bell C.G., Newell A.: *Computer structures: Reading and Examples.* McGraw-Hill Book Comp., 1979.

[35] Lunde A.: *Evaluation of instruction set processor architecture by program tracing.* Cornegie-Mellon Univ. Press., 1974.

[36] Salisbure A.B.: *The evaluation of micro-program implemented emulator TR-60.* Stanford, California, 1978.

[37] Sreenivasen K., Kleiman A.J.: *On the construction on represent able synthetic workload.* CACM, Vol. 17., N° 3., 1974., pp. 127-133.

[38] Kerner H., Kuemmerla K.: *Performance measures, definitions and metric.* Proc. 6-th annual Princiption Conference on Information Science and System, 1973., pp. 213-217.

[39] Kernighan B.W., Hamilton P.A.: *Synthetically generated performance test load for operating systems.* Proc. 1-st annual SIGME symposium on measurement find evaluation, 1973., pp. 121-126.

[40] Babaoglu O.: *On construction synthetic programs for virtual memory environments.* Second summer school on Computer performance and evaluation, North-Holland, 1981., pp. 195-205.

[41] Cabrera L.F.: *Benchmarking UNIX – a comparative study.* Second summer school on Computer performance and evaluation, North-Holland, 1981., pp. 205-217.

[42] Conti D.M.: *US efforts to develop standard benchmark programs.* Performance of computer installations. North-Holland, 1984., pp. 55-67.

[43] Wyrick T.F.: *Benchmarking distributed system: objectives and techniques.* Performance of computer installations, North-Holland, 1984., pp. 67-82.

[44] Ferrary D.: *A performance-oriented procedure for modeling interacting workloads.* Experimental computer performance evaluation, North-Holland, 1981., pp. 57-79.

[45] Friedman H.P.: *Statistical methods in computer performance evaluation.* Experimental computer performance evaluation, North-Holland, 1981., pp. 113-142.

[46] Cheng P.S.: *Trace driven system modeling.* IBM System Journal Vol. 81969., N° 4. pp. 280-289.

[47] Mattson R.L., Gecsei J., Shetz D.R., Traiger I.L.: Evaluation techniques for storage hierarchies. IBM System Journal Vol. 91970., N° 2.pp. 78-107.

[48] Noetzel A.S., Herring L.A.: *Experience with trace driven modeling.* Proc. Symposium on the Simulation of Computer System, 1976., pp. 111-118

[49] Sherman S.W.: *Trace driven modeling: an update.* Proc. Symposium on the Simulation of Computer System, 1976., pp. 87-91.

[50] Balsi O., Sarget R.: *Evaluation of multi-variance response trace driven simulation models*. Proc. Performance evaluation of computer systems. 1983. pp. 309-323.

[51] Смелянский Р.Л., Бахмуров А.Г., Гурьев Д.: *Об одной вероятностной модели программ.*, В журнале: Программирование N° 6., 1986.

[52] Smeliansky R.L., Alonko T.: *On calculation of transition probabilities in the program*. Information Processing Letters N° 3.. 1986.

[53] Gehringer E.F., Schwetman H.G.: *Run-time characteristics of a simulation model.* Proc. Symposium on the Simulation of Computer System, 1984., pp. 121-129.

[54] Бочков С.О., Смелянский Р.Л.: Отладка программ в распределенных вычислительных средах. В журнале: Программирование N° 4., 1988.

[55] Anderson J.W., Brown J.C.: *Graph model of computer systems: application the performance evaluation*. Proc. Conference on Computer Systems Performance evaluation and Modeling. 1978., pp. 166-176. .

[56] Cvetković D., Milić M.: *Teorija grafova i njena primjena.* Beograd: Naučna knjiga, 1977.

[57] Siewiorek D.: *Introduction to PMS.* Computer Vol. 71974., N° 12. Pp. 42-44.

[58] Knudsen M.T.: PMSL: *An interactive language for the system level description and analysis of computer structures*. Cornegie-Mellon, University Press, 1975.

[59] Головкин Б. А.: *Расчет характеристик и планирование параллельных вычислительных процессов*. Москва: Радио и Связь, 1983.

[60] Coffman E. G.: *Computer and job-shop scheduling theory.* New York, Chichester, Weinheim, Brisbane, Singapore, Toronto: John Wiley & Sons, Inc., 1976.

[61] Coop D. H.: *An analytical approach to the measurement, evaluation and prediction of computer performance*. California University Press., 1971.

[62] Milner R.: *A Calculus of Communicating Systems.* Lecture Notes in Computer Science 92, Springer-Verlag, New York: 1980.

[63] Hoare C.A.R.: *Communication Sequential Process*. New Jersey, London, New Delhi, Rio de Janeiro, Singapore, Sydney, Tokyo, Toronto, Wellington: Prentice-Hall, 1985.

[64] Hoare C.A.R.: *Notes on communicating sequential systems. In Control flow and Data flow: concepts of distributed programming.* Springer-Verlag, New York: 1986.

[65] Lonegran R., Androsciani V.: *A software monitor for performance* measurement. Stanford University Press, 1970.

[66] Смелянский Р.Л., Ханов А.В.: *Распределенная объектно-ориентированная ОС*. Тр. П-го Всесоюзная конференция по распределенным автоматам. СМО, Кишинев, 1987.

[67] Svobodova L.: *Micro programmable performance analyzer.* Proc. International Conference on Parallel processing, 1983.

[68] Драммонд М.: *Методы оценки и измерений дискретных вычислительных систем*. Москва: Мир, 1979.

[69] Вулф А., Мак Леод Дж.: *Новые инструментальные средства, ускоряющие разработку программного обеспечения.* Электроника, Том 59. 1986., с. 57-59.

[70] Degano G., Montanary U.: *Distributed system, partial ordering of events, and events structures. Control flow and Data flow: concepts of distributed programming*. Springer-Verlag, 1986.

[71] Летичевский А.А., Капитонова Ю.В.: *Математическая теория проектирования вычислительных систем*. Москва: Наука, 1988.

[72] Миренков Н.Н.: *Параллельное программирование для многомодульных вычислительных систем*. Москва: Наука, 1989.

[73] Бахмуров А.Г.: *Исследование структур многоуровневых параллельных ВС на примере задач МГД-моделирования*. Диссертация на соискание учёной степени к.ф.-м.н. Москва: МГУ, 1990.

[74] Смелянский Р.Л., Зелинский П.П.: *Построение рабочей нагрузки на основе описания поведения параллельных программ*. В сборнике: Программное оборудование и вопросы принятия решений. Москва: МГУ, 1987.

[75] Зелинский П.П., Смелянский Р.Л.: *Два подхода к формированию рабочей нагрузки для имитационных моделей многопроцессорных вычислительных систем*. В сборнике: Программное оборудование и вопросы принятия решений. Москва: МГУ, 1988.

[76] Молнов В.Г. Смелянский Р.Л.: *Исследование структур многопроцессорных систем с общей шиной*. В сборнике: Вычислительные системы и вопросы принятия решений. Москва: МГУ, 1988.

[77] Молнов В.Г. Смелянский Р.Л.: *Комплексный подход к моделированию распределенных вычислительных систем*. В журнале: Программирование N° 1., 1988.

[78] Смелянский Р.Л.: *Об инварианте поведения программ*. В журнале: Вестник Московского университета Серия 15. Вычислительная математика и кибернетика, N° 4., 1990., с 54-58.

[79] Смелянский Р.Л.: *Взаимосвязь программы и вычислительной среды.* В сборнике: Вычислительные комплексы и моделирование сложных систем. Москва: МГУ, 1990.

[80] Смелянский Р.Л.: *Анализ производительности распределенных микропроцессорных систем на основе инварианта поведения программ.* Диссертация на соискание учёной степени доктора физико-математических наук. Москва: МГУ, 1991.

[81] Лебедев А. Н., Чернявский Е.А: *Вероятностные методы в вычислительной технике.* Москва: Высшая школа, 1986.

[82] Башарин Г.П., Бочаров П.П., Коган А.Я.: *Анализ очередей в вычислительных сетях.* Москва: Наука, 1989.

[83] Lamport L.: *Time, clocks and ordering events in a distributed system.* CACM, V.21., N° 7. 1978.

[84] Lamport L., Melliar-Smith P.M.: *Synchronizing clocks in the presence of faults.* Journal of ACM Vol. 32., N° 1. 1985.

[85] Вознесенская Е. В.: *Математическая модель алгоритмов синхронизации времена для распределенного имитационного моделирования.* Программные системы и инструменты., Тематический сборник № 1. Москва: Изд-во факультета ВМиК МГУ, 2000., с. 56-66.

[86] Bruno J.L., Sethi R.: *Code generation for one-register machines.* Journal ACM Vol. 23., N° 3. 1975.

[87] Cook S.A.: *The complexity if theorem proving procedures.* Proc. Third Annual ACM Symposium On Theory on Computing. 1971., pp. 151-158.

[88] Karp R.M.: *Reducibility amount combinatorial problems.* In Complexity of Computer computation. New York: Plenum Press, 1972., pp. 85-103.

[89] Nakata J.: *On compiling algorithms for arithmetic's expressions.* Com. ACM Vol. 10., N° 8. pp. 492-494.

[90] Redziejowski R.R.: *On arithmetic expressions and trees.* Com. ACM Vol. 12., N° 2. pp. 81-84.

[91] Sethi R.: *Complete register allocation problems.* SIAM Journal Computing Vol. 4., N° 3. pp. 246-248.

[92] Bruno J.L., Lassage T.: *The generation of optimal code for stack machines.* Journal ACM Vol. 22., N° 3. 1975.

[93] Aho A.V., Johnson S.C.: *Optimal code generation for expression trees.* Journal ACM Vol. 23., N° 3. 1976.

[94] Ахо А., Хопкрофт Дж., Ульман Дж.: Построение и анализ вычислительных алгоритмов. Москва: Мир, 1979.

[95] Котов В.Е.: *Сети Петри*
Москва: Наука, 1984.

[96] Питерсон Дж.: *Теория сетей Петри и моделирование систем.*
Москва: Мир, 1985.

[97] Котов В.Е.: *Элементы параллельного программирования.*
Москва: Радио и связь, 1983.

[98] Bacon Jean: *Concurrent systems.*
Harlow, England: Addison Wesley Longman Ltd, 1998.

[99] Schneider Steve: *Concurrent and Real-time Systems.*
West Sussex, England: John Wiley & Sons, Ltd, 2000.

[100] Ершов А.П.: *Теория схем программ: состояние дел.* В журнале: *Проблемы кибернетики.* Том 27, 1973.

[101] Ершов А.П.: *Введение в теоретическое программирование: беседы о методе.* Москва: Наука, 1981.

[102] Nivat M.: *Behaviors of processes and synchronizing system of process. Theoretical Foundation of Programming Methodology.*
North-Halland Press, 1982.

[103] Murakami K., Kakuta T., Onai R., Ito N.: *Research on parallel machine architecture for fifth-generation computer systems.*
Computer journal, June, 1985., pp. 76-92.

[104] Jones A.K., Chansler R.J., Durham I., Feiler P.H., Scelza D.A., Schwan K., Vegdal S.R.: *Programming issues raised by multiprocessors.* Proc. IEEE Vol. 6., N° 2., 1978., pp. 151-166.

[105] Schwan K., Jones A.K.: *Special features resource allocation for multiprocessors.* IEEE Software Vol. 3., N° 3., 1986., pp. 60-78.

[106] Lamport L., Mellon-Smith P.M.: *Synchronizing clocks in the presence of faults.* Journal of the ACM, Vol. 32., N° 1., 1985., pp. 52-78.

[107] Kortez H., Ochsenreiter W.: *Clock synchronization in distributed real-time systems.* IEEE Transaction on Computer, Vol. 36., N° 8., 1987., pp. 933-940.

[108] Lorin H.: *Aspects of Distributed Computer Systems.*
New York, Chichester, Brisbane, Toronto: John Wiley & Sons, Inc., 1980.

[109] Cypser R.J.: *Communications Architecture for Distributed System.*
Massachusetts Menlo Park - California, London, Amsterdam, Sydney: 1978.

[110] Stoy J.T.: *Denotation semantics: The Scot-Starchy approach to programming theory.* MIT Press, 1977.

[111] Baker J.W., Kok J.N.: *Contrasting themes in the semantics of imperative concurrency.* LNCS Vol. 224., 1986.

[112] Savić D.: *Turbo Paskal, naredbe i objekti.* Beograd: Kosmos, 1990.

[113] Бахвалов Н.С., Жидков Н.П., Кобельков Г.М.: *Численные методы.* Москва: Наука, 1987.

[114] Климов Г.П.: *Теория вероятностей и математическая статистика.* Москва: Издательство Московского университета, 1983.

[115] Гнеденко Б.В., Коваленко И.Н.: *Введение в теорию массовного обслуживания.* Москва: Наука, 1987.

[116] Ермаков С.М.: *Метод Монте-Карло и смежные вопросы.* Москва: Наука, 1971.

[117] Marsaglio G., Bray T.A.: *A convenient method for generating normal variables*. SIAM Review, Vol. 6., N° 3., pp. 260-264.

[118] Макаров-Землянский Н.В., Шчепанович С.: *Имитационная модель многовходового адаптера межмашинной связи сети ЭВМ "Нерпа".* В сборнике: Информационное моделирование и проектирование подсистем АСУ., Москва: МГУ, 1988., с. 49-63.

[119] Kunze H.J.: *Physikalische Messmethoden.* Stuttgart: B.G. Teubner, 1986.

[120] Тихомиров В.Б.: *Планирование и анализ эксперимента.* Москва: Лёгкая индустрия, 1971.

[121] Адлер Ю.П., Маркова Е.В., Грановский Ю.В.: *Планирование эксперимента при поиске оптимальных условий.* Москва: Наука, 1976.

[122] Šćepanović S.: *Commutation node in packet switching networks simulation.* VIII Conference on Applied Mathematics, Tivat, May 27-29, 1993. Podgorica: Department of Mathematics of University of Montenegro, 1994., pp. 231-239.

[123] Šćepanović S.: *Analiza računarskog sistema kao čvora komutacije u paketskim mrežama za prenos podataka.* - Magistarski rad Beograd: ETF Univerziteta u Beogradu, 1994.

[124] Šćepanović S.: *Simulacija ponašanja distribuiranog računarskog sistema*, Simpozijum o računarskim naukama i informatici, Brezovica, 4-8. april 1995, Zbornik apstrakata, Brezovica: YU INFO, 1995., s. 155.

[125] Šćepanović S.: *Simulation of the distributed database system.* 9th congress of Yugoslav mathematicians., Petrovac, 22-27. May 1995, Zbornik apstrakata, Podgorica: Yugoslav Association of Mathematics Societies, The Society of Mathematic and Physicists of Montenegro and Faculty of Sci.e, 1995., s. 54.

[126] Šćepanović S., Bojović M.: *Modeliranje paketskog komutacionog čvora.* XXXIX Konferencija ETRAN, Zlatibor, 6-9. jun 1995, Zbornik radova, sveska III, Beograd: ETRAN, 1995., s. 85-88.

[127] Šćepanović S., Bojović M.: *Imitacioni model distribuiranog računarskog sistema.* XL Konferencija ETRAN, Budva, 4-7. jun 1996, Zbornik radova, sveska III, Beograd: ETRAN, 1996., s. 82-85.

[128] Шчепанович С., Леонтьев Д., Мратов А.: *Вопросы определения калибровочных параметров имитационных моделей серверных приложений.* Программные системы и инструменты., Тематический сборник № 1. Москва: Изд-во факультета ВМиК МГУ, 2000., с. 48-55.

PODACI POTREBNI ZA DIGITALIZACIJU DOKTORSKE DISERTACIJE

Ime i prezime autora: **STEVAN ŠĆEPANOVIĆ**

Godina rođenja: **1953.**

E-mail: stevansc@ac.me

Organizaciona jedinica Univerziteta Crne Gore: **PRIRODNO-MATEMATIČKI FAKULTET**

Naslov doktorske disertacije: **ANALIZA JEDNE KLASE DISTRIBUIRANIH RAČUNARSKIH SISTEMA POSREDSTVOM IMITACIONOG MODELIRANJA**

Prevod naslova na engleski jezik: **ANALYSIS OF A CLASS OF DISTRIBUTED COMPUTER SYSTEMS BY MEANS OF IMITATED MODELING**

Datum odbrane: **24. 12. 2002.**

Signatura u Univerzitetskoj biblioteci

Napomena

PODACI POTREBNI ZA UNOS DOKTORSKE DISERTACIJE U DIGITALNI ARHIV UNIVERZITETA CRNE GORE

Prevod naslova disertacije na engleski jezik: ANALYSIS OF A CLASS OF DISTRIBUTED COMPUTER SYSTEMS BY MEANS OF IMITATED MODELING

Mentor i članovi komisija (za ocjenu i odbranu):

Mentor:

Prof. dr Miroslav Bojović, vanredni profesor ETF-a u Beogradu

Komisija za ocjenu doktorske disertacije:

Prof. dr Milojica Jaćimović, redovni profesor PMF-a u Podgorici
Prof. dr Miroslav Bojović, vanredni profesor ETF-a u Beogradu
Prof. dr Milan Martinović, redovni profesor PMF-a u Podgorici

Komisija za odbranu doktorske disertacije:

Prof. dr Milojica Jaćimović, redovni profesor PMF-a u Podgorici
Prof. dr Miroslav Bojović, vanredni profesor ETF-a u Beogradu
Prof. dr Milan Martinović, redovni profesor PMF-a u Podgorici

Sažetak:

Intenzivna primjena distribuiranih računarskih sistema i računarskih mreža, u današnje vrijeme, aktualizira probleme njihove analize, projektovanja i izgradnje. Predmetom ove doktorske disertacije javlja se analiza i modeliranje distribuiranih računarskih sistema izgrađenih na bazi personalnih računara. To su heterogeni distribuirani sistemi kod kojih računari međusobno ne dijele memoriju niti generator takta. Kao komunikaciona sredina koristi se lokalna računarska mreža. Sve ovo ukazuje na aktuelnost i savremenost ovog rada, s obzirom na široku primjenu takvih računarskih sistema.

U radu je Razrađen matematički model koji opisuje funkcionisanje distribuiranog računarskog sistema i koji se principijelno razlikuje od poznatih modela. Za razliku od modela poznatih u literaturi, rad obuhvata ne samo algoritamski aspekt ponašanja aplikativnog programa, već i karakteristike hardverske strukture distribuiranog računarskog sistema, i vrijeme kao parametar. U okviru ovog matematičkog modela proučavaju se uzajamni odnosi različitih aspekata paralelizma unutar distribuiranog računarskog sistema. Učinjen je pokušaj da se na nov način opiše funkcionisanje aplikativnog programa. Na osnovu analize ponašanja distribuiranih aplikativnih programa vrši se procjena performansi distribuiranog sistema. Prilikom izgradnje matematičkog modela korišćen je matematički aparat teorije grafova i teorija automata.

Pored matematičkog u radu razvijen je i imitacioni model. Za opis ponašanja distribuiranih sistema, korišćen je imitacioni model sa porukama. Formulisana je koncepcija kompleksnog prilaza imitacionom modeliranju distribuiranog računarskog sistema. Ovakav prilaz dozvoljava da se u okviru jedinstvenog sistema ispituju kako algoritamske, tako i vremenske karakteristike modeliranja. U cilju izgradnje radnog opterećenja imitacionog modela, razrađena je metoda analize i procjene performansi distribuiranog računarskog

sistema, zasnovana na ponašanju aplikativnog programa i nezavisnosti tog ponašanja od hardverske strukture.

Rezultati ovog istraživačkog rada mogu poslužiti kao polazna osnova za razvitak novog perspektivnog naučnog metoda u istraživanju osobina ponašanja distribuiranih sistema. Važna karakteristika predloženog modela je mogućnost procjene efikasnosti distribuiranih računarskih sistema u fazi njihovog projektovanja, čime se smanjuju troškovi i vrijeme njihove izgradnje. To povećava kvalitet projekta, dozvoljava provjeru odnosa najavljenih (zahtijevanih) ciljeva projekta i njegovih mogućnosti.

Praktični značaj rada ogleda se u izgradnji matematički korektne metode koja:

- dozvoljava procjenu sistemskih performansi distribuiranog računarskog sistema, bez izgradnje prototipa ili emulatora komandi sistema koji se analizira,
- dozvoljava prognozu ponašanja aplikativnog programu u novoj hardverskoj sredini na etapi projektovanja,
- dozvoljava da se tačnost procjene nađe u dopuštenim granicama.

Dobijeni teorijski rezultati provjereni su na distribuiranom računarskom sistemu zasnovanom na arhitekturi klijent/server.

Sažetak na engleskom (njemačkom ili francuskom) jeziku:

Intensive use of distributed computer systems and computer nets nowadays, actualises problems of their analysis, projecting and constructing. With the subject of this dissertation appears analysis and modelling of distributed computer systems built on the basis of personal computers. These are heterogeneous distributed systems in which computers don't share neither memory nor tact generator among themselves. The local computer net is used as a communicative environment, in respect to widespread use of such computer systems.

A mathematical model that describes functioning of distributed computer system is worked out in the dissertation and it principally differs from familiar models. In distinction from models known from literature, this work clasps not only algorithmic aspect of conducting of the applicative programme, but also characteristics of the hardware structure of the distributed computer system and programme, and time as a parameter. Within this mathematical model mutual relations of different aspects of parallelism inside the distributed computer system are being studied. There was an attempt to describe functioning of the applicative programme in a new way and, according the analysis of that conducting, to estimate the performances of the distributed systems. Mathematical apparatus of the theory of graphs and theory of automats were used while constructing the mathematical model.

The imitated model was developed together with the mathematical one in the work. Imitated model with messages was used to describe the conduction of the distributed systems. The conception of complex approach to the imitated modelling of distributed computer system was formulated. That kind of approach allows algorithm, as well as time characteristics of modelling to be checked within one unique system. Aiming at constructing the working load of imitated model, a method of analysis and estimation of the performances of distributed computer system was worked out, based on conducting of the applicative programme and autonomy of that conducting from the hardware structure.

The results of this research work can serve as starting base for development of a new perspective scientific method in researching the characteristics of conducting of distributed systems. An important characteristic of the proposed model is capability of estimated

efficiency of distributed computer systems in the phase of their projecting, by which the costs and time for their constructing are reduced. It increases the quality of the project, allows checking relations of asked aims of the project and its abilities.

The practical importance of the work is seen in the constructing the mathematically correct method which:

- Allows estimation of system performances of the distributed computer system, without making the prototype or commands emulator of the system that is being analysed.
- Allows the prognosis of the applicative programme conduction in a new hardware environment during the projecting.
- Allows that the accuracy of the estimation stays among permitted borders.

Found theoretical results are checked on the distributed computer system based on the architecture client / server.

Ključne riječi: računarske mreže, distribuirani računarski sistemi, ruter, imitaciono modeliranje.

Ključne riječi na engleskom jeziku: computer networks, distributed systems, router, modeling

Naučna oblast/uža naučna oblast: računarske nauke

Naučna oblast/uža naučna oblast na engleskom jeziku: computer science

Ostali podaci

IZJAVA O KORIŠĆENJU

Ovlašćujem Univerzitetsku biblioteku da u **Digitalni arhiv Univerziteta Crne Gore** unese doktorsku disertaciju pod naslovom

ANALIZA JEDNE KLASE DISTRIBUIRANIH RAČUNARSKIH SISTEMA POSREDSTVOM IMITACIONOG MODELIRANJA

koja je moj autorski rad.

Doktorska disertacija, pohranjena u Digitalni arhiv Univerziteta Crne Gore, može se koristiti pod uslovima definisanim licencom Kreativne zajednice (*Creative Commmons*), za koju sam se odlučio.

- ❑ Autorstvo
- ❑ Autorstvo – bez prerada
- ❑ Autorstvo – dijeliti pod istim uslovima
- ❑ Autorstvo – nekomercijalno
- ✓ **Autorstvo – nekomercijalno – bez prerada**
- ❑ Autorstvo – nekomercijalno – dijeliti pod istim uslovima

Potpis doktoranda

U Podgorici
1. 02. 2015. god.

Autorstvo
Licenca sa najširim obimom prava korišćenja. Dozvoljavaju se prerade, umnožavanje, distribucija i javno saopštavanje djela, pod uslovom da se navede ime izvornog autora (onako kako je izvorni autor ili davalac licence odredio).

Djelo se može koristiti i u komercijalne svrhe.

Autorstvo – bez prerada
Dozvoljava se umnožavanje, distribucija i javno saopštavanje djela, pod uslovom da se navede ime izvornog autora (onako kako je izvorni autor ili davalac licence odredio). Djelo se ne može mijenjati, preoblikovati ili koristiti u drugom djelu.

Licenca dozvoljava komercijalnu upotrebu djela.

Autorstvo – dijeliti pod istim uslovima
Dozvoljava se umnožavanje, distribucija i javno saopštavanje djela, pod uslovom da se navede ime izvornog autora (onako kako je izvorni autor ili davalac licence odredio). Ukoliko se djelo mijenja, preoblikuje ili koristi u drugom djelu, prerade se moraju distribuirati pod istom ili sličnom licencom.

Ova licenca dozvoljava komercijalnu upotrebu djela i prerada. Slična je softverskim licencama, odnosno licencama otvorenog koda.

Autorstvo – nekomercijalno
Dozvoljavaju se prerade, umnožavanje, distribucija i javno saopštavanje djela, pod uslovom da se navede ime izvornog autora (onako kako je izvorni autor ili davalac licence odredio).

Komercijalna upotreba djela nije dozvoljena.

Autorstvo – nekomercijalno – bez prerada
Licenca kojom se u najvećoj mjeri ograničavaju prava korišćenja djela. Dozvoljava se umnožavanje, distribucija i javno saopštavanje djela, pod uslovom da se navede ime izvornog autora (onako kako je izvorni autor ili davalac licence odredio). Djelo se ne može mijenjati, preoblikovati ili koristiti u drugom djelu.

Komercijalna upotreba djela nije dozvoljena.

Autorstvo – nekomercijalno – dijeliti pod istim uslovima
Dozvoljava se umnožavanje, distribucija, javno saopštavanje i prerada djela, pod uslovom da se navede ime izvornog autora (onako kako je izvorni autor ili davalac licence odredio). Ukoliko se djelo mijenja, preoblikuje ili koristi u drugom djelu, prerada se mora distribuirati pod istom ili sličnom licencom.

Djelo i prerade se ne mogu koristiti u komercijalne svrhe.