БЪЛГАРСКА АКАДЕМИЯ НА НАУКИТЕ

ИНСТИТУТ ПО ИНФОРМАЦИОННИ И КОМУНИКАЦИОННИ ТЕХНОЛОГИИ

Едита Ананиева Джамбазова

Изследване на надеждностните характеристики на отказоустойчива разпределена система за работа в реално време с настройваема надеждност

ДИСЕРТАЦИЯ

за придобиване на образователната и научна степен „доктор“

по докторска програма „Компютърни системи, комплекси и мрежи“

професионално направление 5.3 Комуникационна и компютърна техника

Консултант:

доц. д-р Румен Андреев

София, 2023 г.

На Краси

СЪДЪРЖАНИЕ

Списък на важни термини и понятия

Безопасна при отказ система	Fail-safe system	Система, чиито откази могат да бъдат или са до приемлива степен само безвредни откази
Блок на ограничаване на неизправност	Fault containment unit (FCU)	Компонент на системата, чиито вътрешни откази не се разпространяват извън него
Времеделене и множествен достъп	Time Devision Multiple Access (TDMA)	Стратегия за достъп до обща комуникационна среда, при която комуникационният цикъл е разделен на интервали и всеки интервал е предназначен за определен възел от разпределена система
Възстановяване от грешка	Error recovery	Форма на обработката на грешка, при която едно състояние без грешка замества грешно състояние
Гарантоспособност	Dependability	Способността на една компютърна система да предоставя услуга, на която обосновано да се разчита
Готовност	Availability	Гарантоспособността по отношение на наличността на системата за използване
Грешка	Error	Демонстрация на неизправност в една система
Действаща при отказ система	Fail operational	Една система се нарича действаща при отказ, когато след отказ на компонент пълното изисквано обслужване все още е осигурено.
Детектирана грешка	Detected error	Грешка, разпозната като такава чрез алгоритъм или механизъм за детектиране
Диагностика на грешка	Fault diagnosis	Действие за определяне на причината за дадена грешка по местоположение и същност
Интензивност на отказите	Failure rate	Брой откази за единица време

Излишък	Redundancy	Наличието на повече от един начин за изпълняване на изискваната функция от даден компонент
Компонент	Component	Самостоятелен градивен блок на дадена система
Коректно обслужване	Correct service	Обслужване в съответствие със системните спецификации
Критичен отказ	Critical failure	Отказ, за който се оценява, че е вероятно да причини нараняване на хора, значителни материални щети или други неприемливи последствия
Маскиране на неизправност	Fault masking	Резултатът от системното прилагане на компенсацията на грешка, дори и при отсъствие на грешка
Меки времеви ограничения	Soft real time (SRT)	Ограничения на времето, при които всички обекти могат да бъдат оптимизирани за най-доброто изпълнение
Мълчаща при отказ система	Fail-silent system	Система, чиито откази могат да бъдат или са до приемлива степен само пълни откази
Надеждност	Reliability	Мярка за изпълняването на непрекъснато коректно обслужване
Заплахи за гарантоспособността	Threats to dependability	Нежелани, но не неочаквани обстоятелства, които причиняват или са резултат от негарантоспособността
		Общ термин за неизправност, грешка и отказ
Неизправност	Fault	Доказаната или хипотетичната причина за грешка
Некоректно обслужване	Incorrect service	Изпълняване на обслужване, което не е в съответствие със системната спецификация
Независими неизправности	Independent faults	Неизправности, които се дължат на различни причини

Обработване на грешка	Error processing	Действията, които се предприемат, за да се отстранят грешките от системата
Осигуряване на гарантоспособността	Procurement of dependability	Методи и техники, предназначени да се придаде на системата способността да изпълнява обслужване в съответствие със спецификациите
Отказ	Failure	Отклонение на изпълняваната услуга от спецификациите
Отказоустойчивост	Fault tolerance	Методи и техники за осигуряване на обслужване в съответствие със спецификациите, независимо от неизправностите
Откриване на грешка	Error detection	Действие за установяване, че едно системно състояние е грешно
Отстраняване на неизправност	Fault removal	Методи и техники за ограничаване на присъствието (по брой и сериозност) на неизправности
Покритие	Coverage	Мярка за представимостта на ситуациите, на които е подложена системата по време на нейното валидиране в сравнение с действителните ситуации, с които тя ще се сблъска по време на своята работа
Постоянна неизправност	Permanent fault	Неизправност, чието присъствие не е свързано с нормалните условия на работа на системата, били те вътрешни или външни
Предотвратяване на неизправност	Fault avoidance	Методи и техники за създаване на система без неизправности
Предпазване от неизправност	Fault prevention	Методи и техники, имащи за цел предпазването от поява или въвеждане на неизправност
Прогнозиране на неизправност	Fault forecasting	Методи и техники за оценка на количеството настъпили неизправности, вероятността за

		настъпване на неизправности в бъдеще и последствията от тях
Пълен отказ	Crash failure	Постоянен отказ тип пропускане
Реалновременна услуга	Real-time service	Услуга, за която се изисква да бъде изпълнена в рамките на крайни времеви интервали, налагани от средата
Реалновременна функция	Real-time function	Функция, за която се изисква да бъде изпълнявана в рамките на крайни времеви интервали, налагани от средата
Режим на неизправност	Fault mode	Наблюдаемо състояние на компонент, което може да предизвика отказ при определени условия на работа
Режим на отказ	Failure mode	Наблюдаваното множество от симптоми, посредством които се разпознава отказът
Ремонтопригодност	Maintainability	Способност да се предприемат модификации и ремонти
Самопроверяващ се компонент	Self-checking component	Компонент, който има механизми за детектиране на грешка, свързани с неговата функционална част
Система за (работа в) реално време	Real-time system	Система, изпълняваща поне една реалновременна функция или поне едно реалновременно обслужване
Система с меки времеви ограничения	Soft real-time system	Система, при която времевите ограничения не са строги, резултатът от изчислението може да бъде полезен дори ако не са спазени ограниченията по време
Система с твърди времеви ограничения	Hard real-time system	Система, в която времевите граници трябва да се спазват задължително или резултатът от изчислението е невалиден
Случайна неизправност	Transient fault	Временна физическа външна неизправност

Средно време до ремонт	Mean Time To Repair (MTTR)	Средното време необходимо за верифициране на неизправността и за нейното отстраняване
Средно време до отказ	Mean Time To Failure (MTTF)	Средно време до поява на първия отказ
Средно време между отказите	Mean Time Between Failures (MTBF)	Средната продължителност на интервалите между последователни откази за зададен период от живота на функционалната единица при зададени условия
Средства за гарантоспособност	Means for dependability	Методи и техники, които дават възможност: а) да се снабди системата със способността да изпълнява надеждно обслужване; б) да се постигне доверие в тази ѝ способност
Твърди времеви ограничения	Hard real-time (HRT)	Дефинират система, при която всички обекти се изпълняват до определено ограничение на времето им за завършване
Физическа неизправност	Physical fault	Неизправност в резултат на неблагоприятни физически явления
Цялостност	Integrity	Състояние на изправност

Списък на фигурите

Списък на таблиците

Списък на използвани съкращения и означения

ДМИ	двоен модулен излишък	
ДН	дърво на неизправностите	
ЕМИ	единичен модулен излишък	
ТМИ	троен модулен излишък	
ОРС	отказоустойчива разпределена система	
A	availability	готовност
COTS	Commercial-Off-The-Shelf	готов (компонент)
MTTF	Mean Time To Failure	средно време до отказ
MTTR	Mean Time To Repair	средно време до ремонт
MTTS	Mean Time To Stop	средно време до спиране
MTBF	Mean Time Between Failures	средно време между отказите
MTBR	Mean Time Between Repair	средно време между ремонтите
MTBS	Mean Time Between Stops	средно време между спиранията
R	reliability	надеждност
WCRT	Worst-Case Response Time	максимално време за изпълнение

Увод

Актуалност на темата

Системите за работа в реално време се прилагат за управление на различни процеси (като индустриални производства, автомобили, авиационни системи и др.), където понятието за време е вградено в цялостния процес на производство. Те са разпределени компютърни системи с всички характеристики, които им позволяват да обработват данни и да обменят информация с външния за тях свят. Те „общуват“ с околната среда посредством сензори и активатори, което им дава възможност да управляват реални процеси. Това определя основната им характеристика – работа при времеви ограничения, наложени от средата. Поради функционирането си между управляван процес в реална физическа среда и компютърната си същност те са определяни като кибер-физични системи. Системите за работа в реално време обикновено са разпределени, което означава, че са съставени от самостоятелни компоненти, които комуникират помежду си през комуникационен канал. Те често са свързани с критични за безопасността приложения и към тях се поставят високи изисквания за гарантоспособност, които се вземат предвид още на етапа на тяхното проектиране. Гарантоспособността е интегрално понятие, което определя доверието в способността на една система да доставя коректна услуга. Всички тези аспекти на реалновременните системи – кибер-физични, разпределени, работещи с ограничения по време, отказоустойчиви – прави проектирането им сложно и многообхватно. Това поражда редица изследователски проблеми, които през годините и с развитието на технологиите са намирали различни решения. Изискването за отказоустойчивост е част от процеса на проектиране. Реалновременната функция определя гарантоспособните разпределени системи да работят с глобална времева база, според която да синхронизират всички операции, като предоставят коректна услуга както в областта на стойностите, така и в областта на времето. Кибер-физичната им природа изисква да съчетават разнообразни изисквания. Често изискванията за отказоустойчивост и работа в реално време са трудно съвместими и се налага търсенето на приемлив компромис между тях.

Отказоустойчивостта е неотменимо свойство на тези системи. Тя се постига чрез прилагането на различни подходи и техники за откриване на грешки и възстановяване от тях. В голяма степен те се основават на понятието за излишък. Излишъкът е елемент от структурата на дадена система, без който тя може да изпълнява основните си функции и който подпомага функционирането ѝ в случай на промяна в работната ѝ среда, породена от настъпването на неизправности. Управлението на излишъка е важно за отказоустойчивите системи, защото той

води до повишаване на техните надеждностни характеристики, но в същото време внася допълнителни елементи, които имат своята цена от гледна точка на производителността и разходите за системата. Въвеждането на структурен излишък с цел повишаване на гарантоспособността води до допълнителни закъснения за синхронизация, възстановяване след отказ, включване и изключване на нови компоненти и т.н., което усложнява постигането на изискванията за работа в реално време. Намирането на компромис е сложна задача и е предмет на сериозни изследователски усилия. Търсенето на нов подход за управление на излишъка в гарантоспособни разпределени системи мотивира това дисертационно изследване.

Мотивация

Изискванията за гарантоспособност на реалновременните системи при критични приложения и повишаването на разходите на системата с цел управление на излишъка мотивира идейния замисъл на дисертацията да се създаде архитектура на отказоустойчива разпределена система за работа в реално време, която да позволява разпределение на структурния излишък според изискванията на приложението – наречена от автора *система с настройваема надеждност*. Основният изследователски въпрос е дали могат отказоустойчивите разпределени компютърни системи за работа в реално време да постигнат гъвкавост по отношение на изискванията за надеждност на приложението чрез предложения в дисертацията подход на настройваема надеждност.

Предложената в дисертационния труд архитектура на отказоустойчива разпределена система за работа в реално време с настройваема надеждност е изградена от самостоятелни отказоустойчиви компоненти с различна модулност, съобразена с тяхната критичност. Критичността на компонентите се определя от тежестта на последствията от техен отказ за управляваната система. Системата е моделирана посредством разработена за целта симулационна програма и са изследвани надеждностните ѝ характеристики при различни параметри. Предложеният в дисертацията научно-приложен подход, наречен подход на настройваема надеждност, определя разпределение на хардуерния структурен излишък, съобразено с изискванията на приложението за обща системна надеждност. Системата и подходът на настройваема надеждност предлагат постигане на висока надеждност посредством разпределение на системните хардуерни ресурси по начин, който е съобразен с нуждите на приложението. Това прави отказоустойчивата разпределена система с настройваема надеждност удобна за внедряване в области с разнообразни изисквания за надеждност и смесена критичност на компонентите, във вградени системи, при по-малко отговорни приложения и пр.

Подходът на настройваема надеждност има предимства пред други широко известни разработки при: постигане на по-висока надеждност със същите ресурси, постигане на висока готовност, гъвкавост при разпределението на системните ресурси на етапа на проектиране, приложимост в компактни системи. Моделирането на предлаганата разпределена система с настройваема надеждност и сравнението ѝ с моделите на подобни системи показва, че има възможност по-добре да се разпределят ресурсите на системата при запазване на добри нива на нейните надеждностни характеристики.

Научна постановка на изследването

Обект на изследването са отказоустойчиви разпределени системи за работа в реално време.

Предмет на дисертацията е настройваема надеждност в отказоустойчиви разпределени системи за работа в реално време

Цел на дисертационния труд е да се изследват надеждностните характеристики на предложената от автора отказоустойчива разпределена система за работа в реално време с настройваема надеждност, като те се съпоставят с познатите подобни системи, и на тяхна основа да се разработи подход (на настройваема надеждност) за използване в отказоустойчиви системи за работа в реално време.

Хипотеза

Гарантоспособните разпределени системи постигат отказоустойчивост по много различни начини и на различни нива от системната архитектура. Водещ метод за изграждането им е въвеждането на структурен излишък. Съществуват два основни подхода за прилагане на структурен излишък – посредством специализирани хардуерни и софтуерни компоненти и посредством използване на готови софтуерни и хардуерни компоненти. И при двата подхода отказоустойчивостта срещу физически неизправности се постига чрез репликиране на хардуерните компоненти и се търси гъвкавост при репликирането на софтуерните компоненти. Хипотезата, която се поставя в настоящата дисертация, е, че може да се постигне висока надеждност и гъвкавост на разпределението на системните ресурси посредством настройваема надеждност, реализирана с разпределение на хардуерния структурен излишък.

Доказателствата на хипотезата се измерват чрез верифициране на следните твърдения:

1. Отказоустойчивата система с настройваема надеждност постига висока обща надеждност, съпоставима с надеждността на системи без разпределение на структурния излишък.

2. Съществуват конфигурации на системата с настройваема надеждност, при които се постигат по-добри надеждностни характеристики от тези на системи без разпределение на структурния излишък.
3. Може да се идентифицират условията, при които отказоустойчивата система с настройваема надеждност има по-добри надеждностни характеристики от сравняваните системи.

Методология на изследването

В дисертацията са застъпени основните прийоми на научното познание - анализ, синтез, сравнение и обобщение. Направен е обзор на гарантоспособните разпределени системи от гледна точка на разпределението на структурния излишък, като са откроени техните предимства и недостатъци. На базата на този критичен анализ е поставена целта на настоящото изследване и е формулирана основната хипотеза. Предложен е концептуален модел за вземане на решения при вграждане на гарантоспособност в системи. Въз основа на класификацията на гарантоспособни разпределени системи е направена връзка между жизнения цикъл на разработване на системи и проектирането на системи за критични приложения.

След проучването на съществуващи гарантоспособни системи е предложена архитектура на отказоустойчива система с настройваема надеждност. Направен е преглед на методите за моделиране на гарантоспособни системи и е избран и обоснован изследователски подход – симулационно моделиране. За неговата реализация е създаден програмен продукт, с който са проведени множество експерименти. Получените резултати са систематизирани и анализирани и с тяхна помощ е разработен подход на настройваема надеждност, чрез който да се определят системните конфигурации с обща надеждност според изискванията на приложението.

Основни задачи на изследването:

1. Да се направят проучване, обзор и критичен анализ на гарантоспособни разпределени системи. Да се синтезира класификация на съществуващите гарантоспособни разпределени системи. Да се очертаят изследователски възможности при разпределение на структурния излишък.
2. Да се предложат модел и архитектура на отказоустойчива разпределена система с настройваема надеждност, които дават решение на изискванията за висока надеждност според нуждите на приложението.

3. Да се дефинира метод за изследване на предложения модел. Да се разработи инструмент, реализиращ този метод. Да се състави изследователски протокол.
4. Да се проектират и проведат експериментални изследвания за тестване и анализ на надеждностните характеристики на предложената отказоустойчива система с настройваема надеждност посредством избрания изследователски подход и реализирания програмен продукт. Да се разработи и приложи подход на настройваема надеждност.

Структура на съдържанието

Дисертационният труд е организиран в увод, четири глави, заключение, библиографска справка и две приложения.

В *Увода* са посочени темата, обектът и предметът на дисертационния труд. Описана е накратко актуалността на темата и мотивацията за извършване на дисертационното изследване. Поставена е целта на изследователската работа и задачите, чрез които тя да бъде постигната, водещата хипотеза и приложената при проведените изследвания методология.

В *Глава 1* са представени основополагащите понятия, свързани с гарантоспособните разпределени системи за работа в реално време. Описани са основните методи и техники за постигане на отказоустойчивост. Разгледани са начините за въвеждане на излишък и неговото управление. Представен е обзор и критичен анализ на познатите гарантоспособни разпределени системи. Изведен е концептуален модел на подход за вземане на решение при осигуряване на гарантоспособност и е синтезирана класификация на гарантоспособни разпределени системи. Очертани са възможностите за нови изследвания.

В *Глава 2* е представена архитектурата на предложената от автора отказоустойчива разпределена система с настройваема надеждност. Представени са методите за моделиране на гарантоспособни разпределени системи, както и модела и допусканията на отказоустойчивата система с настройваема надеждност. Там са описани изследваните надеждностни характеристики, въз основа на които системата може да бъде оценявана и сравнявана с други подобни системи. Обоснован е изборът на изследователски подход – симулационно моделиране.

В *Глава 3* са описани изследователските задачи и са представени резултатите от симулационно изследване на отказоустойчивата разпределена система с настройваема надеждност. Представен е програмният продукт за симулационно моделиране на системата с настройваема надеждност. Изследвани са надеждностните характеристики на компонент на системата и на цялата система: надеждност, готовност, средно време до отказ, средно време за

ремонт и т.н. Представен е разработеният в дисертацията подход на настройваема надеждност, който позволява избиране на подходяща конфигурация на структурния излишък в зависимост от изискванията за обща системна надеждност на приложението.

Глава 4 представлява анализ и обсъждане на резултатите. Посочени са предимствата и възможните приложения на предложената отказоустойчива система с настройваема надеждност. Изведени са основните научни и научно-приложни резултати на дисертацията. Очертани са възможностите за по-нататъшни изследвания и приложение на отказоустойчивата разпределена система с настройваема надеждност.

Дисертационният труд завършва със *Заключение*, в което се обобщават получените резултати.

В края е посочена *Библиография*, съдържаща 102 източника.

В *Приложение А* са изведени математическите представяния на надеждностните характеристики, с които борави изследването.

В *Приложение В* е представен кодът на програмата за симулационно моделиране NMRSIM.

Глава 1 Гарантоспособни разпределени системи за работа в реално време

1.1 Гарантоспособност – основни понятия

Понятието гарантоспособност[1] (на англ. dependability) е въведено от Жан-Клод Лапри през 80-те години на 20. в. [1], [2], за да се обхванат различните аспекти на отказоустойчивите системи и да се въведе системност в използването на понятията, свързани със защитата от откази на високонадеждните системи. Основната дефиниция за *гарантоспособност* [2], [3], [4] гласи, че „Гарантоспособност е способността на една компютърна система да доставя услуга, на която може обосновано да се разчита“[2]. Тази дефиниция поставя ударението върху обоснованото доверие в услугата, предоставяна от системата. В [3] е добавена и втора дефиниция на гарантоспособност, която подчертава значението на предотвратяването на откази: „Гарантоспособност на дадена система е способността ѝ да предотвратява откази в услугата, които са по-чести и по-тежки от допустимото“.

Специфичната терминологична основа, използвана в дисертацията, са утвърдените и широко прилагани в областта на гарантоспособните системи понятия и определения [3], [4], [5] и техните български еквиваленти [6].

Услуга (service) е системното поведение от гледна точка на потребителя на системата. Потребителят може да бъде и друга система.

Коректна услуга (correct service) се предоставя, когато услугата прилага системната функция.

1.1.1 Заплахи за гарантоспособността: откази, грешки, неизправности

Заплахите за компютърните системи са причините, които водят до отклонение от коректното им функциониране. Проявата на това отклонение на системно ниво се нарича отказ на услугата [3]. *Отказ на услугата* (service failure) или просто *отказ* (failure) е събитие, което настъпва, когато предоставяната услуга се отклонява от коректната. Отказ в услугата настъпва или поради отклонението ѝ от функционалните спецификации, или защото спецификациите не описват адекватно системната функция. Отказът в услугата е преход от коректна към

[1] Преводът на основните термини, свързани с понятието гарантоспособност, е направен от колектива на секция „Отказоустойчиви компютърни системи“ на Института по компютърни системи (ИКС – БАН) [6]. По-съвременната им интерпретация и превод са на автора.

[2] „Гарантоспособност е свойството на една компютърна система, което позволява да се разчита по доказан начин на обслужването, изпълнявано от нея“ [6].

некоректна услуга, т.е. преход към неизпълнение на системната функция. Периодът на предоставяне на некоректна услуга е *отсъствие на услуга* (service outage). Преходът от некоректна към коректна услуга се нарича *възстановяване на услугата* (service restoration). Отклонението от коректната услуга може да приеме различни форми, наречени режими на отказ и са подредени според *сериозността на отказа* (failure severity) - степента на последствията на отказа за системната среда.

Тъй като услугата е поредица от външните състояния на системата, отказ в услугата означава, че поне едно (или повече) външни състояния на системата се отклонява от състоянието на коректна услуга. Това отклонение се нарича грешка. Доказаната или хипотетичната причина за грешка се нарича *неизправност* (fault). Неизправностите могат да бъдат външни или вътрешни за системата.

Определението за *грешка* (error) е частта от общото състояние на системата, която е възможно да доведе до отказ. Грешката е демонстрация на неизправност в една система. Важно е да се отбележи, че много грешки не достигат до външното състояние на системата и не причиняват отказ. Неизправността е *активна* (active), когато причинява грешка, в противен случай тя е *скрита* (dormant).

1.1.2 Атрибути и средства на гарантоспособността

В основополагащата статия [3] са представени основните понятия и дефиниции, свързани с гарантоспособността. Те се използват и в настоящата работа, като тук са цитирани само дефинициите, които имат отношение към темата на дисертацията. Според наложилата се през последните тридесет години терминология гарантоспособността е интегрално понятие, което се характеризира със следните атрибути:

- *Готовност* (availability): наличност на системата за предоставяне на коректна услуга;
- *Надеждност* (reliability): непрекъснатост на коректната услуга;
- *Безопасност* (safety): отсъствие на катастрофални последствия за потребителя и средата;
- *Цялостност* (integrity): отсъствие на неправилни изменения на системата;
- *Ремонтопригодност* (maintainability): способност да се предприемат модификации и ремонти.

Средствата за постигане на гарантоспособност на компютърните системи са [2], [3]:

- *Предпазване от неизправност* (fault prevention): методи и средства, имащи за цел предпазването от поява или въвеждане на неизправност;

- *Отказоустойчивост* (fault tolerance): методи и средства, имащи за цел предотвратяването на откази в присъствието на неизправности;
- *Отстраняване на неизправност* (fault removal): методи и средства за ограничаване на броя и сериозността на неизправностите;
- *Прогнозиране на неизправност* (fault forecasting): методи и средства за оценка на количеството настъпили неизправности, вероятността за настъпване на неизправности в бъдеще и възможните последствия от тях.

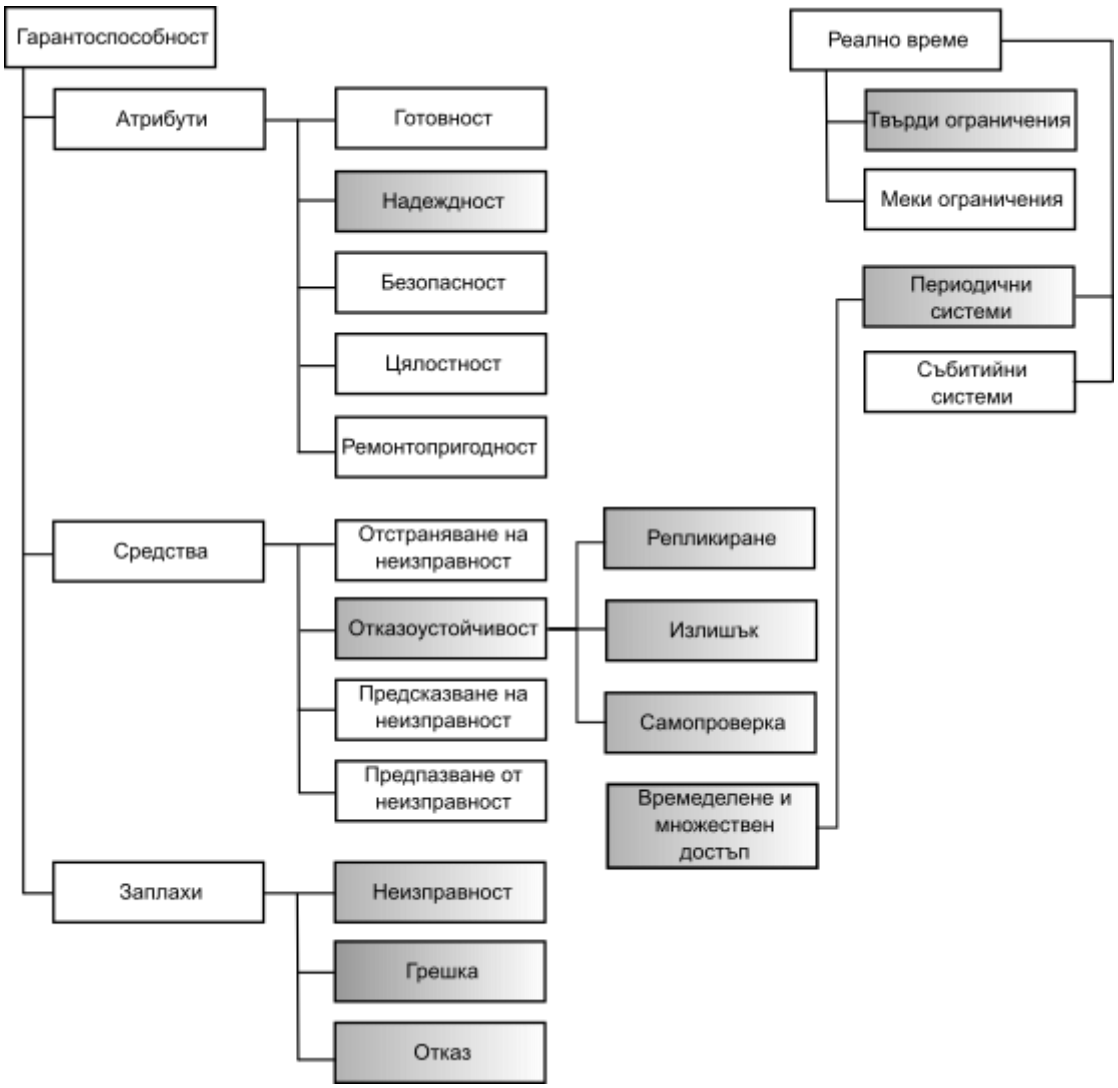
Фокусът на дисертационния труд е върху постигането на отказоустойчивост на разпределени компютърни системи за работа в реално време. Това е онагледено на *Фигура 1-1,* която представя синтез на основните понятия, свързани с гарантоспособността на компютърни системи за работа в реално време. В сиво са отбелязани онези от тях, които имат отношение към дисертацията.

Използваните понятия (*Фигура 1-1*) са обединени в две групи: понятия, свързани с гарантоспособността, и понятия, свързани с реалното време. От областта на гарантоспособността съществен интерес за настоящата работа представлява отказоустойчивостта и методите и техниките за нейното постигане в разпределени системи за работа в реално време. От областта на системите за работа в реално време фокусът в настоящата работа е върху периодични системи, работещи с твърди ограничения по време.

Отказоустойчивостта е свързана с продължаването на работата на системата, въпреки наличието на неизправности в нея. В [6] определението е „Методи и техники за осигуряване на обслужване в съответствие със спецификациите, независимо от неизправностите“. Дефиницията на отказоустойчивост в [3], която се използва в дисертационния труд, е

„Отказоустойчивост е предотвратяване на отказ чрез откриване на грешки и възстановяване на системата“.

Откриването на грешки се осъществява чрез разнообразни техники, които използват подходящи контролни точки от функционирането на системата за извършване на проверка за коректност. Могат да се контролират потока от инструкции на процесора [7], [8], [9], [10], времето за изпълнение на задача/програма [11], [12], [13], [14], [15], [16], крайният резултат на управляващата програма (кодове с излишък, приемащи тестове, сравнение) и т.н. Те често се обединяват под общото наименование *блокове за самопроверка* (self-checking units) [17].



Фигура 1-1. Структура на основните понятия, отнасящи се към гарантоспособността на системи за работа в реално време

Неразривно свързано с понятието за отказоустойчивост е понятието за *излишък* (redundancy) (*Фигура 1-1*). „Излишък е наличието на повече от един начин за изпълняване на изискваната функция от даден компонент“ [6].

Излишъкът в компютърните системи дава възможност системата да продължи да работи, въпреки неизправностите. Неизправностите съпътстват функционирането на системите и са част както от работната среда, така и от самата система. Те са неизбежни, моментът на възникването им е неизвестен и, за да може системата да изпълнява коректно своето обслужване, тя трябва да е в състояние да ги толерира. Това се постига с въвеждането на излишък. Техниките за въвеждане на излишък са обхванати в понятието репликиране (replication). *Репликирането* е размножаване на елементи на системата с цел постигане на отказоустойчивост. Репликират се еднотипни елементи. В зависимост от мястото на прилагане

могат да се добавят и елементи с различна структура, но сходни функции. Видовете излишък и техниките за репликиране са разгледани в т. 1.4.

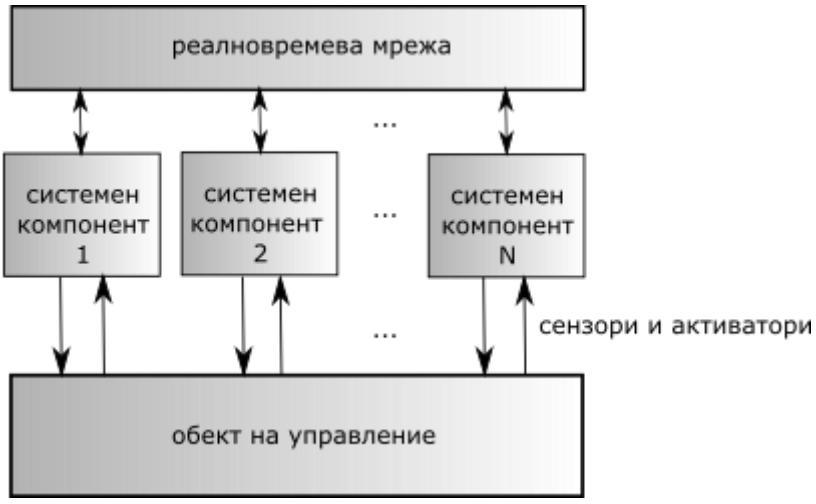
При разпределените системи за работа в реално време има два съществени аспекта: разпределеността и реалното време. *Разпределените системи* в общия случай представляват набор от автономни изчислителни елементи, който за своите потребители изглежда като една кохерентна система [18]. Автономните елементи, наречени възли, изпълняват обща задача, като разпределят дейността си. Възлите са независими елементи, които комуникират помежду си по обща съобщителна среда. Когато разпределените системи се използват за управление на процеси, те трябва да съобразяват своята работа с ограниченията за време на тези процеси. *Система за работа в реално време* (или за краткост „система за реално време“) е такава система, която изпълнява поне една реалновременна функция или поне една реалновременна услуга. *Реалновременната функция* [6] е функция, която се изпълнява в рамките на крайни времеви интервали, налагани от средата. *Реалновременната услуга* [6] е услуга, която се изпълнява в рамките на крайни времеви интервали, налагани от средата. Разпределените системи за реално време обикновено се внедряват в критични по отношение на безопасността приложения, където щетите от системен отказ могат да имат катастрофални последствия. Пример за такива приложения са космическите системи, авиационните системи, автомобилите, химическите процеси, ядрените централи и др. При тях се изисква висока гарантоспособност във всичките ѝ аспекти.

Системите за реално време обработват входна информация от околната средата (обекта на управление) и пресмятат съответната изходна информация в рамките на горна времева граница (относителна граница), която е наложена от изискванията на приложението. Следователно е необходимо поведението на системата по отношение на времето да бъде съобразено с максимално време за изпълнение на приложните задачи. *Максимално време за изпълнение* (worst-case response time - WCRT) на приложна задача е времевият интервал между пристигането на заявка от управляваната система и завършването на необходимата обработка [17]. Относителната времева граница може да се определи като максималния интервал между пристигането на заявката и завършването на съответната обработка.

1.2 Разпределени системи за работа в реално време

Разпределените системи са изградени от компоненти, които обменят съобщения през комуникационна магистрала (реалновременна мрежа) и изпълняват общ алгоритъм за управление (*Фигура 1-2*). От гледна точка на отказоустойчивостта компонентите на системата трябва да осигуряват неразпространение на неизправностите към други компоненти.

Понятието „блок на ограничаване на неизправност“ отразява това изискване. Компонентът на системата е *блок на ограничаване на неизправност* (fault-containment unit – FCU) [17], [19], ако прекият ефект от единична неизправност влияе само върху функционирането на един-единствен компонент [17]. Допуска се, че блоковете на ограничаване на неизправност отказват независимо един от друг. Това допускане е приложимо за хардуерни неизправности, които са обект на настоящото изследване. Елементите на разпределените системи се наричат възли. *Възелът* (node) е обособен блок от разпределена система със самостоятелна задача и достъп до обща комуникационна среда. Понятието „възел“ има повече инженерно измерение, а в дисертацията се използва терминът „компонент“ за функционален блок на системата.



Фигура 1-2. Разпределена система за работа в реално време

Системите за реално време обикновено са разпределени, затова термините „система за реално време“ и „разпределена система“ се използват взаимнозаменяемо в дисертацията.

Системата управлява индустриален процес, наречен обект на управление. Компонентите получават входни данни от сензорите на обекта на управление, изпълняват управляваща програма, която изчислява резултати, и извеждат тези резултати към изпълнителните механизми (активаторите) на обекта на управление (*Фигура 1-2*). За да изпълняват задачите си, те трябва да комуникират с останалите компоненти чрез обмен на данни. Системните компоненти са проектирани да имат безопасно поведение, т.е. нито една неизправност не трябва да достига до изходите на компонента, както и да се разпространява към други части на системата. Това гарантира нейната надеждна работа.

Разпределените системи за работа в реално време работят с крайни времеви интервали, налагани от средата и обекта на управление. При тях коректната услуга трябва да бъде коректна в областта на стойностите и в областта на времето [17]. Това означава системата да предоставя коректен резултат на обекта на управление и да го извежда в рамките на

специфицирания времеви интервал. Ако резултатът бъде коректен по стойност, но бъде изведен по-рано или по-късно от времевата граница, той се отклонява от спецификациите и следователно като цяло е некоректен. Когато времевите интервали в системата за реално време изискват стриктно спазване, за да бъде доставена коректна услуга, системата за реално време е с *твърди времеви ограничения* (hard real-time) [17]. В противен случай тя е система за реално време с *меки времеви ограничения* (soft real-time) [17]. Често системите работят и при двата вида ограничения едновременно, но наличието на поне една функция с твърди ограничения по време прави системата система с твърди времеви ограничения.

Друго съществено разграничение на системите за реално време е в зависимост от задействащия фактор, който определя взаимодействията между системните компоненти. Системите, при които началото на някакво действие по отношение на времето е съществено събитие в системата, се наричат *събитийни разпределени системи* (event-triggered distributed systems) [17]. Системите, при които иницииране на действие се осъществява според определен момент от течението на физическото/реалното време, се наричат *периодични разпределени системи* (time-triggered distributed systems) [17], [20]. Гарантоспособните разпределени системи често изпълняват функции с твърди времеви ограничения и са периодични. Това дава възможност тяхното поведение да бъде предсказуемо и позволява да се използват по-икономично ресурсите им. Този вид системи са обект на настоящото изследване.

За постигането на отказоустойчиво поведение на компонентите в гарантоспособните разпределени системи те се конструират с репликирани модули [17], [21], [22], [23], [24]. *Модул* (module) е най-малката заменима единица на системата. Това понятие има отношение към техниките за репликиране и метода на въвеждане на излишък в системата. Репликирането на ниво компонент може да бъде хардуерно или софтуерно реализирано. Добавят се и допълнителни средства за откриване на грешки към всеки модул [25], [26], [27], [28] – блокове за самопроверка. Самата комуникационна магистрала също може да бъде репликирана [25], [29], [30], [31]. Разнообразието от техники за репликиране дава възможност за избор на най-подходящите решения за конкретно приложение.

1.3 Управление на излишъка в отказоустойчиви разпределени системи за работа в реално време

Гарантоспособните разпределени системи за работа в реално време обикновено са предназначени за критични по отношение на безопасността приложения (safety-critical applications). Едно от основните изисквания за тяхната работа е да бъдат отказоустойчиви.

Отказоустойчивостта се постига чрез използване на разнообразни техники, повечето от които се основават на някаква форма на излишък. Традиционно излишъкът в гарантоспособните разпределени системи за работа в реално време се прилага на ниво компонент на системата и при комуникациите, но съществуват различни видове излишък и техники за прилагането му. Репликиране се прилага, например, в системите за защита на електроснабдителните системи, в блокиращите системи в ж.п. транспорта, при автономните автомобили [32], в критичните инфраструктури и т.н.

Въвеждането на излишък в компютърните системи, като метод за постигане на отказоустойчивост, и репликирането, като техническа реализация на излишъка, са добре познати и проучени [23], [24], [33], [34], [35], [36], [37], [38]. Въпреки че управлението на излишъка е разработено и приложено в различни отказоустойчиви системи отдавна, проектирането на нови гарантоспособни разпределени системи [32], [39], разработването на системи от системи и кибер-физични системи [40], [41] налагат нов поглед и търсене на нови подходи за реализиране на излишъка. В критичните инфраструктури (енергийна система, водоснабдителна система, електрическа система и др.), например, критичните елементи на системата, като системата за надзорно управление и събиране на данни (Supervisory Control and Data Acquisition - SCADA), се реализират чрез използване на структурен излишък [39].

При проектирането на критични за безопасността системи най-важното изискване е в тях да не настъпват катастрофални откази, а това се постига чрез налагане на строги изисквания за гарантирано и предсказуемо поведение на компонентите и системата, регламентиране на взаимодействията по отношение на времето и желания резултат и въвеждане на механизми за отказоустойчивост на различни нива в системната архитектура. Тези изисквания се постигат посредством прилагането на по-консервативни подходи, които водят до разработване на специализирани компоненти, които отговарят на определени стандарти за нива на безопасност, например IEC61508 [42], ISO26262 [43] и др. Разпределението на структурния излишък е фиксирано, разписанието на задачите е предварително зададено, системите работят при твърди времеви ограничения и т.н. Това дава възможност системите да бъдат по-лесно верифицирани и валидирани, но в същото време ги прави по-малко гъвкави по отношение на промени в средата на експлоатация. Възможностите за приложение на репликирането могат да се разширят и това да доведе до оптимални и ефективни решения, съобразени с конкретната област на приложение. Затова е необходимо да се покаже, че методите за реконфигуриране на излишъка могат да доведат до съществени подобрения в сравнение с традиционните методи за отказоустойчивост, каквато е и хипотезата на настоящата дисертация.

Вграждането на средства за отказоустойчивост в гарантоспособните разпределени системи означава получаване на гаранции за откриване и отстраняване на грешки. Въвеждането на излишък е ключов метод за постигане на отказоустойчивост. Както предполага името му, излишъкът е нещо, без което системата може да работи. Повечето системи изпълняват функциите си коректно без използването на допълнителни елементи. За гарантоспособните системи обаче, където има изискване за висока системна надеждност дори и в присъствието на неизправности, една от традиционните техники за прилагане на излишък е репликирането, т.е. въвеждането на допълнителни елементи към основните.

Въвеждането на излишък води до допълнителни ресурси и следователно увеличава разходите за внедряване на системата. Затова то се прилага в системи, които трябва да гарантират, че вероятността за пълен системен отказ е минимална в рамките на практическото използване на системата. Допълнителните разходи за „излишните“ елементи са обосновани от потенциалните последствия от системен отказ, който може да причини тежки щети на оборудването, загуба на критични функционалности или дори наранявания или загуба на човешки живот. Изследователите, проектантите и инженерите търсят начини за постигане на желаната гарантоспособност на минимална цена.

Гарантоспособните разпределени системи за реално време използват техники за репликиране на различни нива в системата. Отказоустойчивостта е интегрирана още на ниво системен проект. Репликирането се прилага към компонентите, комуникационния канал, задачите и т.н. Къде да бъдат използвани репликирани компоненти, се решава на етапа на проектирането на гарантоспособната система.

1.3.1 Проектиране на системата

Цикълът на разработване, наречен още развой, на разпределената система може да бъде представен като итеративен процес [44]. Той разглежда системата от две гледни точки: практическа и абстрактна. Практическата гледна точка към системата е нейното внедряване. В своята работна среда системата е подложена на разнообразни смущения за функционирането, които ѝ пречат да работи коректно. Абстрактният поглед върху системата е нейният модел. Системният модел взема под внимание системните спецификации, предвиденото приложение, желаната функционалност и системната архитектура. За да бъде проектирана системата, е необходимо и теоретично познание, което да обоснове нейната коректна работа. Тези гледни точки трябва да обменят информация помежду си, за да постигнат цялостен системен модел, който може да бъде валидиран и верифициран.

Съществуват два подхода за проектиране на една система: отгоре-надолу и отдолу-нагоре[3]. Стратегията „отгоре-надолу“ се движи от високото ниво на абстракция, преминава през прилагането на определени изисквания и достига до модел, който се реализира. Стратегията „отдолу-нагоре“ предполага първоначална реализация на работещ вариант на системата. В последствие се преминава към нейното формализирано представяне (модел), който е основа за аргументирано въвеждане на необходими подобрения.

Стратегията „отдолу-нагоре“ установява наличието на проблем в една работеща система, който е необходимо да бъде решен. Той обикновено е породен от реалната среда на работа на системата, която например може да предизвика някаква повреда в нея. В контекста на гарантоспособните системи изследваният проблем би бил гарантоспособността като общо понятие (*Фигура 1-3*). Проектираната система трябва да бъде гарантоспособна, тъй като ще работи в критични приложения. Системните инженери се опитват да намерят решение на този проблем. Първоначално те трябва да разберат същината на проблема, т.е. да видят неговата структура. За да постигнат отказоустойчивостта, изисквана от приложението, те трябва да решат кои методи са подходящи. Определя се необходимата за приложението системна надеждност като един от атрибутите на гарантоспособността [2], [3], [35] и съществен показател за нейната коректна работа. Избират се методи за отказоустойчивост. В голяма степен те се основават на въвеждането на излишък. Видовете излишък и начините за неговото въвеждане са разгледани в т. 1.4.

Системните проектанти разработват стратегия за достигане до решение на проблема. В това усилие те се нуждаят от налични теоретични и практически познания, които са верифицирани. Изследователските методи и модели могат да подпомогнат прилагането на подходящи решения. В разпределените системи за реално време се използват много техники за отказоустойчивост. Повечето от тях се основават на излишък, например N-модулен излишък, N-версионно програмиране, разнообразяване на софтуера, блокове за възстановяване и др., които са разгледани по-обстойно в т. 1.4.

За да изградят стратегия за решаване на проблема, разработчиците се съобразяват със специфични свойства на работната среда на системата и нейния контекст. Контекстът е областта на приложение на разглежданата система. Той влияе на онези характеристики на средата, които имат отношение към разработваната система. Резултатът от този итеративен процес е план за проектиране на системата, който е решение на първоначалния проблем.

[3] Термините „отгоре-надолу“ и „отдолу-нагоре“ са използвани в най-общ смисъл, за да подчертаят различните гледни точки при проектирането на една система от по-общото към по-конкретното, и обратно.

На *Фигура 1-3* е представена концепция за осъществяване на подход за вземане на решение по отношение на осигуряване на гарантоспособността на една система чрез използване на излишък.

Фигура 1-3. Концептуален модел на подход за вземане на решение при осигуряване на гарантоспособност

От гледна точка на излишъка описаният процес изглежда по следния начин (*Фигура 1-3*). Дадена разпределена система управлява промишлен процес, т.е. разпределена компютърна среда. Заедно с въпросите, засягащи комуникацията между съставните ѝ компоненти и предоставянето на коректна услуга, системата е подложена на неизправности. Тези неизправности нарушават нейната работа и застрашават управлявания процес, което води до нежелани последствия. Проблемът с гарантоспособността на системата става въпрос на проектирането: как да бъде направена системата отказоустойчива. Неизправностите биха могли да имат много причини и биха могли да възникнат при различни условия, моментът на появата им е неизвестен. Освен това, те са неизбежни, винаги присъстват в системите. Тъй като неизправностите са непредсказуеми, системата трябва да има ресурси, с които да доставя услугата, за която е предназначена, дори в присъствието на неизправности. Въвеждането на излишък е една от стратегиите за решаване на проблема с необходимата отказоустойчивост. Това предполага, че при отказ на един компонент неговата функция се поема от неговия резервен компонент и по този начин системата продължава да работи според своите спецификации. Инженерните въпроси с прилагането на излишъка, като координиране между репликираните модули, проектиране на средства за самопроверка, избор на режим на неизправност/отказ (избор на вида неизправности, които ще толерира системата) и т.н. трябва да бъдат съчетани с изследователски решения, например дефиниране на системния модел,

разработване на модели на компонентите, изследване на различни работни сценарии с моделите, техники за отказоустойчивост, оценка на надеждността и др. На *Фигура 1-3* те са обединени под общото наименование оценка на гарантоспособността. Моделите и техните параметри зависят от приложението (например кибер-физични системи, критични за безопасността системи, автомобили и др.) и от работната среда, т.е. разпределената компютърна среда. Атрибутите на гарантоспособността се определят въз основа на конкретното приложение. Резултатите, получени от изследването на моделите, се използват при проектирането на системата. Определя се подходящата техника на репликиране.

Репликирането на компоненти може да бъде разглеждано през призмата на стратегията за проектиране на системата. След идентифицирането на подходящата техника за репликиране процесът на проектиране може да влезе в известните схеми на системно проектиране. Съществуват добре установени подходи за проектиране на системи въобще и на инженерни системи в частност. Описаният общ подход дава поглед върху проектирането на гарантоспособни системи и спомага за намирането на възможности за прилагане на излишък в гарантоспособни разпределени системи. Той предлага начално познание за процеса на проектиране на системата, като въвежда гледната точка на отказоустойчивостта.

Съществуващите модели за проектиране на системи помагат да бъде конструирана система от първоначалната идея до крайното внедряване [18]. Процесът е стандартизиран и широко прилаган при разработването на системи с желаните свойства. Той е известен като жизнен цикъл на разработване на системи (System Development Life Cycle - SDLC). Жизненият цикъл на разработване на системи се прилага със следните етапи на проектиране: планиране, анализ, проект, внедряване и поддръжка. Много модели включват и етапа на тестване преди внедряването, тъй като той е важна част от процеса. По време на етапа на планиране се идентифицират концепцията и общите изисквания към системата. С напредването на процеса се дефинират функционалните изисквания към системата. На тяхна основа се разработва системната архитектура. Развойният процес навлиза в повече подробности и започва проектирането на системните хардуер и софтуер. Системата е готова за внедряване, но трябва да премине през тестване на компоненти, интеграционно тестване, тестване на цялата система и приемателно тестване. При критичните системи се изисква и т.нар. assurance case [45], който представлява организирана и явна дискусия за коректност и включва преглед на техническите изисквания, проектирането на системата, анализ въз основа на модели, както и тестване.

В [46] процесът на разработване на надеждни системи, наречен проектиране за надеждност (Design for Reliability - DfR), е представен с шест инженерни дейности:

идентифициране, проектиране, анализиране, верифициране, валидиране и управление. Той е в съзвучие с описания общ процес на решаване на проблем (*Фигура 1-3*).

Описаният процес на определяне на техниката за репликиране, който отчита много аспекти (*Фигура 1-3*), може да бъде използван като входни данни на схемата на жизнения цикъл на разработване на системи или да бъде вграден в него, като по този начин специфицира изискванията за гарантоспособност.

1.4 Излишък при гарантоспособните разпределени системи за работа в реално време

Дефиницията на понятието „излишък“, която ще използваме, е следната:

Излишъкът е функционалност или компонент на една компютърна система, който добавя ресурси за изпълнение на коректната ѝ услуга.

Както беше посочено, той не е нужен, за да може системата да изпълнява своята нормална работа, но е необходим в случай на възникване на неизправност, за да гарантира доставянето на системната услуга при тези извънредни обстоятелства. Това е метод за внедряване на отказоустойчивост при гарантоспособни компютърни системи и на свой ред се реализира чрез техники за репликиране.

Излишъкът може да бъде структурен, времеви (по време) или функционален [23], [33], [34], [37], [38]. *Структурният излишък* представлява въвеждане на допълнителни елементи (хардуерни или софтуерни) към системната архитектура, които да поемат функцията на отказалите. Той може да има различни стилове и степени на репликиране. *Стиловете на репликиране* определят дали репликирането е активно, пасивно или комбинация от двете [47], [48]. Активният структурен излишък се осъществява чрез „горещи“ резервни компоненти - модули, които работят едновременно с основния, наречен първичен, модул, като изпълняват същата задача и сравняват резултатите си с него [37], [38]. Пасивният структурен излишък използва резервни модули на първичния модул. Един от резервните модули става активен, когато първичният модул откаже [37], [38]. *Степента на репликиране* [47], [48] определя броя на репликите на компонент, т.е. броя на модулите, от които е изграден.

Времевият излишък се реализира чрез добавяне на време, най-често чрез повторение. Например, дадена управляваща програма може да бъде изпълнена два пъти и резултатите от двете изпълнения да бъдат сравнени. В случай на открита неизправност компонентът може да остане „мълчалив“ (да не изведе резултат) и да опита да се възстанови. Друг начин за прилагане на времеви излишък е да бъде изпратено отново погрешно съобщение, след като бъде открита грешка в него. По този начин се отстранява ефектът на случайните

неизправности, тъй като се допуска, че при повторното изпращане случайната неизправност ще е престанала да действа и съобщението ще бъде правилно. Времевият излишък трябва да се прилага внимателно в системите за реално време, за да се съхрани тяхната коректност в областта на реалното време.

Функционалният излишък означава репликирането на някои функции с различни средства, например разнообразяване на софтуера, приемащи тестове и др. В този случай репликите не са идентични, а техните функции са подобни.

Информационният излишък се използва в теорията на кодирането и се прилага в компютърните системи, но е извън обхвата на този труд.

1.4.1 Стил на репликиране при структурен излишък

Стилът на репликиране определя начина, по който репликираните компоненти изпълняват своята работа. Отказоустойчивите компоненти имат репликирани модули и само един от тях, първичният, извежда изходния резултат. Останалите реплики са вторични. В зависимост от стила на репликиране излишъкът може да бъде пасивен или активен.

Резервиране

Резервирането е пасивна форма на излишък [23], [33], [35]. Само първичният модул изпълнява приложението, а останалите реплики са негов резерв. Резервните модули се добавят към първичния, за да поемат функциите му в случай на негов отказ. При хардуера това се постига чрез *резерв в готовност* (standby sparing). Резервният модул е идентичен на активния, но не участва в системната работа, докато първичният модул не откаже. В този случай той става активен, а отказалият модул може да бъде диагностициран и в последствие ремонтиран. Резервирането може да се прилага по два начина в зависимост от това как състоянието на основния модул се прехвърля на резервния. При „студеното“ резервиране резервният модул може да поеме функциите на основния само когато той откаже и след това извлича състоянието му от файл, съхраняван в споделена памет. При „топлото“ резервиране резервните модули стоят в режим на готовност и периодично получават данни за състоянието на основния модул.

Репликиране

Репликирането представлява размножаване на идентични копия на даден компонент, като това включва както структурата му, така и функциите му, с цел откриване на грешки в работата на основния компонент. На практика това означава, че идентични модули изпълняват едни и същи функции върху едни и същи входни данни и сравняват резултатите си [23], [33], [35], [37], [38]. Репликите могат да работят като активни или пасивни резерви (вж.

Резервиране). Активните резервни модули работят едновременно, но само първичният модул извежда резултата към обекта на управление.

Техниките за репликиране могат да се внедряват в различни части на архитектурата на гарантоспособната система. Те могат да бъдат реализирани на хардуерно или софтуерно ниво, в комуникациите и във времезависими елементи на проектираната система.

Пасивният и активният излишък могат да се прилагат заедно, като образуват много различни комбинации.

1.4.2 Степен на репликиране при структурен излишък

Степента на репликиране определя броя на модулите в даден компонент. В зависимост от важността на компонента за системната работа той може да има един или повече репликирани модули или въобще да няма излишък. Степента на репликиране зависи и от изискванията за отказоустойчивост на компонентите.

При хардуерните реализации репликирането приема формата на *N-модулен излишък* (N-modular redundancy - NMR) [23], [34], [35], [37], [38]. Той най-често се прилага във вид на двоен модулен излишък и троен модулен излишък. При *двоен модулен излишък* (ДМИ) (dual modular redundancy - DMR) двете реплики сравняват своите резултати и, в случай на разминаване, компонентът не извежда резултат, като остава мълчалив при отказ. Обикновено модулите имат допълнителни механизми за отказоустойчивост, наречени *блокове за самопроверка*, за да решат кой модул е отказал. При *троен модулен излишък* (ТМИ) (triple modular redundancy - TMR) има три активни модула и гласуващ блок (voter). Активните модули работят едновременно, а гласуващият блок определя мажоритарния резултат, който бива изведен към обекта на управление. Гласуването на резултатите позволява да бъде маскирана една неизправност. В зависимост от режима на неизправност компонентът може да продължи да работи след отказ на един модул с два и дори един изправен модул, оборудван със средства за самопроверка.

Репликирането на софтуера се реализира като блокове за възстановяване и N-версионно програмиране. При подхода с *блокове за възстановяване* (recovery blocks - RB) [49], [50] се правят две алтернативни програми (наречени алтернативи) от обща спецификация на услугата и се въвежда приемащ тест, който решава дали резултатът е правилен. Приемащият тест се прилага последователно върху резултатите на двете алтернативи. Ако резултатите на първичната алтернатива не преминат приемащия тест, се изпълнява втората алтернатива. Подходът с блокове за възстановяване съответства на резервирането в готовност при хардуера.

При *N-версионното програмиране* (N-version programming - NVP) [49], [51], [52] съществуват N ($N \geq 2$) варианта на софтуера, които се изпълняват едновременно и резултатите им се сравняват. Вариантите са софтуерни програми, които са написани от различни екипи от програмисти и по възможност използват различни алгоритми. Предполага се, че това спомага да бъдат избегнати общите грешки, които програмистите са склонни да правят. Резултатите на софтуерните версии се гласуват и се извежда мажоритарният резултат. Хардуерният еквивалент на N-версионното програмиране е N-модулният излишък.

N-самопроверяващото програмиране (N Self-Checking Programming - NSCP) [49] е друг подход за въвеждане на излишък в софтуера. При него се изпълняват N самопроверяващи се софтуерни компонента, като един от тях се смята за действащ, а останалите самопроверяващи се компоненти са негови „горещи“ резерви. При отказ на действащия компонент действието се превключва към някой от резервните самопроверяващи се компоненти.

1.4.3 Времеви излишък

Времевият излишък изисква да бъде заделено допълнително време за изпълнението на задачи [23], [37], [53], [54]. Например, управляващата програма се изпълнява два пъти последователно и се сравняват резултатите [25], за да се преодолее ефектът на евентуална случайна неизправност. В случай на разлика компонентът остава мълчалив. Друга форма на времеви излишък при гарантоспособните разпределени системи е да се изпраща дадено съобщение повторно, ако бъде докладвана неизправност преди първото предаване [25], [37], [38]. Неизправността в съобщението се открива с вградени механизми, като CRC код, сравнение с резултатите на другите компоненти и т.н. По този начин се толерират случайни неизправности. Времевият излишък създава по-малък разход в сравнение със структурния излишък, но може да повлияе на производителността на системата и затова трябва да бъде подчинен на ограниченията за реално време.

1.4.4 Функционален излишък

Функционалният излишък се внедрява в софтуера. В [33] той се дефинира като квалифициране на поведението на системата по отношение на нейните входни/изходни взаимоотношения. Според дефиницията, представена в [33], дадена система има *функционален излишък* в три случая: (i) ако някои теоретично възможни входни стойности не са приложими според системните спецификации; (ii) ако някои теоретично възможни изходни стойности не са произведени според системните спецификации; и (iii) ако някои теоретично възможни входни/изходни стойности никога не възникват според системните спецификации. Функционалният излишък е полезен при откриването на грешки.

1.5 Въвеждане на излишък

Разпределените форми на излишък могат да се прилагат в съчетание и/или на различни нива в системата [55]. При гарантоспособните разпределени системи компонентите са физически репликирани (хардуерен излишък), софтуерните модули са разположени на различни процесори (софтуерен излишък), комуникационните канали биха могли също да бъдат репликирани (хардуерен излишък), а често се прилага и времеви излишък при изпълнението на задачите.

Някои системи използват вид йерархия на излишъка. Те дефинират нивата на важност на своите елементи и въвеждат комуникационна процедура между тези нива. В [56], например, се определят нива на доверие (integrity levels) и нива на критичност (criticality levels).

Гарантоспособността в разпределените системи за реално време, които работят в критични за безопасността приложения, е станала част от тяхното проектиране. Всички параметри и системни компоненти, които са важни за отказоустойчивото функциониране на системата, се включват в жизнения цикъл на разработване на системата. Описателен многослоен модел на проектиране на разпределени системи със структурен излишък е илюстриран на *Фигура 1-4*.

Фигура 1-4. Синтез на подход на гарантоспособни разпределени системи със структурен излишък

Въвеждането на структурен излишък включва определянето на критичните елементи, на важните системни параметри и на нивата на критичност (ако такива се предвиждат в системата). Компонентите на разпределената система управляват различни параметри на обекта на управление с различна значимост за отказоустойчивото функциониране на системата. На етапа на определяне на изискванията в жизнения цикъл на разработване на

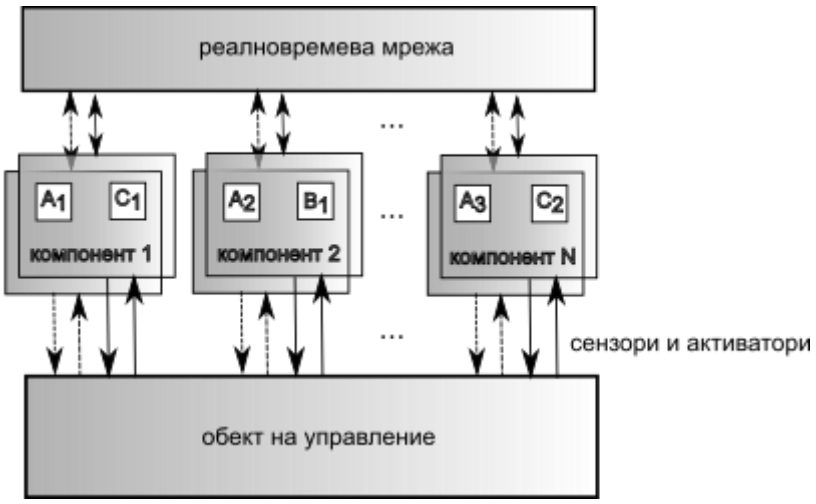
системата трябва да бъдат определени контролираните параметри и да бъдат откроени нивата на критичност. Компонентите, които управляват важните параметри, получават високо ниво на критичност и трябва да бъдат отказоустойчиви. На етапа на проектиране на системата се определят степента и стила на репликиране. Степента на репликиране определя дали ще бъдат приложени компоненти с единичен модулен излишък (ЕМИ), двоен модулен излишък (ДМИ) или троен модулен излишък (ТМИ). Избраният стил на репликиране определя дали ще бъде използвано активно или пасивно репликиране. Модулите в системните компоненти могат да бъдат равномерно разпределени, т.е. всички компоненти могат да имат еднакъв брой модули, или могат да използват смесен излишък. На етапа на действителното изграждане в жизнения цикъл на разработване на системата се внедрява определената системна архитектура, например GUARDS [56], [57], многостранна гарантоспособност (versatile dependability [47], [48]) или DECOS [58], [59], както е показано на *Фигура 1-4*.

Излишъкът рядко се прилага в чист вид при гарантоспособните разпределени системи. Винаги се прилага комбинация от стилове на репликиране на различни системни нива. При хардуера системните компоненти могат да бъдат репликирани по идентичен начин или в зависимост от тяхната критичност. Софтуерните компоненти също могат да имат различен брой копия на различни компоненти/процесори, за да се постигне по-добро използване на системните ресурси. При изготвянето на разписание на задачите в гарантоспособните разпределени системи се прилагат подходи за смесена критичност (mixed criticality), които отговарят на изискванията за отказоустойчивост и реално време [60], [61].

1.6 Внедряване на различни степени на репликиране

1.6.1 Еднакъв излишък за всички компоненти

Най-прекият начин да се приложи излишък е да бъдат репликирани активните компоненти и да бъдат сравнявани техните резултати. При разпределените системи възлите могат да бъдат изградени от два или три идентични модула, които да изпълняват една и съща задача (*Фигура 1-5*). Резултатите на репликираните модули се сравняват (както е в [25], [28], [49], [58]) или се гласуват в случай на използване на троен модулен излишък (както е в [17], [62]). Ако няма разминаване, предполагаемо верният резултат се извежда към обекта на управление. При разлика не се извежда резултат, а компонентът се поставя в безопасно състояние според конвенциите на системата.



Фигура 1-5. Гарантоспособна разпределена система за работа в реално време

Хардуерните компоненти на системата имат еднаква степен на репликиране, но софтуерните компоненти, изпълняваните задачи, могат да имат различен излишък. Например, на *Фигура 1-5* са изобразени три задачи – A, B и C; задача A има три копия, задача B има едно, а задача C има две. Копията могат да бъдат разположени в различни системни компоненти, като по този начин се постига изолиране на грешките.

Двете исторически установили се направления при гарантоспособните разпределени системи са да бъдат изграждани отворени архитектури като Delta-4 [26], [63], [64], [65] или строго детерминирани архитектури като MARS [25]. Проектът Delta-4 определя компютърна архитектура за работа в реално време, която използва репликирани софтуерни компоненти, разположени на различни хост компютри. Хост компютрите са отказоустойчиви (хардуерен излишък). Определено е хомогенно множество от техники за отказоустойчивост, което позволява изграждането на различни системи и приложения с различни изисквания за гарантоспособност и производителност, които да бъдат конфигурирани без повторно проектиране на софтуерните компоненти. В DELTA-4 се използват готови компоненти, т.нар. Commercial Off-The-Shelf (COTS) компоненти. Възлите са разделени в две различни подсистеми: хост система, която е COTS компонент, и контролер за прикрепване към мрежата (network attachment controller – NAC), който е компонент с режим на отказ от вида „мълчание при отказ” и използва специализиран самопроверяващ се хардуер. Необходимостта да се насочат системите към по-твърди времеви изисквания води до специфицирането на архитектура за разширена производителност (extended performance architecture – XPA). XPA системите се състоят от множество от разпределени хомогенни възли, свързани чрез мрежа от вида „разпределен интерфейс за данни с оптично влакно”, където хостът също се смята за мълчалив при отказ. За разлика от NAC обаче поведението на мълчание при отказ на хостовете

се постига чрез използването на „меки“ възли с мълчание при отказ, където поведението на мълчание при отказ се постига чрез софтуерно управление на репликираните процесори.

В MARS [25] целта е да се изгради предсказуемо гарантоспособна разпределена система, основана на твърди ограничения за реално време. Системата е проектирана така, че да запазва своята функционалност дори и при условия на върхово натоварване. Архитектурата се състои от един или повече клъстера, които са разпределени системи, съставени от единични компютри (на една платка), наречени компоненти, свързани чрез мрежа за времеделене и множествен достъп (time division multiple access – TDMA). Всички компоненти поддържат глобална времева база, която им позволява да синхронизират действията си и да използват подход със задействане по време. В MARS разписанията както на възлите, така и на мрежата се определят off-line и се съхраняват в статична таблица. Гарантирано е, че компонентите са мълчаливи при отказ чрез използването на хардуер за самопроверка, който работи с двоен активен излишък и използването на две мрежи за реално време с излишък, където съобщенията се изпращат с дублиране. Активният излишък е физически реализиран като двоен модулен излишък, т.е. два компонента изпълняват едни и същи задачи. Компонентите в MARS имат механизми за самопроверка, които гарантират тяхното поведение на спиране при отказ.

Много гарантоспособни разпределени системи са изградени с идеята да предлагат повторно използване на компонентите, по-ниска цена на проектиране и да предоставят гъвкавост по отношение на изискванията на приложението. Системата DEAR-COTS [66], [67] е подход за намаляване на цената на проектиране, като внедрява готови (COTS) компоненти в гарантоспособните разпределени системи. Системата е конструирана от разпределени възли, които изпълняват приложения с твърди ограничения по време. Подсистемата с твърди ограничения по време (hard real-time subsystem - HRTS) на DEAR-COTS е основана на софтуерното интегриране на хардуерни COTS компоненти. Едно приложение за реално време с твърди ограничения по време е разделено на няколко задачи, които се изпълняват на различни възли на HRTS. Всеки възел има собствено COTS ядро и хардуер и подпомагащият софтуер на HRTS предоставя разпределението на приложението и управлението на излишъка. В DEAR-COTS се използва активно репликиране на множества от репликирани задачи.

Използването на COTS компоненти е залегнало и в архитектурата на GUARDS [57]. Генеричната отказоустойчива компютърна архитектура е дефинирана в три направления на ограничаване на неизправностите: нива на доверие (integrity levels), платна (lanes) и канали (channels). Отказоустойчивостта и управлението на доверието са софтуерно приложени, с минимално количество специализиран хардуер. Нивата на доверие са предназначени да

предотвратяват преминаването на проектни неизправности извън областите на ограничаване. Целта е да се защитят критичните компоненти от грешки, дължащи се на проектни неизправности в по-малко критичните компоненти. Нивото на доверие отразява степента, до която даден приложен обект е надежден (т.е. до която може да му се има доверие). Архитектурата GUARDS използва множество процесори или платна за откриване и диагностициране на физически неизправности в даден канал. Отказоустойчивостта се постига посредством активно репликиране на приложните задачи върху набор от канали.

Подходът за отказоустойчивост чрез софтуерни средства е фундаментален въпрос за COTS-базираните надеждни архитектури. Тъй като не съществува специализиран хардуер със свойства за самопроверка, софтуерът е този, който трябва да управлява репликирането и отказоустойчивостта. При този подход отказоустойчивото обслужване се прилага чрез координиране на група от софтуерни компоненти, репликирани на различни възли. Идеята е да се управлява групата от софтуерни компоненти така, че да се маскират откази в някои от тях. Координацията между репликите създава илюзия на други софтуерни компоненти, че групата е единичен (без откази) софтуерен компонент [65].

Архитектурата със задействане по време (Time-Triggered Architecture - TTA) [53], [62], [68], [69] разширява подхода на MARS, за да даде възможност за изграждане на гарантоспособни разпределени системи за реално време. Един компонент на TTA е блок на ограничаване на неизправности. Отказоустойчивите блокове са изградени от репликирани компоненти, за да се маскират произволни неизправности в компонентите. TTA компонентите взаимодействат със средата само посредством обмен на съобщения. Те притежават свързващи интерфейси, основани на съобщения, които са независими от технологията на внедряване на компонентите. Архитектурата TTA предоставя глобално синхронизирана времева база [70]. Обменът на съобщения между модулите се извършва върху два репликирани канала. Най-малката заменима единица (smallest replaceable unit - SRU) [17], [53] се състои от хост подсистема и комуникационна подсистема. Хост подсистемата изпълнява приложния софтуер, а комуникационната подсистема отговаря за изпълнението на комуникационния протокол. Една или повече SRU могат да формират отказоустойчив блок.

Dacapo [27] е отказоустойчива разпределена система за реално време, разработена за автомобилни приложения. Тя е от вида периодични вградени системи, които работят циклично. Състои се от малък брой компютърни възли, които комуникират по отказоустойчива мрежа. Възлите са пространствено разпределени в местата по автомобила, където се събират сензорите и активаторите. Всеки възел основно съдържа вграден контролер и комуникационен блок. Те от своя страна се състоят от две множества от функционални

блокове с поведение от типа „мълчание при отказ”. Това дава възможност да се реконфигурира възелът по различни начини. В най-лошия случай даден възел е „работещ при отказ”/”мълчалив при отказ” за всеки две последователни постоянни неизправности, т.е. е напълно работещ при първата постоянна неизправност и „мълчи“, когато настъпи втората неизправност. Комуникационната мрежа съдържа две последователни магистрали. Системата с глобална магистрала е „работеща при отказ”/”работеща при отказ”, което означава, че работи дори и при възникване на две постоянни неизправности.

Съществува присъщ излишък в управляващите функции на превозното средство. Дори ако откаже някоя локална управляваща функция, е възможно превозното средство да бъде управлявано с намалена производителност, при положение че отказалата функция се държи по предвидим начин. Системата Dacapo позволява изпълнението на критични за безопасността задачи, подредени в предварително изготвено разписание, и на събитийни задачи (т.е. такива, които не са критични за безопасността), подредени по време на изпълнение.

Правенето на предварително разписание на задачите се прилага често при отказоустойчивите разпределени системи за реално време. Мотивацията за това основно е, че цикличната система с предварително разписание, освен че се проектира просто, дава предимства, които са изключително ценни за целевите приложения:

- Детерминизмът по отношение на времето позволява гарантираното спазване на реалновременните граници.
- Ефектът от грешка при предаване на съобщение или пропускането на реалновременна граница поради случайна неизправност обикновено е ограничен във времето. Това се дължи на факта, че всички променливи, които е необходимо да се знаят извън даден компонент, се обновяват периодично.
- Предварителното познаване на комуникационното разписание позволява бързото откриване на много възможни грешки.

Освен това, цикличната работа се съгласува добре с периодичната работа на системите за управление.

1.6.2 Различен излишък за компонентите

Има системи, които използват различна степен на репликиране за своите компоненти. Някои от контролираните системни параметри или някои системни компоненти са по-важни за системната работа от други. Важността на такива компоненти се определя от последствията от техен отказ. Ако системата не може да си позволи да загуби някои от своите функции, защото това би довело до катастрофален отказ, компонентът се смята за критичен.

В GUARDS [57] съществуват нива на доверие и нива на критичност. Нивата на доверие се дефинират според степента, до която може да се има доверие на даден системен компонент – колкото по-доверен е компонентът, толкова по-високо ниво на доверие има [56]. Степента на доверие, възложена на компонента, зависи от неговата критичност. За критични компоненти се смятат тези, чийто отказ води до тежки последствия. Такива компоненти имат по-висока степен на доверие. Нивата на доверие са част от политиката на доверие, която предпазва от протичане на информация от ниско към високо ниво на доверие. Нивата на критичност са свързани с нивата на доверие, но се отнасят за различни въпроси. Нивата на доверие, посредством политиката на доверие, дефинират допустимите потоци от данни между нивата и оползотворяването на ресурсите от страна на компонентите от гледна точка на потенциалните последствия от отказите на компоненти на всяко ниво [57].

Моделът на репликиране в DEAR-COTS [28], [66] използва активен излишък на софтуерните компоненти и позволява определянето на степента на репликиране на специфични части от приложението за реално време в съответствие с надеждността на компонентите и желаното ниво на надеждност за приложението. Системата не е насочена към критични за безопасността приложения, тъй като те изискват по-високи нива на гарантоспособност и по-ограничено множество допускания за отказ. Архитектурата DEAR-COTS е насочена към разпределени компютърно управляеми системи, които работят в реално време и може да използва както равномерно, така и смесено разпределение на излишъка.

Дефинира се *n*-репликиран компонент, който определя степента на репликиране. Тя се дефинира за специфични части от реалновременното приложение в зависимост от надеждността на неговите компоненти и нивото на надеждност за цялото приложение. Възможно е да се репликират само определени софтуерни компоненти с цел да се намали броят на задачите и обменяните съобщения. Частите, които не са репликирани, трябва да притежават по-висока собствена надеждност. В архитектурата е определено ниво за управление на репликите (Replica Manager layer), което предоставя на приложенията с твърди времеви ограничения необходимите ресурси за комуникация между разпределените задачи и репликираните компоненти. Трябва да бъде гарантирано детерминистично изпълнение на репликите.

Проектът DECOS [58], [59] предлага интегрирана разпределена архитектура, която да поддържа системи със смесена критичност. Системите със смесена критичност се състоят от разпределени части на приложението с различни нива на критичност, изпълняващи се върху един и същ физически хардуер. Системата DECOS е основана на принципите на „силно капсулиране на грешките, отказоустойчивост посредством репликиране и излишък и

разделяне на критичните за безопасността функции от некритичните“ [58]. Функционалното разпределение в следствие на тези принципи води до структура с определен брой разпределени приложни подсистеми (Distributed Application Subsystems - DASs) и дейности (jobs). Системната услуга се изпълнява от клъстери, които са множества от разпределени компютри на възлите, взаимно свързани от мрежа със задействане по време (time-triggered network). Компютрите на възлите предоставят защитена среда за изпълнение на дейностите посредством капсулирани дялове. Според федерирания подход части от приложението с различни нива на критичност се прилагат като самостоятелни блокове със собствени процесори и В/И системи. В DECOS изискването за липса на взаимно влияние между капсулираните дялове позволява на възлите да имат много дейности, които принадлежат на различни DASs, с различни нива на критичност.

За постигане на гъвкавост на разпределените системи по отношение на изискванията на работната среда за гарантоспособност редица системи прилагат софтуерни решения на ниво мидълуер (middleware[4]). Тези подходи отчитат особеностите на прилагане на COTS компоненти и надеждностните изисквания на приложенията. COTS компонентите са готови продукти, които спестяват средства за разработване и съкращават времето за проектиране на гарантоспособни разпределени системи. Те обаче имат неизвестна надеждност и неясна вътрешна структура. Това се компенсира чрез допълнителни механизми за отказоустойчивост. За да се постигне адаптиране на отказоустойчивата система към изискванията за гарантоспособност на средата/обекта на управление, се прилагат софтуерни решения, най-често CORBA[5] мидълуер.

Разпределените системи с групова комуникация, изискват специализиран и често сложен софтуерен слой и/или допълнителен хардуер, за да предоставят групова комуникация и добро покритие за осигуряване на режим „мълчание при отказ“. Повечето предлагат и среда за разработване на разпределени приложения и постигат отказоустойчивост посредством репликиране.

Архитектурата AQuA [71] е предназначена за гарантоспособни разпределени системи, изградени от готови компоненти. Тя отразява необходимостта разпределената система да се адаптира към промени при изпълнението на приложението, което управлява. Това се постига

[4] Терминът middleware се използва в дисертационния труд за обозначаване на софтуера, който стои между операционната система и приложенията и улеснява комуникацията и управлението на данните при разпределени приложения.

[5] CORBA е съкращение на Common Object Request Broker Architecture™. Това е отворена независима от производителя архитектура и инфраструктура на OMG®, която използват компютърните приложения, за да работят заедно в мрежа. За повече информация: https://www.corba.org/faq.htm.

чрез софтуер на ниво мидълуер и механизми, които позволяват надеждностните изисквания на разпределените приложения да бъдат отразени от системата. Архитектурата AQuA дава възможност на приложенията да заявяват желано ниво на готовност, като използват рамка от обекти на качество (Quality Objects - QuO) и мениджър на гарантоспособност, който да конфигурира системата според заявката и според промените в системните ресурси в резултат на неизправност. Посредством QuO в реално време архитектурата AQuA обработва и извиква заявки за готовност, мениджърът на гарантоспособност конфигурира системата в отговор на неизправности и заявки за готовност, а протоколен стек осигурява услуги за групова комуникация. Използва се активно и пасивно репликиране, а стилът на репликиране се избира въз основа на изискванията за производителност и гарантоспособност на конкретното приложение.

Chameleon [72] е адаптивна софтуерна инфраструктура, която позволява да бъдат едновременно поддържани различни нива на изисквания за готовност в мрежова среда. Инфраструктурата предоставя гарантоспособност посредством използването на специални обекти, наречени ARMORs (Adaptive, Reconfigurable, and Mobile Objects for Reliability), които управляват всички операции в средата на Chameleon. Идеята на Chameleon е да работи върху произволна мрежа от ненадеждни възли и да предостави механизми за постигане на отказоустойчивост, съобразени с конкретно приложение. Тази инфраструктура може да работи върху мрежа, състояща се от готов хардуер (процесорни елементи и мрежа) и софтуер (операционна система). Ядрото на Chameleon е ARMOR архитектура, която може да се реконфигурира, и се състои от библиотека от повторно изпълними техники за отказоустойчивост. Ядрото представлява основата за създаване на различни реализации на ARMOR и конфигуриране на системата за изпълнение на конкретни стратегии за отказоустойчиво изпълнение. Механизмите за отказоустойчивост, които използва Chameleon, са: репликиране на приложение, гласуване на резултатите от изчислението, откриване на нетипично прекратяване, рестарт, поставяне на контролни точки и връщане назад. Инфраструктурата е приложима при среди, използващи обработка на изображения, и в приложения за разпределени бази данни.

Системите AQuA и Chameleon не са предназначени за работа в реално време. Те предлагат гъвкавост при въвеждането на гарантоспособност в разпределени системи, но нямат задача да отговарят на ограничения във времето. Системата MEAD (Middleware for Embedded Adaptive Dependability) [48] предлага съчетаване на противоречивите изисквания за отказоустойчивост и реално време при прилагане на гарантоспособност на ниво мидълуер. MEAD е инфраструктура, която предлага прозрачна и регулируема отказоустойчивост в

реално време, проактивна гарантоспособност, системно адаптиране към пълен отказ, неизправности в комуникацията и във времето с отчитане на наличните ресурси и скалируемо и бързо откриване на неизправности и възстановяване от тях. Регулируемата отказоустойчивост се постига чрез т.нар. подход на многостранната гарантоспособност (versatile dependability) [47], [48]. Този подход дава възможност за изграждане на гарантоспособни софтуерни архитектури, като се отчитат три важни аспекта – отказоустойчивост, производителност и ресурси. Той предоставя набор от инструменти, наречени „копчета“, за настройване на баланса между тези аспекти. Рамката на многостранната гарантоспособност разграничава „копчета“ (настройки) от високо и от ниско ниво. Вътрешните свойства на отказоустойчивост, като стил на репликиране, минимален брой реплики, интервали на поставяне на контролни точки, интервали на наблюдение за неизправности и т.н., са „копчета“ от ниско ниво. „Копчетата“ от високо ниво съответстват на външните свойства на системата, например скалируемост и готовност. Рамката на многостранната гарантоспособност се основава на отказоустойчиви спецификации на мидълуера. Предвидена е за разпределени асинхронни системи. Специализиран софтуерен модул, наречен репликатор, управлява групи реплики от вида клиент и сървър. Репликирането се прилага на ниво процес, тъй като процесът може да съдържа няколко обекта, които трябва да бъдат възстановявани в случай на отказ на процеса. Репликаторът поддържа активен и пасивен стил на репликиране.

Повечето гарантоспособни разпределени системи за реално време следват архитектурния стил, изобразен на *Фигура 1-5*. Те използват еднакво репликирани физически компоненти и смесен излишък на софтуерните компоненти. Съществуват възможности за разработване на системи, които се адаптират към конкретни приложения чрез разпределение на хардуерния структурен излишък.

1.7 Изводи и резултати

В *Глава 1* са представени основните понятия от предметната област на дисертацията – гарантоспособни разпределени системи за работа в реално време. Откроена е тяхната структура по отношение на гарантоспособността и реалното време.

Управлението на структурния излишък е разгледано през призмата на системното проектиране. Създаден е концептуален модел на подход за вземане на решения при осигуряване на гарантоспособност. Този модел се вписва в жизнения цикъл на проектиране на системи и отговаря на изискването гарантоспособността да бъде заложена в системните спецификации.

Направен е обзор на методите и техниките за въвеждане на излишък в отказоустойчиви системи и е синтезирана класификация на гарантоспособни разпределени системи със структурен излишък, която показва вграждането на изискванията за гарантоспособност в етапите на проектиране на системата.

Представен е кратък обзор на известните гарантоспособни разпределени системи от гледна точка на управлението на структурния излишък. Те са представени според начина, по който внедряват различни степени на репликиране. Двете тенденции са да се използва еднакъв излишък за всички системни компоненти или компонентите да имат различен излишък. Разгледани са техните предимства и недостатъци. Очертана е възможност да се изследва различно разпределение на хардуерния структурен излишък.

Изложението в *Глава 1* отговаря на изпълнението на задача 1 на дисертацията.

Постигнатите научни и научно-приложни резултати са:

1. Създаден е концептуален модел на подход за вземане на решения при осигуряване на гарантоспособност;
2. Предложен е синтез на класификация на гарантоспособни разпределени системи със структурен излишък.

Част от резултатите по тази глава са представени в публикацията

1. Djambazova, E., & Andreev, R. (2023). Redundancy management in dependable distributed real-time systems. *Problems of Engineering Cybernetics and Robotics*. (под печат)

Глава 2 Моделиране на отказоустойчивата разпределена система с настройваема надеждност

Идеята за разпределение на излишъка се прилага под различна форма в някои системи – многостранна гарантоспособност [47], [48], смесена критичност в DECOS [58], [59], нива на критичност в GUARDS [57] и др. Понятието „смесена критичност“ при разпределените системи е свързано с определянето на максимално време за изпълнение (WCET) при изготвянето на разписанието за работа на компонентите/задачите. Прилагането на смесената критичност се доближава до дефинираното от автора понятие „настройваема надеждност“ дотолкова, доколкото става въпрос за определяне на системни компоненти с различна степен на критичност за системата. При настройваемата надеждност обаче критичността на компонентите има значение за разпределението на излишъка и за индивидуалната надеждност на всеки компонент и приноса му към общата надеждност. Подходът на многостранна гарантоспособност [47] също дава възможност за настройване на отказоустойчивостта чрез промени на структурния излишък. Целта е да се внесе гъвкавост на системата по отношение на променящи се условия на средата по време на работа. Реализацията на подхода е на ниво мидълуер, където със софтуерни ресурси се въвеждат механизми на настройваемост по отношение на отказоустойчивост, производителност и ресурси, но не се определят нива на критичност. При системата с настройваема надеждност, представена в дисертацията, степента на репликиране на хардуерните компоненти се определя в зависимост от тяхната критичност за приложението, а разпределението на излишъка в компонентите отразява изискването за обща надеждност на системата.

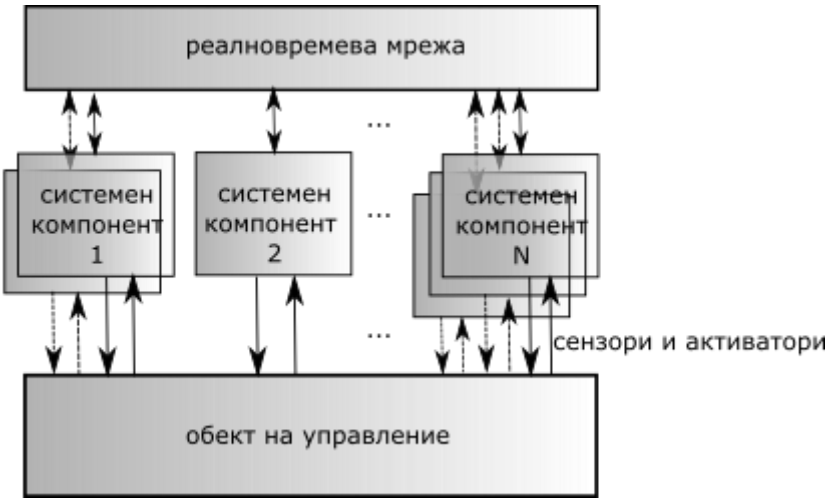
Предложената в дисертацията архитектура на система с настройваема надеждност е основана върху понятието за настройваемост.

Настройваемост е свойството на гарантоспособната разпределена система за реално време да разпределя структурния излишък според изискванията за надеждност на приложението.

Отказоустойчивата разпределена система с настройваема надеждност [73], [74], [75], [76] прилага различни степени на репликиране на хардуерните компоненти (*Фигура 2-1*). Необходимостта системата да има компоненти с различна степен на репликиране е свързана с тяхната критичност. За разлика от подходите, прилагащи смесена критичност на частите на приложния софтуер, които се изпълняват на еднакво/равномерно репликиран хардуер, подходът с настройваема надеждност предлага степента на репликиране на хардуерните

компоненти да се определя според тяхната критичност на етапа на проектиране и те да работят със смесен излишък. В дисертацията е направена количествена оценка на възможностите за реализиране на гарантоспособни разпределени системи с предложения подход и те са сравнени със системи без разпределение на структурния излишък. Сравнението е направено с помощта на модели, решени посредством симулационно моделиране, и резултатите (представени в *Глава 3* и [76]) показват, че съществуват разпределения на излишъка, които постигат общата системна надеждност, изисквана от приложението.

Отказоустойчивата разпределена система с настройваема надеждност (*Фигура 2-1*) е изградена от компоненти, ограничаващи неизправностите. Тяхната отказоустойчивост се гарантира чрез репликиране и самопроверка. Компонентите могат да имат различна степен на репликиране, т.е. изградени са от различен брой модули в зависимост от критичността си. Всеки модул има блок за самопроверка, чрез който се откриват грешки и се осигурява спиране/мълчание при отказ. Системата работи при твърди времеви ограничения. Този подход позволява предвидимост на поведението на системата, като времената за изпълнение на системните задачи са гарантирани. Това намалява гъвкавостта на системата при реконфигуриране, но улеснява проектирането, комуникацията и надеждната работа на системата при максимално натоварване. Чрез промяна на структурния излишък на компонентите на етапа на проектиране се цели промяна на надеждността на системата според изискванията на приложението. Моделират се хардуерни неизправности и се цели постигане на хардуерна надеждност.



Фигура 2-1. Гарантоспособна разпределена система за реално време с настройваема надеждност

Избраният в дисертацията метод за изследване на предложените модел и архитектура на отказоустойчивата разпределена система с настройваема надеждност е симулационно

моделиране. Той е основан на Марковски вериги с непрекъснато време и дискретни състояния. В т. 2.1 е направен кратък преглед на най-използваните методи за моделиране на гарантоспособни системи. Моделът на отказоустойчивата система с настройваема надеждност и допусканията при нейното моделиране са представени в т. 2.2, където е обоснован и изборът на изследователски метод за моделиране на системата. Изследваните надеждностни характеристики, представени в *Глава 1*, са отделени в *Приложение А*, където са описани техните математически дефиниции и представяне.

2.1 Анализ на начините за моделиране на гарантоспособни системи

Моделирането е често използван похват за изследване на системи, преди те да бъдат реално проектирани и внедрени. Това дава възможност да се проверят различни хипотези и да се намери подходящият модел, въз основа на който да се търсят конкретни инженерни решения. Моделирането позволява сравнение между различни варианти за изграждане на системата или на части от нея по ефективен начин по отношение на цената, тъй като възможните инженерни реализации не винаги могат да се оценят количествено при сложни системи. При отказоустойчивите системи много фактори благоприятстват използването на модели. Изискването за гарантоспособност предполага въвеждането на различни средства за постигането ѝ (вж. *Глава 1*). Оценката на атрибутите на гарантоспособността може да се извърши аналитично, използвайки изискваните от приложението параметри. При системите за реално време е съществено да се спазват времевите граници при взаимодействието на компонентите: при синхронизацията, изготвянето на разписания за работата на компонентите и техните репликирани модули, протокола за обмен на съобщения и пр. Друг съществен фактор, изискващ моделиране и оценка, са неизправностите, на които е подложена системата. Това предполага избор на адекватен режим на неизправност и отказ, на подходящ математически модел на тяхното въздействие, на средства за математическо решаване на модела и т.н. Покритието на механизмите за откриване на грешки също може да се моделира.

Неизправностите имат различна причина, произход, времетраене. Моментът им на възникване е непредвидим. Те съпътстват техническите системи през целия им жизнен цикъл. Поради стохастичната природа на неизправностите показателите, с които се оценява тяхното въздействие върху функционирането на системата, имат вероятностен характер. Затова за описанието им се използват понятия от областта на теорията на вероятностите и математическата статистика. Дефинираните в *Глава 1* атрибути на гарантоспособността имат математическо изражение. Всички използвани в дисертацията математически дефиниции на

надеждностните характеристики на отказоустойчивата разпределена система са представени в *Приложение А*, а в хода на изложението са пояснени и цитирани онези от тях, които имат отношение към дискутирания проблем. Методите за моделиране са разнообразни. Най-често използваните от тях са представени в т. 2.1.1.

2.1.1 Методи за моделиране на гарантоспособни системи

Системното моделиране е изкуството на разработване на абстрактно представяне на действителна система с цел извличане и анализиране на нейното поведение от гледна точка на производителност и гарантоспособност при различни условия на функциониране, без позоваване на измервания върху действителна система или нейния прототип [77].

Според таксономията на техниките за моделиране на отказоустойчиви системи, представена в [54], моделите са разделени в две основни групи в зависимост от това дали моделират надеждност или готовност. Някои техники за моделиране са специфични за оценяване на надеждността, когато се моделира система без ремонт – комбинаторните техники (серийни/паралелни, M от N, несерийни/непаралелни), а други са подходящи за оценяване на готовността – модели с опашки и комбинаторни модели. Техниките за моделиране, които дават възможност да се оценява и надеждност (на ремонтируеми системи), и готовност, са разновидностите на Марковските вериги, които могат да бъдат независими от времето, времезависими и хибридни. Времезависимите Марковски вериги са групирани като модели с дискретно време, модели с непрекъснато време и модели с Монте Карло симулация.

Моделите на системата могат да бъдат анализирани или математически оценявани посредством три различни подхода [77]:

- Симулационен;
- Аналитичен;
- Хибриден (комбинация от симулационни и аналитични методи).

Моделират се само основните характеристики на системата. При симулирането тя се описва в компютърна програма, която имитира нейната динамика. При аналитичните методи се съставят и решават системи от математически уравнения, които определят системната динамика [77], [78], [79], [80]. Предимството на симулирането е, че могат да се представят подробно характеристиките на изследваната система, без да се налагат много ограничения върху модела. При аналитичните модели допусканията често се опростяват, за да могат да се решат системите от уравнения. Точността при симулирането се ограничава единствено от времето, необходимо за получаване на краен резултат. Колкото по-подробен е моделът,

толкова повече време отнема решаването му. Често се прилагат техники за паралелна и разпределена обработка, за да се ускори този процес [34]. При аналитичните модели решенията изискват по-кратко време. Възможно е и комбинираното прилагане на двата подхода, което все още не е честа практика [77].

При друга класификация [81] моделите се разглеждат като комбинаторни и модели, основани на пространство на състоянията. Комбинаторните модели представят структурата на системата чрез логически връзки между компонентите, водещи до получаване на крайно състояние – отказ на системата. Моделите, основани на пространство на състоянията, представят поведението на системата посредством достижими състояния и възможни преходи между тях.

Комбинаторните модели включват диаграми с блокове на надеждност (Reliability Block Diagrams - RBD) [77], [82], [83], [84], дървета на събитията (Event Trees - ET) [81] и дървета на неизправностите (Fault Trees - FT) [77], [85], [86]. Те са сравнително лесни за проектиране и обработване и могат да се анализират с комбинаторни методи. Техен недостатък е ограничената им моделираща способност, дължаща се на допускането за статистическа независимост на събитията.

Дървото на неизправностите (ДН) [86] е широко разпространен модел за анализ на надеждността на сложни системи, тъй като дава възможност за интуитивно представяне на режимите на системен отказ, лесно се обработва и за него има създадени софтуерни инструменти. ДН моделира как комбинации от събития на откази на компоненти могат да причинят отказ на цялата система. По същество това е нецикличен насочен граф, който се характеризира от два вида възли: събития и логически елементи (gates). Събитията се отнасят за откази на компоненти, подсистеми или на цялата система. Събитието е булева променлива, която първоначално е „лъжа“ (false) и става „истина“ (true), когато настъпи отказ. Логическите елементи са свързани посредством дъги към няколко входни събития и към едно-единствено изходно събитие. Те реализират прости логически схеми, които при определена комбинация от събития на входовете, извеждат отказ към изходното събитие. Логическите елементи могат да бъдат „логическо И“, „логическо ИЛИ“ и „k от n” (“k out of n”). Последното означава, че логическият елемент извежда „истина“, ако поне k от n входни променливи е „истина“; в противен случай извежда „лъжа“. Корен на ДН е т.нар. върхово събитие (top event – TE), което моделира отказ на системата. Дъгите на насочения граф спазват ориентацията на логическите вериги: от входните събития към логическия елемент и от логическия елемент към изходното събитие. Тъй като ДН не е цикличен граф, връзките на събитията с логическите елементи чрез

дъги не трябва да определя циклични пътища. Моментът на възникване на отказ на основно събитие (т.е. отказ на компонент) е случайна променлива с вероятностно разпределение, обикновено отрицателно експоненциално. В такъв случай параметърът на разпределението е интензивността на отказите на компонента, λ, обратно пропорционално на неговото средно време на живот. Допуска се, че основните събития са статистически независими и са крайни възли на ДН. Чрез ДН могат да се изчислят количествено системната не-надеждност, средното време до отказ и факторите на важност (критичност на компонент).

Моделите с пространство на състоянията включват Марковски вериги и мрежи на Петри [84], [87]. Те представят поведението на системата посредством достижими състояния и възможни преходи между тях. Те имат по-голяма моделираща мощност от комбинаторните модели, които не могат да обхванат характеристиките на моделираната система – например ремонт на компонент или на цялата система, зависимост на състоянията едно от друго и т.н. Недостатъкът им е, че анализът на пространството на състоянията може да изисква голям брой изчисления при експоненциално нарастване на броя на състоянията в модела. Когато анализът на моделите с пространство на състоянията стане практически невъзможен поради нарастване на броя на състоянията, характеристиките на гарантоспособността могат да бъдат получени чрез симулиране на тези модели.

Марковските вериги (Markov chains) [54], [81], [84], [88] са изградени от състояния и преходи между тях. Състоянието на системата представлява нейно описание в даден момент от време. Когато се моделира надеждност, състоянието представлява конкретна комбинация от работещи и отказали компоненти. Функционирането на системата във времето води до преминаването ѝ от едно състояние в друго, тъй като компонентите отказват или биват ремонтирани. Тези промени се изразяват чрез преходи в модела. Марковските вериги се изобразяват като графи, в които възлите са състоянията на системата, а дъгите представляват преходите между тях. Състоянието е специфична конфигурация на системата, която е валидна за определен интервал от време. Престоят в дадено състояние се определя със случайна променлива, зависеща от вероятностите на преходите. При Марковските вериги с дискретно време [84] времето се дискретизира на стъпки с единична дължина Δ. Всеки преход се характеризира с вероятността да настъпи преход, когато бъде достигната времевата стъпка. При Марковските вериги с непрекъснато време [84] преходът между състоянията може да настъпи във всеки момент от времето.

Основното допускане при Марковските вериги е, че вероятността за преход зависи единствено от текущото състояние. При Марковските вериги с непрекъснато време

продължителността на престоя в дадено състояние не влияе нито на вероятностното разпределение на следващото състояние, нито на вероятностното разпределение на оставащото време в същото състояние преди следващия преход. Това допускане предполага, че времето на престой в дадено състояние е експоненциално разпределено. По този начин Марковският модел се съгласува със стандартното допускане, че интензивностите на отказите са постоянни. Затова преходите в Марковските вериги с непрекъснато време се характеризират с отрицателно експоненциално разпределение, което определя вероятността за преход в дадено състояние като функция на времето. Интензивността на преходите, λ, се определя за всеки преход. При постоянни интензивности Марковската верига се нарича хомогенна; ако интензивностите са времезависими, Марковската верига е нехомогенна.

Чрез Марковските вериги могат да се определят няколко характеристики на гарантоспособните системи: вероятност за пребиваване в определено състояние, анализ на преходните състояния, анализ на поглъщащите състояния. Марковските вериги дават възможност за моделиране на ремонтируеми системи, на свързани събития, на компоненти с много състояния и др.

Мрежите на Петри (Petri nets) [89], [90] съдържат два вида възли: места и преходи. Местата съдържат краен брой маркери (tokens); броят на маркерите се нарича маркиране. Текущото състояние на системата се моделира чрез текущото маркиране на мрежата, което е броят маркери във всяко място на мрежата. Преходите определят преминаването от едно състояние в друго. Преходът е свързан към множество от входни места посредством входни дъги и към множество от изходни места чрез изходни дъги. Дъгите имат множител. Преходът е задействан, ако всяко входно място съдържа поне броя маркери, определени от множителите на входната дъга. Когато преходът се задейства, определен брой маркери се консумира от входните места и определен брой маркери се произвежда в изходните места. Броят на консумираните и произведените маркери се определя съответно от множителя на входните и изходните дъги. Задействането на преход променя маркирането на мрежата и системното състояние.

Съществуват различни видове мрежи на Петри. Стохастичните мрежи на Петри (Stochastic Petri Nets - SPN) [77], [91], [92] добавят стохастична и времева информация към мрежите на Петри, за да могат да се моделират динамични системи. Към всеки преход в мрежата се добавя време на задействане/запалване, което представлява времето, което трябва да изтече от момента на разрешаване на преход до действителното му запалване, независимо от запалването на други преходи. Обобщените мрежи на Петри (Generalized Stochastic Petri

Nets - GSPN) [77], [93] се характеризират с присъствието на два вида преходи: незабавни преходи, които се задействат веднага щом бъдат разрешени, и преходи по време. Задействането на последните се забавя за период от време, чиято продължителност е случайна променлива, определена от отрицателно експоненциално разпределение с параметър интензивността на задействане на прехода по време. При обобщените мрежи на Петри има забраняващи дъги, които свързват дадено място към преход с цел да забранят прехода.

Проблемът при методите с пространство на състоянията е т.нар. ефект на „експлозия“ на пространството на състоянията – експоненциално нарастване на пространството на състоянията при нарастване на броя на компонентите. Това води до повишаване на цената на изчисленията. В такъв случай мрежите на Петри и Марковските вериги могат да бъдат симулирани с последваща апроксимация на резултатите по отношение на анализа на модела [81].

Симулационното моделиране на гарантоспособните системи се използва, когато аналитичните подходи са или трудни за изпълнение, или не достатъчно точни. Симулацията дава възможност да се моделират и верифицират системи с множество компоненти и голяма степен на детайлизация. Най-често симулацията моделира дискретни събития, каквито са възникването на неизправности и откази, възстановяването от неизправност, ремонтът на компонент или на цялата система, пристигането на съобщение и т.н., които настъпват в дискретни моменти от времето. Симулацията се основава на верига от събития, които се подреждат по време и се обработват последователно в зависимост от момента им на възникване, като се записва съответната статистика. В [34] са откроени ключовите изисквания при създаване на симулационна програма:

- Дълбоко разбиране на симулираната система;
- Изброяване на събитията от интерес за изследването;
- Определяне на зависимостите между събитията (ако съществуват такива);
- Разбиране на преходите между състоянията;
- Правилно оценяване на разпределенията на различните входни случайни променливи;
- Идентифициране на статистическите данни, които ще се събират;
- Коректно анализиране на изходните статистически данни, за да се извлекат желаните системни атрибути.

При симулацията на гарантоспособни системи могат да се определят надеждностните характеристики на системата, като надеждност, готовност, средно време до отказ, средно

време до ремонт, време на престой и др. При определянето на надеждността чрез симулация тя се изпълнява, докато системата влезе в състояние на отказ, и след това се намира общото изтекло време до системен отказ. От тези времена може да се получи функцията на разпределение на времето до първи отказ, чието допълнение е функцията на надеждността $R(t)$.

Основен недостатък на симулационното моделиране, както беше споменато, е продължителността на изпълнение на симулационната програма. Съществуват техники за ускоряване на процеса [34], които позволяват по-бързо да се постигнат желаните резултати.

Методът на моделиране на гарантоспособни разпределени системи чрез Марковски вериги и симулация е предпочетен в дисертацията поради своите предимства:

1. Възможност за моделиране на система с много компоненти и състояния,
2. Възможност за моделиране на различни надеждностни характеристики и изследване на тяхното влияние,
3. Възможност за задаване на променливи характеристики,
4. Възможност за моделиране на независими събития,
5. Възможност за увеличаване на степента на детайлизация на изследваните системни характеристики,
6. Относително лесна програмна реализация.

Сравнително дългото време на изпълнение на симулациите не представлява съществено ограничение при съвременните компютърни системи.

2.1.2 Надеждностни характеристики на отказоустойчивата система с настройваема надеждност

Като част от изискванията към симулационната програма за моделиране на отказоустойчивата система с настройваема надеждност се изчисляват следните характеристики на системата (представени в *Приложение А*):

- надеждност (reliability – *R*);
- средно време до отказ (Mean Time To Failure – *MTTF*);
- средно време до спиране (Mean Time To Stop - *MTTS*);
- общо време на престой (downtime);
- средно време между отказите (Mean Time Between Failures – *MTBF*);
- средно време между спиранията (Mean Time Between Stops – *MTBS*);
- средно време за възстановяване/ремонт след отказ (Mean Time To Repair – *MTTR*);

- готовност (availability – *A*).

Както беше посочено в т. 2.1.1, най-често използваната функция на разпределение при моделиране на отказоустойчиви системи е експоненциалното разпределение. То е подходящо заради свойството си да не съдържа памет за състоянието на системата. Това означава, че разпределението на оставащото време на живот на даден компонент не зависи от това колко дълго е бил в експлоатация, с други думи, че компонентът не остарява. Експоненциалното разпределение в достатъчна степен отразява динамиката на гарантоспособните системи и предлага удобно математическо представяне. Експериментално е доказано, че при хардуерните компоненти съществува дълга фаза от техния живот, съпоставима с периода им на експлоатация, през която интензивността на неизправностите е постоянна. При моделирането това се отразява в допускането за експоненциално разпределение с постоянна интензивността на неизправностите [84].

Надеждността е вероятността системата да бъде в работно състояние в даден момент *t*. Надеждността се определя като функция на времето с израза (П1 от *Приложение А*). Тя представлява вероятността системата да не е отказала в момент *t*. Допуска се експоненциално разпределение на събитията в системата – неизправности на компоненти, възстановяване на компоненти след неизправност, системен отказ и ремонт на системата (при ремонтируеми системи), с постоянна интензивност на неизправностите. При това допускане функцията на разпределение (П4) от *Приложение А* става [37], [84]

$$F(t) = 1 - e^{-\lambda t}. \quad (1)$$

Плътността на разпределение (П20) (*Приложение А*) става [37], [84]

$$f(t) = \lambda e^{-\lambda t}. \quad (2)$$

Надеждността на системата, представена с израза (П1) (*Приложение А*), се изразява като експоненциална функция [37], [84]

$$R(t) = e^{-\lambda t} = 1 - F(t). \quad (3)$$

Средното време до отказ MTTF се изчислява като средна стойност на времето до отказ на компонент за определен период на работа. То е математическото очакване на времето до (първи) отказ (П5) (*Приложение А*). При постоянна интензивност на неизправностите изразът (П5) става [37], [84]

$$MTTF = \frac{1}{\lambda}. \quad (4)$$

Средното време между отказите MTBF е предсказаното изтекло време между присъщите откази на дадена система по време на нейната работа. *MTBF* може да се изчисли като средно аритметичното време между отказите на системата (П7) (*Приложение А*). То обикновено е част от модел, който допуска, че отказалата система се ремонтира незабавно. Това е противоположно на средното време до отказ, което измерва средното време между отказите при допускането, че отказалата система *не се* ремонтира. *MTBF* се измерва за ремонтируеми системи. При експоненциално разпределение и постоянна интензивност на неизправностите $MTBF=1/\lambda$.

Средното време за ремонт MTTR представлява средното време, което се изисква за ремонт на отказали елементи на системата. Изразено математически, това е общото време за ремонтна поддръжка, разделено на общия брой действия за тази поддръжка, за даден период от време (П13) (*Приложение А*). При експоненциално разпределение с постоянна интензивност на неизправностите

$$MTTR=1/\mu. \quad (5)$$

Готовността A е частта от времето, през която системата е в работно състояние. Тя се измерва с вероятността системата да е работоспособна в момент *t*, независимо от това колко пъти е била неработоспособна в интервала (0,*t*) (П10) (*Приложение А*). Тя се определя като отношението на (a) общото време, през което работещ блок е в състояние да бъде използван по време на даден интервал, към (b) дължината на интервала. При системи без ремонт готовността е равна на надеждността.

За постоянни интензивности на неизправностите и ремонтите готовността може да бъде изразена чрез следната формула [37], [84]

$$A = \frac{\mu}{\lambda+\mu} = \frac{MTBF}{MTBF+MTTR}. \quad (6)$$

Това показва, че готовността зависи само от средното време до отказ и средното време до ремонт, а не от разпределението на отказите и ремонтите. Тази формула е в сила за стационарна готовност.

Общото време на *престой* е сумата от всички периоди, през които системата е била неработеща. Средното време за престой се изразява с (П12) (*Приложение А*).

Системата работи при следните определения за отказ и стоп:

Отказ на системата настъпва, когато повече от половината компоненти откажат с неоткрит отказ или при повече от половината отказали компоненти броят на тези с неоткрит отказ е по-голям от броя на компонентите с открит отказ.

$$N_u > \frac{N}{2} \text{ или } N_u + N_d > \frac{N}{2} \text{ и } N_u \geq N_d, \quad (7)$$

където N_u е броят на компонентите с неоткрит отказ, а N_d – броят на компонентите с открит отказ.

Системата *спира*, когато повече от половината компоненти откажат с открит отказ или при повече от половината отказали компоненти броят на тези с открит отказ е по-голям от броя на компонентите с неоткрит отказ.

$$N_d > \frac{N}{2} \text{ или } N_u + N_d > \frac{N}{2} \text{ и } N_d > N_u. \quad (8)$$

При *ремонтируема система* след стоп е възможен ремонт и системата продължава да работи. Изчисляват се *R, MTTF, MTTR, MTTS, MTBS, MTBF, A* и *downtime*.

При *неремонтируема система*, според възприетата за системата дефиниция на стопово състояние, стопът е всъщност отказ в безопасно състояние (fail safe), т.е. компонентът отказва в безопасно за обекта на управление състояние, като евентуално е възможен ремонт, но след много по-дълго време. Стопът от програмна гледна точка е еквивалентен на отказ. Смисълът му е да се определи колко пъти системата попада в безопасно състояние. Изчисляват се *R, MTTF* и *MTTS*.

Резултатите в симулационната програма се определят на базата на следните допускания:

- Експоненциално разпределение на случайните променливи;
- Постоянна интензивност на неизправностите;
- Независимост на отказите и на ремонтите;
- Случайно настъпване на моментите на отказ, определено чрез генератор на псевдослучайни числа.

2.2 Допускания при моделирането на системата

Моделът на отказоустойчивата система с настройваема надеждност е изграден въз основа на допускания, отразяващи нейното поведение и възприетите за изследването режими на неизправност, отказ и ремонт.

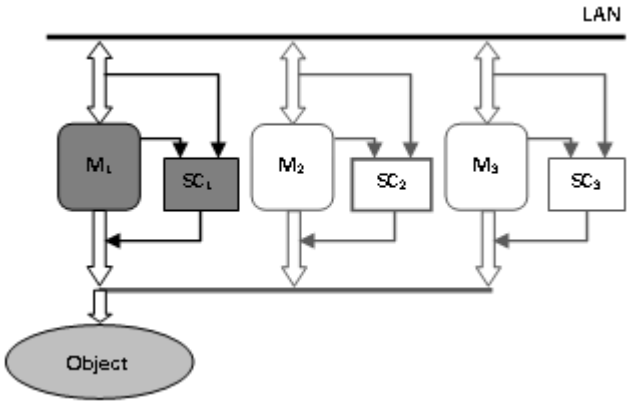
Разпределената система с настройваема надеждност [73], [74], [75], [76] е изградена от компоненти с активен излишък. Компонентите имат еднотипни модули, всеки от които притежава средства за самопроверка с покритие C. Степента на репликиране на компонентите се определя в зависимост от тяхната критичност – колкото по-важен за приложението е даден компонент, толкова по-критичен е той и е по-висока степента му на репликиране. Разглеждат се три степени на критичност/репликиране. Компонентите, чийто отказ не застрашава нормалното функциониране на системата и не води до катастрофални последствия, могат да разчитат само на средствата си за самопроверка и да не бъдат репликирани, т.е. да са ЕМИ компоненти. Техният коефициент на покритие е C_1. Компонентите с по-голяма критичност за приложението, които обаче е достатъчно просто да спрат да извеждат управляващо въздействие в случай на отказ, са компоненти с ДМИ. При тях е възможно отказалият модул да бъде ремонтиран, стига неизправността му да е открита от блока за самопроверка. Компонентите с ДМИ имат коефициент на покритие $C_2>C_1$. Най-критичните компоненти за системата, чийто отказ би имал катастрофални последствия за приложението, се изграждат с ТМИ. При тях компонентът може, освен да открива откази и да предотвратява проникването им към обекта на управление, да маскира неизправност и да позволява ремонт на отказал модул, без да се спира функционирането на целия компонент. Компонентите с ТМИ имат коефициент на покритие $C_3>C_2>C_1$.

Отказоустойчивата система с настройваема надеждност толерира постоянни хардуерни неизправности с интензивност λ_p. При компоненти с двоен и троен модулен излишък е възможно възстановяване (локален ремонт) след неизправност на модул с интензивност на възстановяване μ_p. Системата може да работи без или с ремонт, като при ремонтируема система интензивността на ремонтите е μ_{sys}.

При моделирането на цялата система се разглеждат постоянни хардуерни неизправности на компонент.

2.3 Модел на компонент на системата

Компонентът на разпределената система за управление с настройваема надеждност е изграден от еднотипни модули M_i (i=1, 2, 3), всеки от които има собствен блок за самопроверка SC_i (*Фигура 2-2*) [94]. В зависимост от критичността им модулите могат да се дублират или триплират, за да се постигне по-висока надеждност в точката от управляващия контур, където е разположен компонентът.



Фигура 2-2. Компонент на разпределената система с настройваема надеждност

Един от модулите е основен и той единствен извежда управляващо въздействие към обекта на управление. Всички модули изпълняват едновременно управляващата програма и изчисляват управляващото въздействие (активен излишък). Модулите, които не извеждат резултат към обекта на управление, извършват контрол на основния модул и участват при сравнението на резултатите. Те могат да поемат управлението при отказ на основния модул.

Компонентът е изграден от еднотипни модули, които могат да работят самостоятелно или да се репликират в дублирана или триплирана конфигурация. Всеки модул представлява компютър, снабден с блок за самопроверка, който следи работата на процесора. Модулът има интерфейс за връзка с останалите компоненти на системата през комуникационен канал и интерфейс към обекта на управление. Компонентът отказва в режим „мълчание при отказ" (fail silence), т.е. при открит отказ на компонента към обекта на управление не се извежда управляващо въздействие. Това поведение на компонента при отказ се гарантира от блоковете за самопроверка и чрез репликирането на модулите, които го изграждат. Режимът на „мълчание при отказ" е характерен за системите, които се използват в критични по отношение на безопасността приложения. Той е свързан и с принципа на автономност на възлите. Според този принцип възлите се проектират така, че да не зависят от останалите възли в системата, както и да не влияят върху тяхното поведение.

При единичен модул само средствата за самопроверка могат да открият отказ при изпълнението на управляващата програма и да забранят извеждането на управляващо въздействие. Ако приложението налага по-силна защита на изходите и по-високо покритие на неизправностите, към единичния модул се добавят допълнителни един или два модула със собствени средства за самопроверка, като по този начин се въвежда второ ниво на откриване на неизправностите чрез сравнение. При наличието на повече от един модул в компонента е

възможно ремонтиране на отказал модул, без да се извежда целият компонент от системата. Това увеличава средното време до отказ на компонента и готовността на системата.

Чрез репликирането на модулите в даден компонент може да се отговори на различни нива на критичност, диктувани от приложението на отказоустойчивата система с настройваема надеждност.

2.3.1 Функции на системен компонент

Характерно за предложения подход е вграждането на локален блок за самопроверка във всеки модул, реализация на протокол за разпределено гласуване и следене на състоянието. Блокът за самопроверка [73] е вграден като допълнителен блок към конфигурацията на модула и има следните функции:

1. Проверяващи функции:
    - Проверка на хода на изпълнение на програмата чрез сигнатурно-времеви механизми (например, [7], [8], [9], [10], [11], [12], [13], [95], [96], [97]);
    - Сравнение на резултатите и/или гласуване.
2. Управляващи функции:
    - Разрешение на изходните схеми;
    - Спиране на процесора на модула;
    - Синхронизация по времеви прозорец.

Разглежда се периодична разпределена система, която работи с твърди времеви ограничения. Обменът на съобщения между компонентите се осъществява посредством стратегия „времеделене и множествен достъп“, при която всеки компонент извежда съобщение в определен времеви прозорец.

2.3.2 Режим на отказ

Компонентът отказва, когато откажат всички негови модули. Благодарение на въвеждането на средства за самопроверка и излишък не всеки неоткрит отказ в отделен модул води до отказ на целия компонент. Възможни са две състояния при отказ на модул – стоп и отказ. Дефинирането на тези две състояния зависи от това колко модула изграждат компонента. Най-общо компонентът е в състояние на отказ, когато неизправност в някой или всички негови модули достигне до обекта на управление. Компонентът преминава в стопово състояние, когато неизправността е открита, но са изчерпани ресурсите му да работи нормално.

По-строгите дефиниции за стопово и отказово състояние са следните.

Даден компонент преминава в *стопово състояние*, когато има разлика при сравнението на резултатите между модулите и блокът за самопроверка на отказалия модул е открил неизправността. В случай на триплиран компонент, когато сравнението се прави с мажоритарно гласуване, компонентът спира, когато няма мажоритарен резултат или има два модула с открита неизправност.

Даден компонент преминава в *отказово състояние*, в случай че едновременно откажат всичките му модули или неизправността не е открита, а сравнението не показва разлика.

При стопово състояние на компонент:

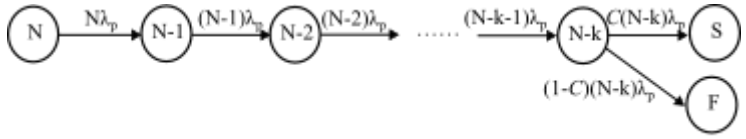
- Той се изключва от системата, при което не комуникира с останалите компоненти;
- На изходите се поддържа безопасно управляващо въздействие (fail safe).

Компонентът подлежи на ремонт както след спиране, така и след отказ. Предимството на стоповото състояние е във възможността да се диагностицира по-лесно и бързо причината за отказ чрез информацията от средствата за самопроверка, което намалява времето за престой. Друга важна роля на средствата за самопроверка (блок за самопроверка и сравнение) е, че дават възможност за ремонтиране на модул с открита неизправност, без да се спира работата на целия компонент. Това води до повишаване на готовността на компонента и на разпределената система като цяло. При неоткрита неизправност на компонент той се ремонтира заедно с цялата система.

2.4 Модел на системата

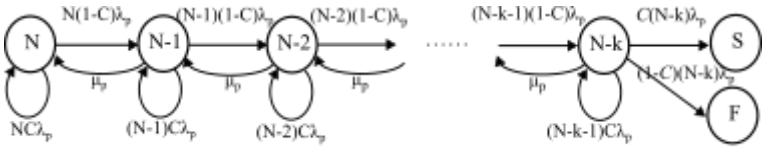
Работата на системата е представена с Марковски процес, където всяко от състоянията в Марковската верига изобразява състоянието на системата след настъпила неизправност, според броя на работещите компоненти N, а дъгите са преходите между отделните състояния при възникване на постоянна неизправност с интензивност на неизправностите λ_p. Покритието на механизмите за самопроверка е C. Разгледани са различни възможности за работата на системата: с и без възстановяване след постоянна неизправност и с и без системен ремонт.

При положение че системата работи до изчерпване на ресурсите си, влиянието на коефициента на покритие C върху надеждността е незабележимо (*Фигура 2-3*). Даден компонент отказва с открит отказ, но от системна гледна точка това е просто един компонент по-малко. Фактът, че отказът е открит, не подобрява надеждността.



Фигура 2-3. Марковски модел на система без възстановяване след постоянна неизправност и без ремонт

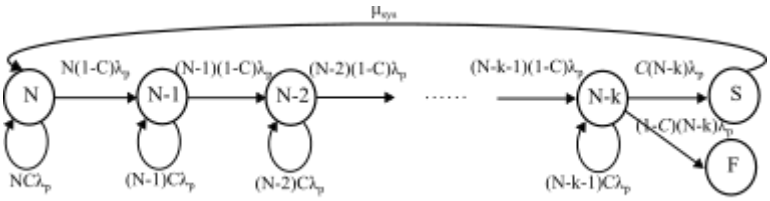
Ако C спомага за удължаване на живота на компонента, това влияе и върху надеждността на системата. Това именно се показва, когато има частичен ремонт на компонент с открит отказ (*Фигура 2-4*). Тогава животът на системата се удължава, тъй като отказалите компоненти се възстановяват и се връщат отново в системата. Влиянието на C расте, защото само при детектиран отказ на компонент той може да бъде ремонтиран преди да настъпи системен отказ.



Фигура 2-4. Марковски модел на система с възстановяване от постоянна неизправност, но без ремонт

Възстановяването на компонент след постоянен отказ се изразява във възможността след *открит* постоянен отказ компонентът да бъде върнат в работоспособно състояние. Това може да се случи след замяна на компонента с нов или след диагностика и ремонт на отказалия компонент. При моделирането се приема, че такъв локален ремонт винаги е възможен и успешен и че ремонтът не зависи от модулността/степента на репликиране на компонента.

Ремонтируемата система (*Фигура 2-5*) също има две крайни състояния: отказ и стоп. За разлика от неремонтируемата (*Фигура 2-4*) обаче тя може да бъде възстановена след спиране с интензивност на потока на системните ремонти μ_{sys}.

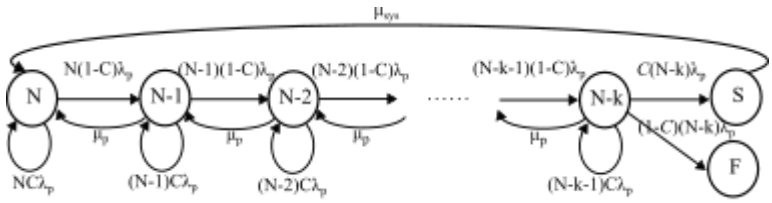


Фигура 2-5. Марковски модел на система с ремонт, но без възстановяване от постоянна неизправност

Системен ремонт се извършва след попадане на системата в стопово състояние. При ремонта отказалите компоненти – независимо от това дали отказите им са били открити по време на работа на системата, или не – биват заменени с нови. Докато системата е в ремонт, в нея не настъпват нови откази. Не се прави разлика между изправните и отказалите компоненти

по отношение на ремонта и на интензивностите на отказите им. Изправните компоненти продължават работа след ремонта със същите начални характеристики. При моделирането времето на ремонта се добавя към момента на попадане на системата в стопово състояние, а времената на следващите откази се определят след времето на завършване на ремонта.

Отказоустойчивата разпределена система с настройваема надеждност може да работи и със системен ремонт и възстановяване на компонент от постоянна неизправност (*Фигура 2-6*). Това предполага постигане на по-добри надеждностни характеристики.



Фигура 2-6. Марковски модел на система с ремонт и възстановяване от постоянна неизправност

Всички описани възможности в Марковските модели (*Фигура 2-3* - *Фигура 2-6*) са изследвани чрез симулационното моделиране на системата, а резултатите са представени в *Глава 3*.

2.5 Изводи и резултати

В *Глава 2* е представена разработената от автора отказоустойчива разпределена система с настройваема надеждност. Тя предлага различни степени на репликиране на хардуерните компоненти. Необходимостта системата да има компоненти с различна степен на репликиране е свързана с тяхната критичност, която се определя от изискванията на приложението. Контролираните в работната среда на дадена система параметри имат различна важност за нейното безотказно функциониране. Компонентите на системата, които управляват определени параметри, съответно имат различна критичност. Затова те могат да бъдат изградени с различна степен на репликиране и това да не води до намаляване на надеждността на системата. Отказоустойчивата система с настройваема надеждност е моделирана при определени допускания за нейната работа с цел да се провери тази хипотеза.

Представени са различни методи за моделиране на гарантоспособни системи, които предоставят възможности за определяне на надеждностните характеристики на системите, като надеждност, готовност, средно време до отказ, средно време за ремонт, средно време между отказите, средно време между ремонтите и време за престой.

Предложената от автора отказоустойчива разпределена система с настройваема надеждност е моделирана въз основа на използване на Марковски вериги и валидирана чрез симулация. Методът на симулационното моделиране е избран, защото симулацията дава възможност да се моделират системи с множество компоненти и голяма степен на детайлизация. Изследваните надеждностни характеристики са определени за експоненциално разпределение и постоянни интензивности на неизправностите и ремонтите. Представени са моделите на компонент на системата и на цялата система. Предвидени са възможности за моделиране на системата с настройваема надеждност с и без възстановяване от постоянна неизправност, както и с и без системен ремонт.

Представените в *Глава 2* резултати са следствие на изпълнението на изследователски задачи 2 и 3 на дисертацията. Постигнатите научни резултати са:

1. Представен е авторски архитектурен модел на отказоустойчива разпределена система с настройваема надеждност, като са дефинирани изискванията към нейните компоненти и системата като цяло.
2. Създаден е модел на предложената отказоустойчива разпределена система с настройваема надеждност;

Част от резултатите по тази глава са публикувани в:

1. Djambazova, E., & Andreev, R. (2023). Redundancy management in dependable distributed real-time systems. *Problems of Engineering Cybernetics and Robotics*. (под печат)
2. Djambazova, E. (2012). Adjusting reliability of a fault-tolerant distributed process control system – Preliminary results. International Conference “Automatics and Informatics 2012”, Sofia, Bulgaria, pp. 175-178.
3. Djambazova, E. (2009). Node reliability of a fault-tolerant distributed process control system – Simulation results. International Conference “Automatics and Informatics’ 2009”, Sofia, Bulgaria, pp. I-131 – I-134.
4. Джамбазов, К., & Ананиева, Е. (1995). Управляващи системи с модулно настройване на отказоустойчивостта. Национална конференция с международно участие „Автоматика и информатика ‘95”, София, стр. 247-250.

Глава 3 Изследване на отказоустойчивата система с настройваема надеждност

Отказоустойчивата система с настройваема надеждност, чиято архитектура и модел бяха представени в *Глава 2*, управлява обект, като получава входни данни от неговите сензори, изпълнява управляващ алгоритъм и изчислява изходни резултати, които извежда към активаторите на обекта. Параметрите на обекта на управление могат да имат различна важност за коректната системна работа. Отказ в точка от управляващия контур, където се отчита/измерва и управлява даден параметър, може да има катастрофални последствия. Предполага се, че компонент на системата с два или три модула може да осигури по-висока надеждност. Единичните компоненти имат сравнително ниско ниво на отказоустойчивост, но те могат да се използват за управление на по-малко важни параметри на обекта на управление. Отказ на единичен компонент може да бъде приемлив за системата. Единичните компоненти все пак имат средства за отказоустойчивост, вградени в техните блокове за самопроверка. Настройването на системната надеждност посредством разпределяне на степента на репликиране на компонентите според нуждите на управлявания обект може да използва ресурсите на системата по-ефикасно и би могло да подобри нейната надеждност.

Симулационното моделиране на системата с настройваема надеждност изследва как промяната на структурния излишък влияе върху общата системна надеждност. Както беше посочено в *Глава 2*, компонентите с различна степен на репликиране имат различно ниво на критичност за системата. Това определя и коефициента на покритие на техните средства за самопроверка. Единичните компоненти имат най-малък коефициента на покритие – C_1. Компонентите с двоен модулен излишък имат коефициент на покритие $C_2>C_1$, а триплираните компоненти имат коефициент на покритие $C_3>C_2>C_1$. Коефициентите на покритие могат да се изменят в зависимост от условията на средата или от условията на функциониране в рамките на някакви граници. Компонентите са разположени на различни места и понасят различни въздействия на околната си среда. Покритието на техните средства за отказоустойчивост (блокове за самопроверка и степен на репликиране) зависи от специфични характеристики на хардуера, на изпълняваната управляваща програма, на разпределението на паметта и т.н. По този начин компонентите с еднаква степен на репликиране могат да имат различни коефициенти на покритие, но в едни и същи граници.

Тези коефициенти заедно с интензивността на неизправностите λ_p определят поведението на компонентите в симулационния модел на отказоустойчивата система с

настройваема надеждност. Събитията в системата са свързани с промяната на нейното състояние. Това са случайни събития, характеризиращи се със съответната интензивност на възникване:

- интензивност на постоянна неизправност на компонент λ_p,
- интензивност на възстановяване на компонент след постоянна неизправност μ_p,
- интензивност на ремонтите на системата след открит отказ μ_{sys}.

Допуска се, че отказал компонент може винаги да бъде възстановен след постоянна неизправност (локален ремонт), а времената до отказ са нормално разпределени.

Разработен е следният *изследователски протокол*:

1. Изследване на компонент
2. Изследване на системата
   - Входни данни: брой компоненти в системата *N*, брой модули в системата *M*, интензивност на постоянните неизправности λ_p, интензивност на възстановяване след постоянна неизправност μ_p, интензивност на ремонтите μ_{sys}, граници на коефициентите на покритие C_1, C_2 и C_3.
   - Изходни резултати: *R*(*t*), *A*, *MTTF*, *MTTR*, *MTTS*, *MTBF*, *MTBS*, *downtime*
   - Определяне на възможните разпределения на структурния излишък при зададените *N* и *M*.
3. Определяне на времената до отказ при дадените входни параметри за всички разпределения на модулния излишък на системи с и без настройване на структурния излишък.
4. Изследване на влиянието на различни параметри върху надеждностните характеристики на отказоустойчивата система с настройваема надеждност и на системи без разпределение на структурния излишък.
5. В зависимост от резултатите определяне на конфигурациите с най-висока системна надеждност.

3.1 Симулационно моделиране на отказоустойчива система с настройваема надеждност

Избраният в *Глава 2* изследователски метод на симулационно моделиране е приложен чрез разработване от автора на дисертацията на симулационна програма. Тя е проектирана според описания изследователски протокол и изискванията за определяне на надеждностните характеристики на изследваната система, формулирани в т. 2.1.2. Програмата симулира отказоустойчиви системи с различно разпределение на структурния излишък и системи без

разпределение на структурния излишък, което дава възможност за тяхното сравнение и анализиране.

3.1.1 Симулационна програма

Разработеният програмен продукт NMRSIM за симулиране на поведението на отказоустойчиви разпределени системи (представен по-пълно в *Приложение В*) определя показателите на системите от интерес за изследването и дава възможност за сравнение на предложената система със системи без настройване на надеждността. Програмата е написана на език за програмиране C++ и разработването ѝ изпълнява следната последователност:

1. Определяне на основните изисквания към симулационната програма;
2. Разработване на алгоритъм за работа на симулационната програма;
3. Изготвяне на структура на програмния продукт.

Изискванията към програмата за симулационно моделиране на отказоустойчивата разпределена система за управление с настройваема надеждност са:

1. Да симулира поведението на системата във времето, като отразява зададените дефиниции за стоп и отказ;
2. Да представя настъпването на постоянна неизправност като стохастичен процес с експоненциално разпределение и интензивност на неизправностите λ_p;
3. Да има възможност да моделира система с ремонт и без ремонт;
4. Да представя моментите на извършване на ремонтите като стохастичен процес с експоненциално разпределение и интензивност на ремонтите μ_p (за локален ремонт) и μ_{sys} (за системен ремонт);
5. Да моделира система с произволен брой модули и компоненти;
6. Да изчислява надеждностните характеристики, посочени в *Глава 2*, т. 2.1.2;
7. Да моделира поведението при неизправност на компонент и цялата система;
8. Да разпределя структурния излишък при зададен общ брой компоненти и общ брой модули;
9. Да моделира коефициента на покритие на средствата за самопроверка;
10. Да изчислява надеждността като функция на времето;
11. Да изчислява статистическите показатели на получените резултати;
12. Да съхранява получените резултати.

Програмният продукт за симулационно моделиране на предложената отказоустойчива разпределена система с настройваема надеждност е основан на моделиране на функционирането на системата във времето и моментите на настъпване на неизправност.

Процесът на работа на системата е представен като поредица от единици време *t*, които маркират коректното обслужване от страна на системата. Моментите на смущение в коректното обслужване, т.е. на възникване на неизправност t_F, се моделират въз основа на допускането за експоненциално разпределение и постоянна интензивност на неизправностите λ_F (формула (3)) чрез генератор на псевдослучайни числа. Средствата за самопроверка са представени чрез коефициентите на покритие C_1, C_2 и C_3.

Алгоритъмът на симулационната програма има структурата от *Фигура 3-1*.

```
determine component redundancy distributions
assign Nc initial values of C1, C2, C3
for each C1 do
   for each C2 do
      for each C3 do
         for all module redundancy distributions do
            while system is operational do
               determine time to system failure
               save time to failure
            end
            determine mttf, mtbf, availability, reliability
         end
         save dependability characteristics for the current distribution
      end
end
```

Фигура 3-1. Псевдокод на симулационната програма

Моментите на възникване на първата неизправност се определят за всички компоненти на системата в зависимост от зададената интензивност на неизправностите. Чрез сравнение на тези моменти се определя кога настъпва първата неизправност и в кой компонент възниква тя. В зависимост от коефициента на покритие на съответния компонент неизправността може да бъде открита или не. В случай на открита неизправност се проверява дали тя може да бъде отстранена и съответният модул - ремонтиран. Ако системата работи без локален ремонт (μ_p=0), откритата неизправност води до спиране на компонента и той може да бъде поставен в безопасно състояние – стоп. Ако системата има достатъчно компоненти, тя продължава да функционира с намален ресурс. Определят се моментът и местоположението на следващата неизправност и този процес продължава, докато системата попадне в отказово състояние. Процесът се повтаря циклично, като се записват моментите на системен отказ и на спиране за всяка итерация. Въз основа на натрупаните данни се изчисляват надеждността и останалите надеждностни характеристики. При системи с ремонт, ако неизправността е била открита, отказалият модул може да бъде ремонтиран, като се определя времето до ремонт. Ако, докато модулът се ремонтира, системата има достатъчно ресурси, след изтичане на времето за ремонт за модула се определя моментът на следваща неизправност. При ремонтируемите системи се

използва и стопово състояние, при което системата извежда безопасно управляващо въздействие, докато се ремонтира.

Описаният цикъл на работа се повтаря за всички определени в началото разпределения на структурния излишък, като надеждностните характеристики се съхраняват във файлове, които могат да бъдат изобразявани и сравнявани.

На *Фигура 3-2* е показана блоковата структура на симулационната програма.

Фигура 3-2. Структура на симулационна програма NMRSIM

- Блок на входни данни

  В него се задават N, M, λ_p, μ_p, μ_{sys}, C_1, C_2, C_3 и R_{total}.

- Генератор на псевдослучайни числа, работещ според алгоритъма, представен в [98]

  Той се използва при определяне на момента на настъпване на неизправност и за генериране на случайни стойности на коефициентите на покритие. За да се постигне независимост на неизправностите и отказите, всеки компонент има собствен генератор на псевдослучайни числа, определящ моментите на неговите неизправности. По същата причина се използват различни генератори на псевдослучайни числа за определяне дали дадена неизправност е открита в зависимост от степента на репликиране на компонента с неизправност.
- Блок за определяне на конфигурациите на структурния излишък

  При дадени N и M се определят всички възможни разпределения на структурния излишък.
- Блок за определяне на неизправност в компонент

  В него се определят моментите на неизправност за съответния компонент.
- Блок за определяне на текущата неизправност

  Текущата неизправност се определя след сравнение на моментите на неизправност във всички компоненти.
- Блок за определяне на коефициентите на покритие на отделните компоненти

  В зависимост от степента на репликиране се задават коефициентите C_1, C_2 и C_3.
- Блок за определяне на състоянието на компонент в зависимост от коефициента му на покритие

  В този блок се определя дали компонентът е отказал, дали отказът му е открит и дали е в ремонт.
- Блок за определяне на системното състояние в зависимост от състоянието на отделните компоненти

  В този блок се определя дали системата е в стопово или отказово състояние според дефинициите за стоп (8) и отказ (7) (*Глава 2*, т. 2.1.2). При наличие на системен отказ се записват всички натрупани времена до отказ и се преминава към определяне на надеждностните характеристики. Ако системата е в стопово състояние, цикълът продължава, а моментът на спиране се записва.
- Блок за изчисляване на надеждностните характеристики

  Определят се *R*, *MTTF*, *MTTR*, *MTBF*, *MTBR*, *MTTS*, *MTBS*, *A* и *downtime*.

- Блок за определяне на конфигурациите, които постигат зададената системна надеждност

  Надеждността на всички конфигурации на структурния излишък се сравнява с R_{total}.
- Блок за определяне на статистически параметри

  Определя се стандартното отклонение на натрупаните данни за времето до отказ.
- Блок на изходните резултати

  Получените резултати за надеждностните характеристики на всички конфигурации на структурния излишък се записват във файлове.

Взаимодействието между отделните блокове на симулационната програма е изобразено схематично на *Фигура 3-2*.

Блоковата структура на програмата дава възможност тя да бъде разширявана и надграждана, за да могат да се моделират и други гарантоспособни разпределени системи.

Статистическа обработка на резултатите

Моментите на настъпване на отказ представляват генерална съвкупност според статистическата терминология [99]. Тъй като размерът ѝ е голям (над 40 според изискванията на статистическите пресмятания; в нашия случай размерът на генералната съвкупност е 10^5), според централната гранична теорема [37], [100], [101] може да се допусне нормално разпределение на генералната съвкупност [99]. При тези допускания могат да се определят доверителни интервали за *MTTF*, *MTBF*, *MTTS*, *MTBS*, времето за престой, които са средно аритметични стойности.

Надеждността се изчислява в зависимост от времената до отказ и можем да допуснем, че, ако те представляват достатъчно голяма извадка, нейното определяне е достоверно. Готовността се изчислява по познатата формула (6) (*Глава 2*, т. 2.1.2) и е функция на *MTBF* и *MTTR*.

Моментите на настъпване на отказ се определят с помощта на генератор на псевдослучайни числа, като се допуска експоненциално разпределение и постоянна интензивност на отказите. Изпълняват се *n* итерации, като за всяка се записват моментите на системен отказ. Тази поредица от времена се разглежда като генерална съвкупност и въз основа на нея се определят стандартното отклонение и доверителният интервал на средното време между отказите.

Случайната променлива при симулацията на ОРС с настройваема надеждност е TBF, времето между отказите. За всяка итерация на симулационната програма се определят *n* негови

стойности, означени с tbf_1, tbf_2, …, tbf_n, като за тях се използва функция T(tbf_1, tbf_2, …, tbf_n) за оценка на θ – параметъра на функцията на разпределение на TBF. Оценката на θ е означена с $\hat{\theta}$.

За определяне на доверието в оценката на θ се изчислява доверителен интервал, в който се очаква да попадне θ. Той се дефинира по следния начин [34]:

Доверителен интервал с *доверително ниво* 1-α за параметър θ е интервалът [a, b], изчислен като функция на извадка с размер *n*, tbf_1, tbf_2, …, tbf_n, по такъв начин, че, ако се изчислят подобни интервали въз основа на по-голям брой извадки с размер *n*, частта 1-α от тези интервали действително ще включва параметъра θ.

1-α обикновено се избира да бъде 0.95 или 0.99 (или 95% или 99%). В симулационната програма доверителните интервали се използват за оценяване на математическото очакване на случайната величина TBF. Избрано е доверително ниво 95%, защото се смята, че то в достатъчна степен гарантира, че получените резултати за средната стойност попадат в съответния интервал.

Генералната съвкупност е *tbf*, размерът ѝ е *n*. Да се определи доверителният интервал за *MTBF* при доверително ниво 95%.

$$\alpha=1-0.95=0.05 \quad (9)$$

От съответната таблицата се определя z=1.96.

При голяма извадка за изчисляване на стандартното отклонение се използва формулата [99]:

$$\sqrt{\frac{\sum(tbf_i-\overline{tbf})^2}{n}}, \quad (10)$$

където $\overline{tbf}$ е средната на извадката.

Както е отбелязано в [99], в статистическите програми се използва формулата:

$$s=\sqrt{\frac{n(\sum tbf^2)-(tbf)^2}{n(n-1)}}. \quad (11)$$

Именно тя е използвана и в симулационната програма.

Стохастичната грешка е:

$$\sigma_{\overline{tbf}}=\frac{s}{\sqrt{n}}. \quad (12)$$

Доверителният интервал е:

$$MTBF \pm 1.96 * \sigma_{\overline{tbf}}. \quad (13)$$

С вероятност 0.95 може да се твърди, че $MTBF$ е в границите $MTBF \pm 1.96 * \sigma_{\overline{tbf}}$.

3.2 Резултати от симулационно моделиране на системата

Отказоустойчивата система с настройваема надеждност е симулирана при следните параметри: брой компоненти N=10 и N=20, брой модули съответно M=20 и M=40, интензивност на постоянните неизправности $\lambda_p=10^{-3}$ 1/h и $\lambda_p=10^{-4}$ 1/h[6], интензивност на възстановяване след постоянна неизправност μ_p=0.1 1/h, интензивност на ремонтите μ_{sys}=0.1 1/h, граници на коефициентите на покритие $C_1 \in [0.8, 0.9)$, $C_2 \in [0.9, 0.95)$ и $C_3 \in [0.95,\ 1.0)$.

Отказоустойчивите разпределени системи за реално време работят в разнообразна среда, където са подложени на влиянието на различни фактори. Освен че това води до неизправности и откази, то влияе и върху покритието на средствата за самопроверка на компонентите. Могат да се задават еднакви стойности на C за всички възли с еднаква степен на репликиране, а могат да се задават и различни стойности, но в рамките на посочените интервали. Възлите имат различни, но постоянни за всеки опит, стойности на C в рамките на интервала за тяхната степен на репликиране. На практика модулите нямат напълно еднакви покрития и не работят в напълно идентични условия. В подкрепа на допускането, че възлите с еднаква степен на репликиране могат да имат различно покритие, но в рамките на някакви граници, са:

- Резултатите за единичен компонент и влиянието на неговото покритие в зависимост от степента на репликиране [12], които са разширени в т. 3.2.1;
- Резултатите за покритието на различни средства за самопроверка в единичен компонент, като контролен таймер (watchdog timer) [13], [14], [15], [16], [97], сигнатурни техники за контрол на потока от инструкции [8], [9], [10], където покритието се влияе от специфичните характеристики, както на процесора (по-общо, хардуера), така и от изпълняваната управляваща програма, разпределението на паметта и т.н.

Множество изследвания показват, че коефициентът на покритие на средствата за самопроверка не е постоянна величана, а зависи от потока и набора от инструкции на процесора, разпределението на паметта, случайни и постоянни неизправности в системата и т.н. [8], [11], [12], [15], [96]. Предложени са техники за неговата оценка [7], [9], [10], [13], [14],

[6] Интензивностите на събитията в системата са илюстративни, като е спазено реалистично съотношение между интензивностите на неизправностите и на ремонтите – времената за ремонт са много по-кратки от тези между неизправностите.

[16], [19], [95] [97], които дават основание да се приеме, че коефициентите на покритие на средствата за самопроверка, а следователно и на компонентите, се менят в рамките на определени интервали. Размерността на интервалите за коефициентите на покритие зависи от конкретното приложение, от контекста и средата на работа на системата и от изискванията към нея. В дисертационния труд са приети примерни стойности, като е запазено съотношението $C_3>C_2>C_1$.

При изследването на различните разпределения на структурния излишък в компонентите е използвана следната нотация:

система (*i, j, k*),

където

i – брой на единичните компоненти,

j – брой на компонентите с двоен модулен излишък,

k – брой на компонентите с троен модулен излишък.

Например, система (3,4,3) при N=10 и M=20 означава система с 3 единични, 4 дублирани и 3 триплирани компонента.

Изследвани са разпределения на структурния излишък, които запазват общия брой на модулите в системата. Граничните случаи на системи, изградени само от единични компоненти и само от триплирани компоненти са проиграни единствено при настройването на модела. Логично е да се очаква, че системите само с ТМИ компоненти имат най-висока надеждност. Точно такива са високонадеждните системи, описани в *Глава 1* (т. 1.2 и т. 1.6), като TTA, DECOS, DEAR-COTS и др. Има и системи, които са изградени от модули с ДМИ (*Глава 1*, т. 1.2 и т. 1.6), като MARS и Dacapo, където структурният излишък на хардуерните компоненти е еднакъв за всички компоненти. Хипотезата на настоящото изследване предполага проучване на възможностите за разпределение на хардуерните ресурси на системата според тяхната критичност при определени изисквания за надеждност, съобразени с приложението. Затова като еталонна система за сравнение е избрана система, изградена само от дублирани компоненти, която не разпределя структурния излишък. При това положение броят на модулите в системата е M=2N. Интерес представляват само конфигурациите, които удовлетворяват ограниченията за N и M.

3.2.1 Изследване на компонент

Резултатите от симулационното моделиране са представени за компонент, изграден от два и от три модула. Показателите на надеждността са отчетени за случаите на компонент с локален ремонт и без локален ремонт. Изследвано е влиянието на постоянни и случайни

неизправности. Данните са за следните интензивности на неизправностите и ремонтите[7]: постоянни неизправности на процесор λ_p=10^{-2} 1/h, случайни неизправности на процесор λ_t=0.1 1/h, възстановяване на процесор след постоянна неизправност μ_p=0.1 1/h, ремонт на компонент μ_c=0.1 1/h.

Симулационно моделиране на компонент с двоен модулен излишък

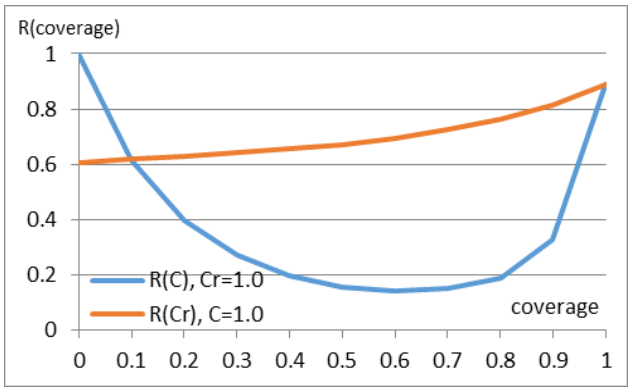
При компонент с двоен модулен излишък отказоустойчивостта се постига чрез блоковете за самопроверка на двата модула (с коефициент на покритие *C*) и чрез сравнение на резултатите на модулите. Компонентът *отказва*, когато едновременно откажат и двата процесора на модулите **и** средствата за самопроверка не са открили отказа. Компонентът попада в *стопово състояние*, когато сравнението показва разлика, но средствата за самопроверка **не са** открили отказа.

Частични резултати от изследването на надеждностните характеристики на компонент на разпределената система за управление с настройваема надеждност са представени в [94]. Впоследствие те са допълнени и са представени по-долу. На фигурите са изобразени изследваните надеждностни характеристики на компонент във функция от коефициента на покритие на средствата за самопроверка *C* и коефициента на възстановяване след случайна неизправност C_r.

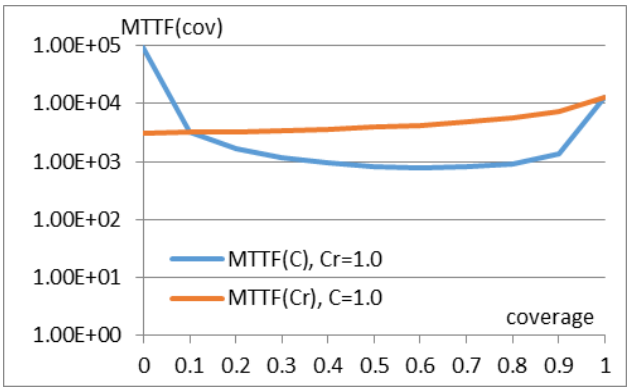
На *Фигура 3-3 – Фигура 3-10* са показани надеждностните характеристики на дублиран компонент във функция от коефициентите *C* и C_r за система с локален ремонт. Компонентът е симулиран за работен период 10^5 часа.

Потвърждава се изводът от [94], че за ниски и високи стойности на коефициента на покритие дублираният компонент има по-висока надеждност, *MTTF*, *MTTS* и време за престой (*Фигура 3-3*, *Фигура 3-4*, *Фигура 3-6* и *Фигура 3-10*), отколкото за средни стойности на този коефициент. Това се дължи на влиянието на сравнението, което при ниски стойности на *C* на практика го неутрализира, защото при сравнение неизправностите в компонента се откриват с вероятност 1. При високи стойности на *C* покритието на неизправностите има силно значение за по-добрите надеждностни показатели на компонента. Коефициентът на покритие на средствата за самопроверка подобрява готовността на компонента (*Фигура 3-9*) и почти не влияе върху неговото време за ремонт (*Фигура 3-5*).

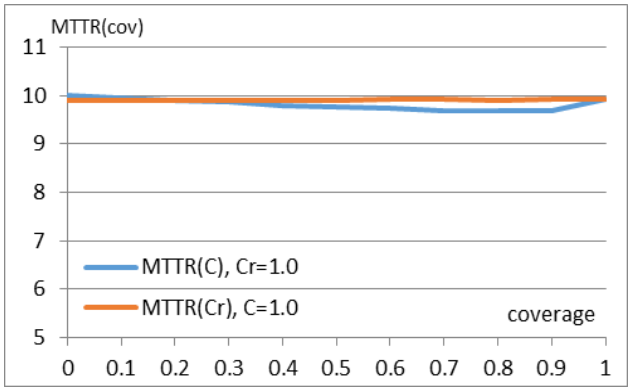
[7] Стойностите на интензивността на неизправностите са примерни и са подбрани така, че да се изпълнява по-бързо програмата. Това не нарушава работата ѝ по същество.



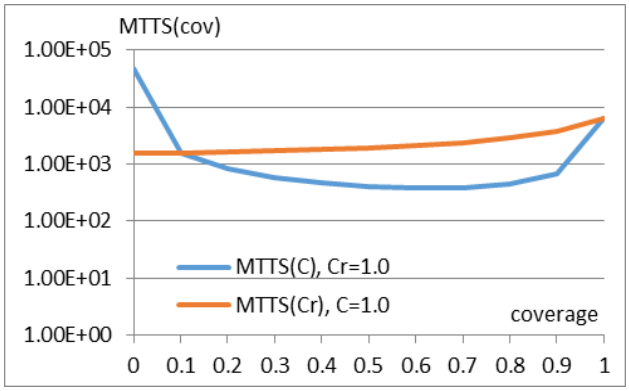
Фигура 3-3. Надеждност на дублиран компонент на система с локален ремонт



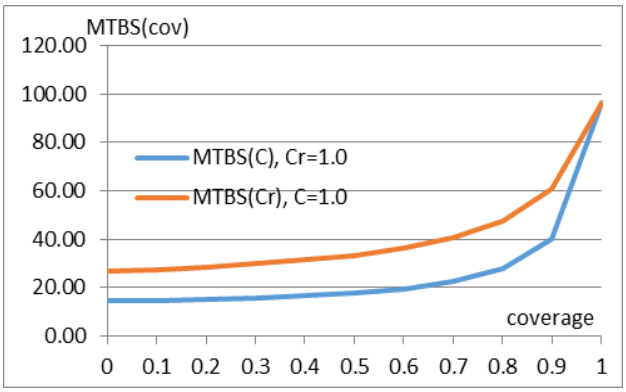
Фигура 3-4. MTTF на дублиран компонент на система с локален ремонт



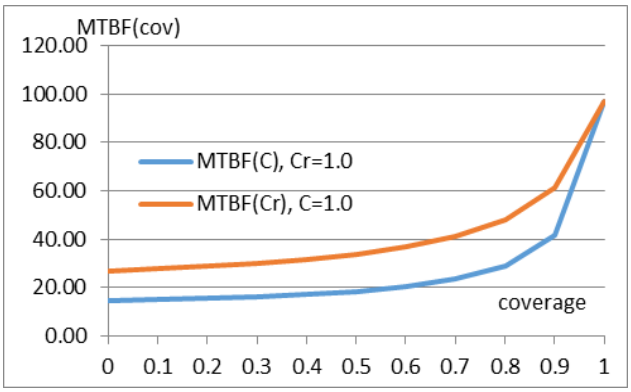
Фигура 3-5. MTTR на дублиран компонент на система с локален ремонт



Фигура 3-6. MTTS на дублиран компонент на система с локален ремонт

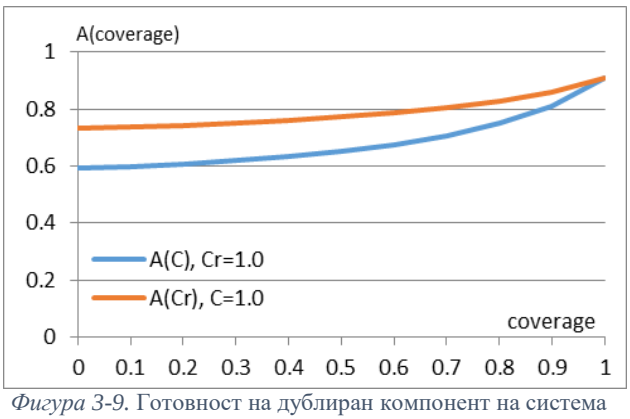


Фигура 3-7. MTBS на дублиран компонент на система с локален ремонт

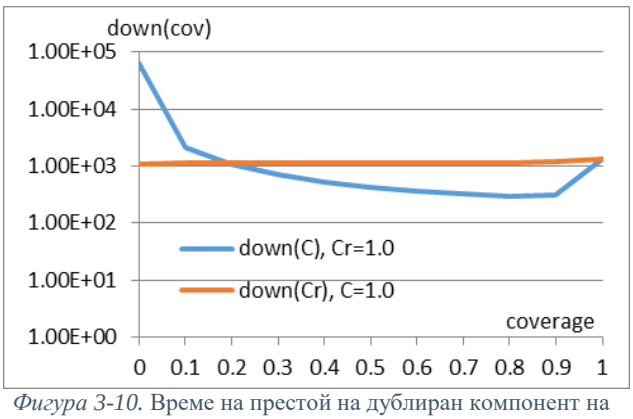


Фигура 3-8. MTBF на дублиран компонент на система с локален ремонт

Коефициентът на възстановяване след случайна неизправност C_r има много добро влияние върху способността на компонента да се ремонтира (особено видно от *Фигура 3-7* и *Фигура 3-8*). Отстраняването на случайните неизправности удължава времето, през което компонентът е работоспособен.

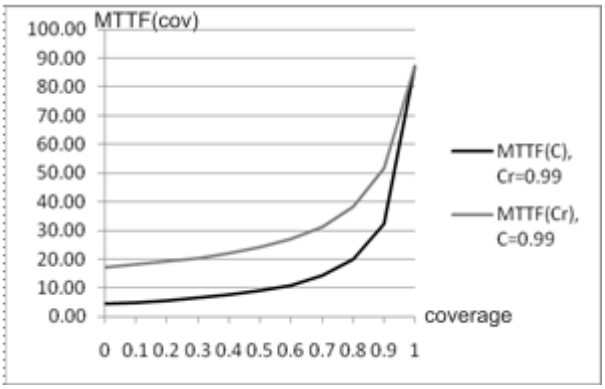


Фигура 3-9. Готовност на дублиран компонент на система с локален ремонт

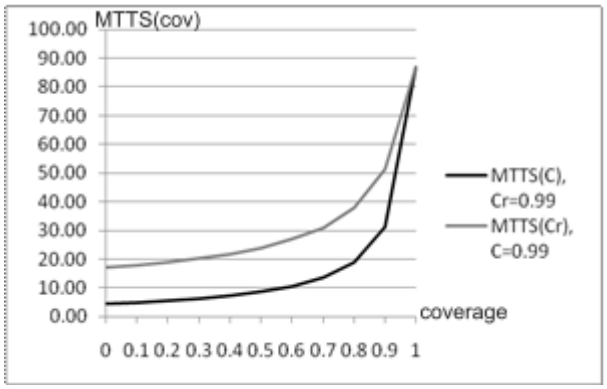


Фигура 3-10. Време на престой на дублиран компонент на система с локален ремонт

На *Фигура 3-11* и *Фигура 3-12* са показани надеждностните показатели на дублиран компонент за система без локален ремонт. Вижда се, че средствата за самопроверка и възстановяването след случайна неизправност чувствително подобряват времената до отказ и спиране в безопасно състояние.



Фигура 3-11. MTTF на дублиран компонент на система без локален ремонт



Фигура 3-12. MTTS на дублиран компонент на система без локален ремонт

Симулационно моделиране на компонент с троен модулен излишък

При компонент с троен модулен излишък отказоустойчивостта се постига отново чрез блоковете за самопроверка, а сравнението е всъщност мажоритарно гласуване.

Отказ настъпва, когато повече от половината модули са с неоткрита неизправност. Компонентът попада в *стопово състояние*, когато повече от половината модули са с открита неизправност. Модулите могат да бъдат в едно от следните състояния: нормално, отказово с открит отказ и отказово с неоткрит отказ. Точната зависимост между броя на модулите в съответно състояние и състоянието на компонента е показана в *Таблица 3-1*.

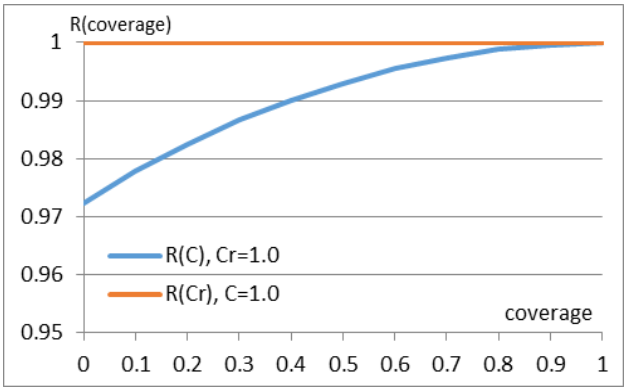
Таблица 3-1. Взаимна връзка между състоянието на модулите и състоянието на компонента

Нормално състояние	**Открит отказ**	**Неоткрит отказ**	**Състояние на компонента**
3	0	0	Нормално състояние
2	1	0	Нормално състояние

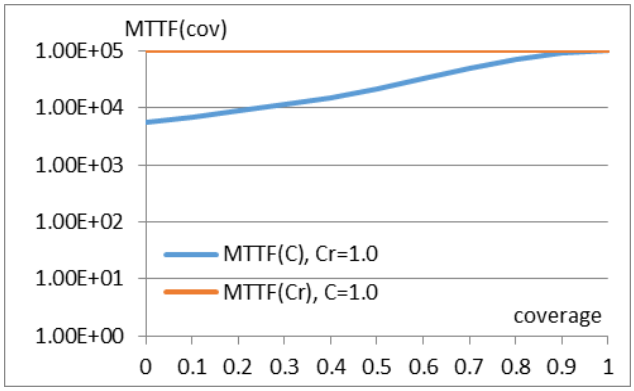
2	0	1	Нормално състояние
0	2	1	Стоп
1	2	0	Стоп
1	0	2	Отказ
0	1	2	Отказ
1	1	1	Стоп
0	0	3	Отказ
0	3	0	Стоп

Триплираният компонент на отказоустойчивата разпределена система също е изследван за система с и без локален ремонт. На *Фигура 3-13 – Фигура 3-20* са представени надеждностните характеристики на компонента за система с локален ремонт.

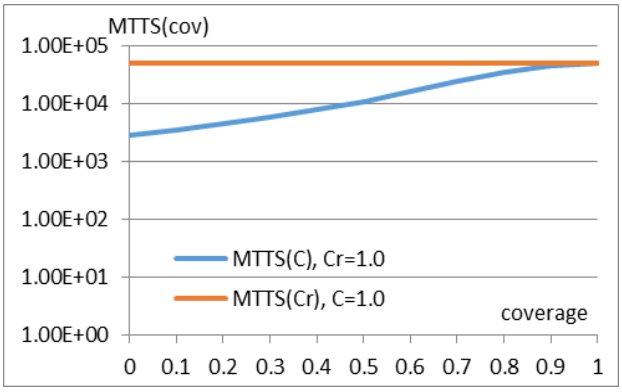
Коефициентът на възстановяване след случайна неизправност влияе слабо върху повечето надеждностни показатели на триплирания компонент (*Фигура 3-13 – Фигура 3-16*). Той обаче води до намаляване на времето за престой и оттам на времената между отказите и на готовността на компонента (*Фигура 3-17 – Фигура 3-20*).



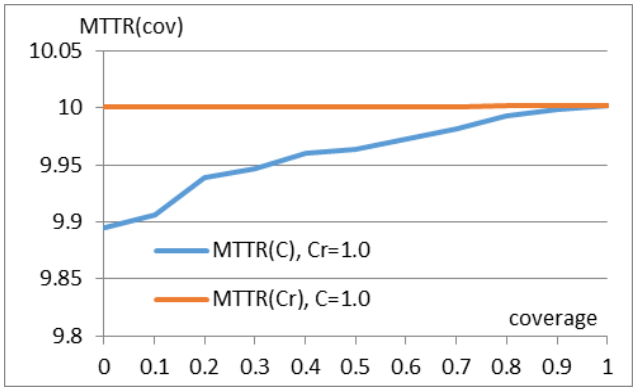
Фигура 3-13. Надеждност на триплиран компонент на система с локален ремонт



Фигура 3-14. MTTF на триплиран компонент на система с локален ремонт



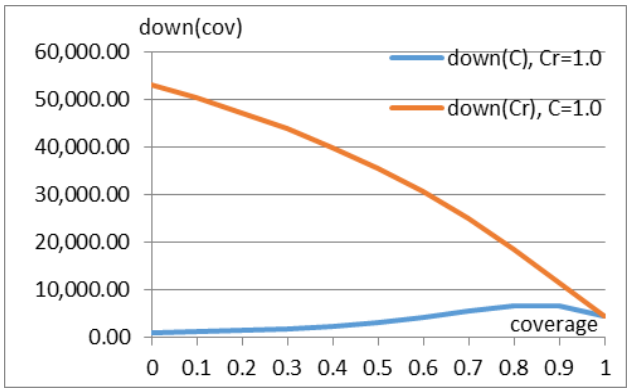
Фигура 3-15. MTTS на триплиран компонент на система с локален ремонт



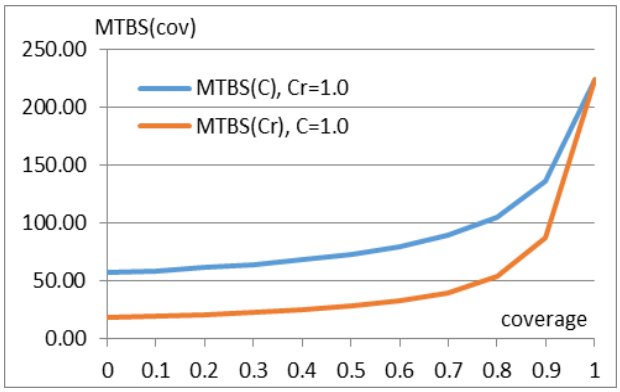
Фигура 3-16. MTTR на триплиран компонент на система с локален ремонт

Коефициентът на покритие на блока за самопроверка очаквано подобрява надеждността и времената до отказ и спиране (*Фигура 3-13 – Фигура 3-15*), както и

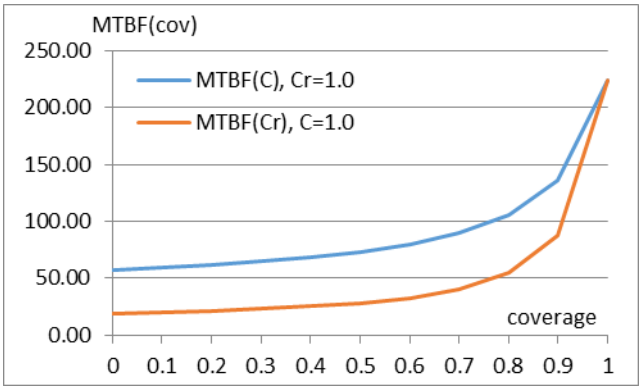
показателите, свързани с работоспособността на компонента (*Фигура 3-18 – Фигура 3-20*). Той обаче увеличава времето, прекарано от компонента в ремонти (*Фигура 3-17*).



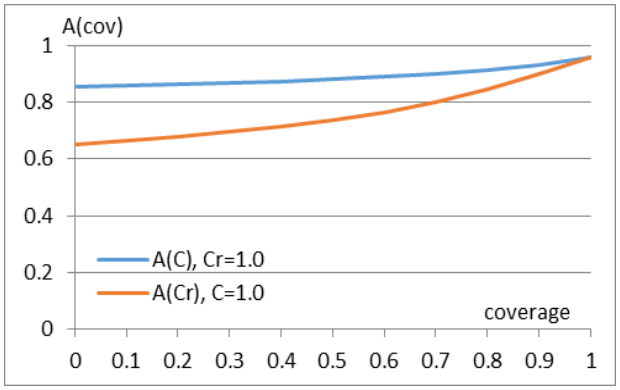
Фигура 3-17. Време за престой на триплиран компонент на система с локален ремонт



Фигура 3-18. MTBS на триплиран компонент на система с локален ремонт

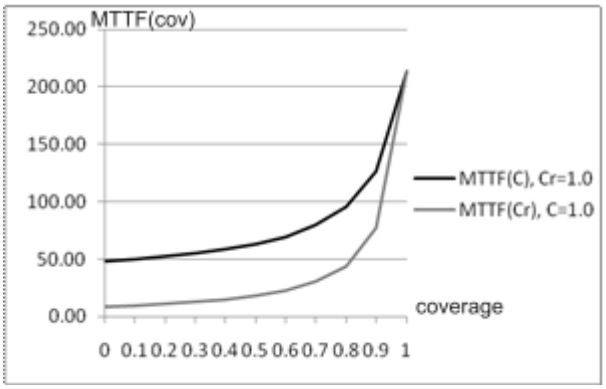


Фигура 3-19. MTBF на триплиран компонент на система с локален ремонт

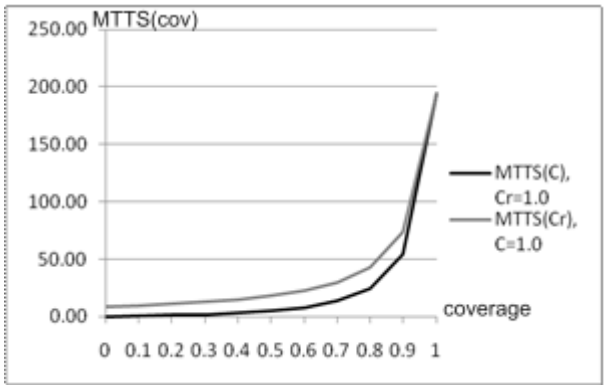


Фигура 3-20. Готовност на триплиран компонент на система с локален ремонт

Надеждностните характеристики на триплиран компонент на система без локален ремонт са показани на *Фигура 3-21* и *Фигура 3-22*. Вижда се, че при липса на локален ремонт системата има нужда от средства за самопроверка с високи C и C_r.



Фигура 3-21. MTTF на триплиран компонент на система без локален ремонт



Фигура 3-22. MTTS на триплиран компонент на система без локален ремонт

От направените изследвания на дублиран и триплиран компонент на отказоустойчива разпределена система може да се направи извод, че добавянето на блокове за самопроверка към всеки модул от компонента подобрява значително надеждностните характеристики на системата, особено когато тя работи без възможност за локален ремонт. Това дава основание да се очаква, че отказоустойчивостта на цялата система ще се подобри и изследваните показатели ще бъдат още по-високи.

3.2.2 Система с 10 компонента

Отказоустойчивата разпределена система с настройваема надеждност е моделирана първоначално за N=5 и M=10 [74]. Изследвано е влиянието на интензивността на локалните ремонти μ_p, интензивността на системните ремонти μ_{sys} и на коефициента на покритие C на средствата за самопроверка. При това изследване е прието, че средствата за самопроверка във всички компоненти имат еднакво покритие.

Резултатите за отказоустойчивата система с настройваема надеждност с N=5 показват, че коефициентът на покритие на средствата за самопроверка на компонентите подобрява системната надеждност [74]. Този ефект е двустранен. Ако бъде открит отказ на компонент, той може да бъде ремонтиран, дори ако системата не е ремонтируема. Ако бъдат открити отказите на компонент, системата преминава в стопово състояние, което е безопасно за обекта на управление. При системи с ремонт системата може да бъде ремонтирана след влизане в стопово състояние.

Промяната на степента на репликиране на компонентите също променя системната надеждност [74]. Наблюдават се случаи, в които се постига по-добра надеждност с използване на разпределение на структурния излишък, отколкото при еднакъв излишък за всички компоненти. Това означава, че системата може да има еднаква или по-висока надеждност, когато използва ресурсите си по различен начин.

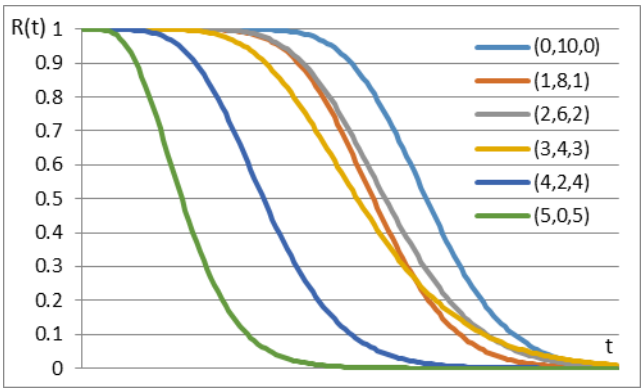
Затова изследванията на отказоустойчивата разпределена система с настройваема надеждност са продължени за N=10 и M=20 [75].

Системата има възможност за възстановяване от постоянна неизправност на компонент и за ремонт след системен отказ. Изследвани са 6 конфигурации на структурния излишък. Тяхната надеждност при C_1=0.88, C_2=0.94 и C_3=0.99 е показана на *Фигура 3-23*. Стойностите на коефициентите на покритие на компонентите с различна степен на репликиране, C_1, C_2 и C_3, са максимални за изследването. При тези условия най-висока надеждност има системата (0,10,0) (светло синята графика на *Фигура 3-23*). Тази система представлява известните системи, които работят с активен модулен излишък на два модула (изградени само с

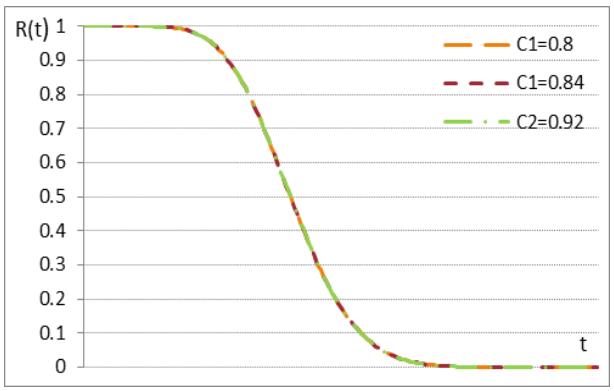
дублирани компоненти), т.е. без разпределение на структурния излишък. Най-ниска надеждност има система (5,0,5) (зелената графика на *Фигура 3-23*), която е изградена само от единични и триплирани компоненти. Системи (1,8,1) (оранжевата графика на *Фигура 3-23*) и (2,6,2) (сивата графика на *Фигура 3-23*) имат близки стойности на надеждността и поради това са представени само резултатите за система (2,6,2). Система (3,4,3) (жълтата графика на *Фигура 3-23*) се доближава по надеждност до системи (1,8,1) и (2,6,2). Интерес представляват системите (3,4,3), (2,6,2) и (0,10,0) поради сравнително по-високата си надеждност.

Система (0,10,0) е сравнена със системите, съдържащи компоненти с различен модулен излишък, но със същия общ брой модули, а именно системи (2,6,2) и (3,4,3). Системната надеждност е изследвана за различни стойности на коефициентите на покритие C_1, C_2 и C_3, като е определено влиянието на всеки един от тях, при постоянни стойности на останалите два.

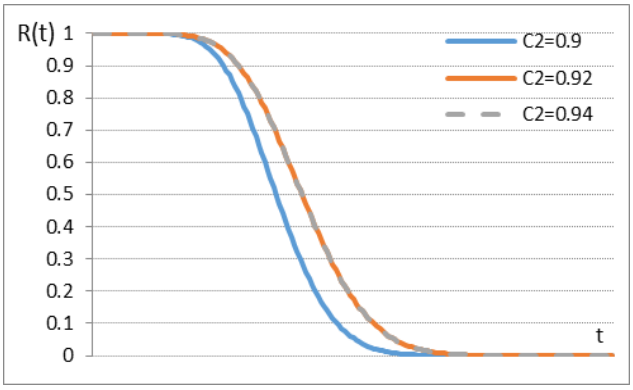
Надеждността на системата с три единични, четири дублирани и три триплирани компонента, (3,4,3), е изследвана, за да се проследи влиянието на коефициентите на покритие на компонентите (*Фигура 3-24* - *Фигура 3-26*). На *Фигура 3-24* е изобразено влиянието на коефициента на покритие на единичните компоненти. Вижда се, че C_1 не влияе въобще върху надеждността на системата (3,4,3) – графиките на надеждността за трите стойности на C_1 съвпадат (*Фигура 3-24*). Това е обяснимо предвид наличността на компоненти с по-високо покритие. От друга страна, C_1 не влошава надеждността на системата, въпреки че единичните компоненти са толкова, колкото триплираните.



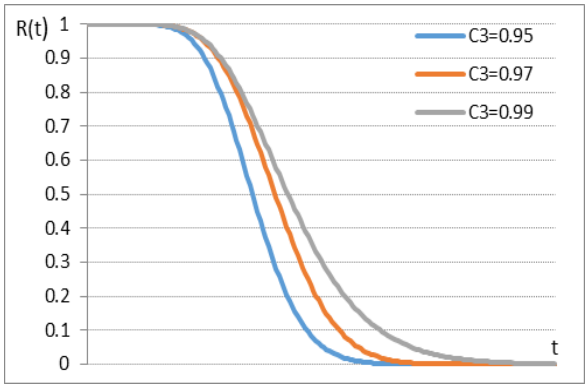
Фигура 3-23. Надеждност на система с 10 компонента за C_1=0.88, C_2=0.94 и C_3=0.99



Фигура 3-24. Надеждност на система (3,4,3) за различни стойности на C_1, при C2=0.94 и C3=0.97



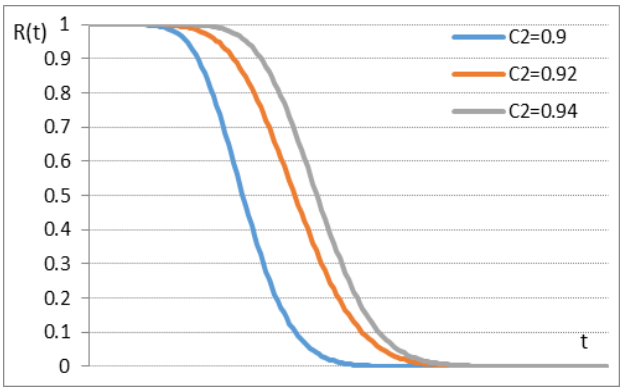
Фигура 3-25. Надеждност на система (3,4,3) за различни стойности на C_2, при C_1=0.88 и C_3=0.97



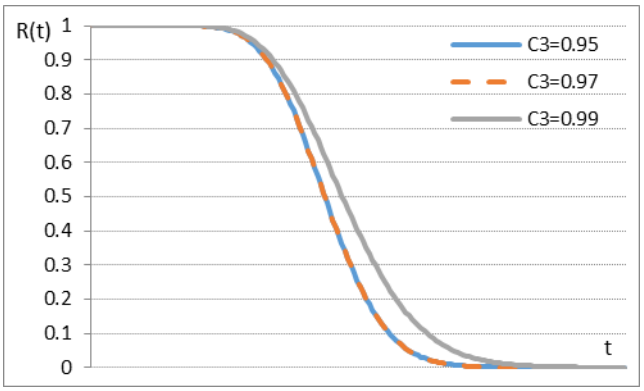
Фигура 3-26. Надеждност на система (3,4,3) за различни стойности на C_3, при C_1=0.88 и C_2=0.94

Коефициентите на покритие на дублираните компоненти C_2 (*Фигура 3-25*) и на триплираните компоненти C_3 (*Фигура 3-26*) увеличават надеждността. На *Фигура 3-25* кривите на надеждността за C_2=0.92 (в оранжево) и C_2=0.94 (в сиво) съвпадат. Влиянието на коефициента на покритие C_3 (*Фигура 3-26*) е значително, като подобрява надеждността на системата (3,4,3), когато се използват ТМИ компоненти с C_3=0.99.

Системата с два единични, шест дублирани и два триплирани компонента, (2,6,2), е по-силно повлияна от увеличаването на коефициента на покритие C_2 (*Фигура 3-28*), отколкото от увеличението на C_3 (*Фигура 3-27*). Това се обяснява с по-големия брой дублирани компоненти в сравнение с броя на триплираните компоненти. Резултатите за C_3=0.95 (синята графика на *Фигура 3-28*) и C_3=0.97 (оранжевата графика на *Фигура 3-28*) са почти еднакви. Само най-голямата стойност на C_3=0.99 води до подобряване на системната надеждност (сивата графика на *Фигура 3-28*). От друга страна, увеличаването на коефициента на покритие на модулите с ДМИ C_2 води до значително подобряване на надеждността (*Фигура 3-27*).



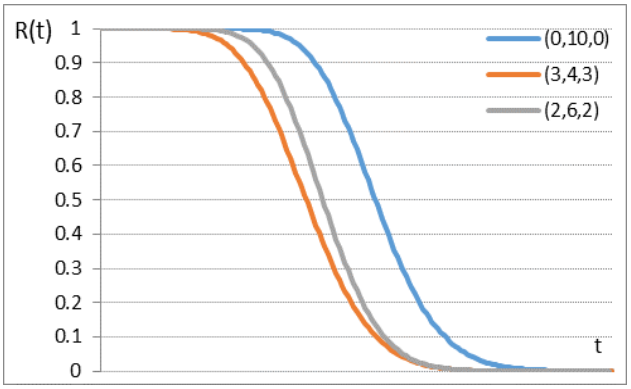
Фигура 3-27. Надеждност на система (2,6,2) за различни стойности на C_2, при C_1=0.88 и C_3=0.97



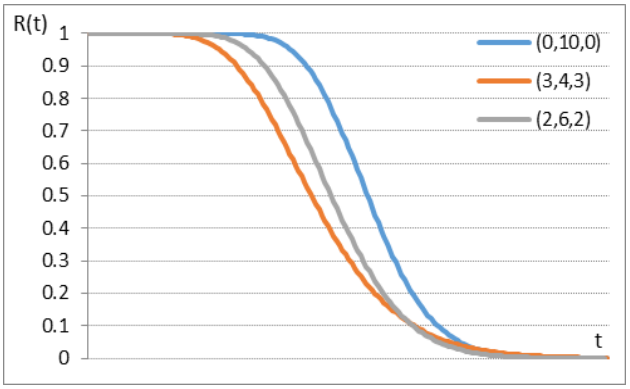
Фигура 3-28. Надеждност на система (2,6,2) за различни стойности на C_3, при C_1=0.88 и C_2=0.94

На *Фигура 3-29* и *Фигура 3-30* е направено сравнение на надеждността на системите (3,4,3), (2,6,2) и (0,10,0) при C_1=0.88 и C_2=0.94, а коефициентът C_3 е различен (C_3=0.97 на *Фигура 3-29* и C_3=0.99 на *Фигура 3-30*). Най-висока надеждност има системата, изградена изцяло от дублирани компоненти (0,10,0). Системата (3,4,3) е с най-ниска надеждност, която

се подобрява при увеличение на C_3 (*Фигура 3-30*, C_3=0.99). Системата (2,6,2) показва средна стойност на надеждността. Може да се направи изводът, че системите с преобладаващ брой дублирани компоненти, (2,6,2) и (0,10,0), постигат по-висока системна надеждност.



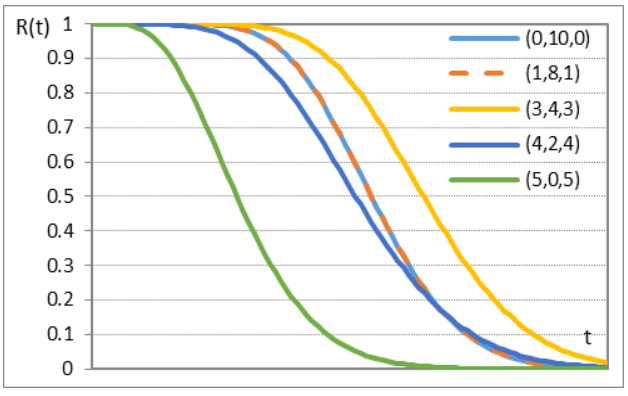
Фигура 3-29. Надеждност на системите (3,4,3), (2,6,2) и (0,10,0) при C_1=0.88, C_2=0.94 и C_3=0.97



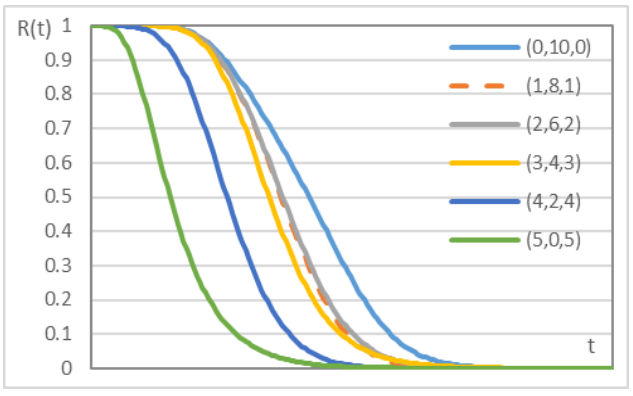
Фигура 3-30. Надеждност на системите (3,4,3), (2,6,2) и (0,10,0) при C_1=0.88, C_2=0.94 и C_3=0.99

При сравнение на системи с различно разпределение на структурния излишък за по-малък коефициент на покритие C_2=0.9 (C_1=0.88 и C_3=0.97) (*Фигура 3-31*) подреждането на системите по надеждност се променя. Най-висока надеждност има системата (3,4,3) (жълтата крива на *Фигура 3-31*), следвана от системите (1,8,1) (оранжевата крива на *Фигура 3-31*) и (0,10,0) (светло синята крива на *Фигура 3-31*), които имат еднаква надеждност, и системите (4,2,4) (тъмно синята крива на *Фигура 3-31*) и (5,0,5), която има най-ниска надеждност (зелената крива на *Фигура 3-31*).

На *Фигура 3-32* е изобразена надеждността на всички конфигурации на системата с 10 компонента, когато коефициентите на покритие на компонентите се менят в зададените в т. 3.2 интервали. Тези резултати са разгледани по-подробно в т. 3.3.



Фигура 3-31. Сравнение на надеждността на системи с различен структурен излишък при C_1=0.88, C_2=0.9 и C_3=0.97



Фигура 3-32. Сравнение на надеждността на системи с различен структурен излишък при $C_1 \in [0.8, 0.9)$, $C_2 \in [0.9, 0.95)$ и $C_3 \in [0.95,\ 1.0)$

В *Таблица 3-2* са посочени надеждностните характеристики на изследваните системи с N=10, M=20, C_1=0.88, C_2=0.9 и C_3=0.97 (*Фигура 3-31*). Потвърждава се, че системите (0,10,0),

(1,8,1) и (2,6,2) имат еднакви характеристики, което означава, че при тези коефициенти на покритие добавянето на няколко единични и триплирани модули не променя системната надеждност. Което може да се разглежда и като добра новина, защото включването на единични компоненти не влошава надеждността. При тези условия системата (3,4,3) има най-висока надеждност и най-добри надеждностни характеристики.

Таблица 3-2. Надеждностни характеристики на системите с различно разпределение на структурния излишък на компонентите при C_1=0.88, C_2=0.9 и C_3=0.97

	(0,10,0)	**(1,8,1)**	**(2,6,2)**	**(3,4,3)**	**(4,2,4)**	**(5,0,5)**
MTTF	53834.91	53834.91	53834.91	63673.97	51209.93	29134.64
MTTS	26713.11	26713.11	26713.11	36981.47	23942.41	17403.79
MTTR	12.88696	12.88696	12.88696	10.01339	14.85938	23.23656
downtime	12.88696	12.88696	12.88696	10.01339	14.85938	23.23656
MTBS	26713.11	26713.11	26713.11	36981.47	23942.41	17403.79
MTBF	53829.02	53829.02	53829.02	63651.17	51205.34	29133.34
A	0.999761	0.999761	0.999761	0.999843	0.99971	0.999203
stops	0.0002	0.0002	0.0002	0.0005	0.0001	0.00005

От *Фигура 3-31* и *Таблица 3-2* се вижда, че надеждността е чувствителна към промени в коефициента на покритие на компонентите с ДМИ. Участието на компоненти с ТМИ не винаги допринася за увеличаване на надеждността. Обяснението може да се търси в ниското покритие на единичните компоненти, което не може да бъде компенсирано от триплираните компоненти. Това се потвърждава и от надеждността на системата (5,0,5) (*Фигура 3-31*), която има само единични и триплирани компоненти, и показва най-ниска надеждност между изследваните системи.

Поради забелязаната чувствителност на надеждността към промени в коефициента на покритие са проведени изследвания с произволни стойности на коефициентите на покритие в рамките на зададените интервали.

Както беше отбелязано в т. 3.2, в зависимост от средата на функциониране на разглежданите ОРС коефициентите на покритие на техните компоненти се менят в рамките на определени интервали. В представените на *Фигура 3-23 – Фигура 3-31* резултати стойностите на коефициентите на покритие C_1, C_2 и C_3 се избират в симулационната програма стъпково в съответния интервал, за да се провери влиянието на покритието върху системната надеждност. Тези резултати показват, че разпределението на структурния излишък в компонентите няма еднозначно влияние върху системната надеждност. Затова бяха проведени изследвания, при които коефициентите на покритие за компонентите представляват случайни стойности в съответните интервали $C_1 \in [0.8, 0.9)$, $C_2 \in [0.9, 0.95)$ и $C_3 \in [0.95,\ 1.0)$. На *Фигура 3-32* се

вижда, че най-висока надеждност има системата без настройване на надеждността, (0,10,0) (светло синята крива), следвана от системите (1,8,1) (оранжевата крива) и (2,6,2) (сивата крива). Близка, но по-ниска, надеждност има системата (3,4,3) (жълтата крива). Тези резултати са сходни с надеждността на системите от *Фигура 3-23*, но се различават от резултатите на *Фигура 3-31*.

Симулационното моделиране на отказоустойчивата система с настройваема надеждност с 10 компонента показва, че системата има добри надеждностни характеристики. Участието на единични компоненти не влошава значително нейната надеждност, а включването на триплирани компоненти води до увеличаване на общата надеждност. Изследваните варианти на разпределение на структурния излишък (системи (3,4,3) и (2,6,2)) показват надеждност, която е сравнима с тази на система без настройване на надеждността – система (0,10,0).

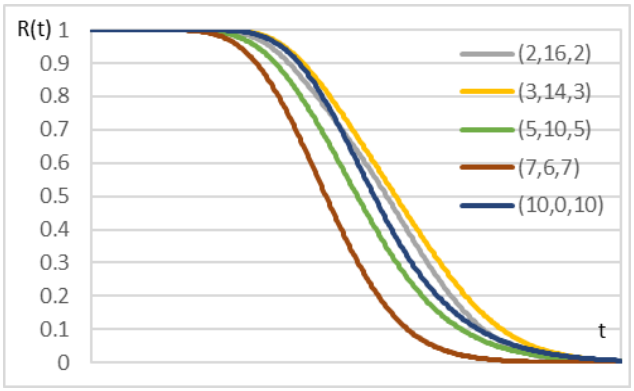
Резултатите от изследването на система с 10 компонента показват, че има разпределения на структурния излишък, които подобряват системната надеждност при определени условия. За да се провери дали това се потвърждава за повече компоненти, е изследвана система с 20 компонента и 40 модула.

3.2.3 Система с 20 компонента

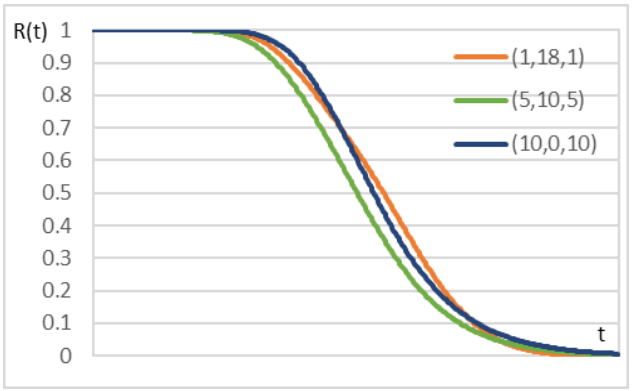
Отказоустойчивата система с настройваема надеждност с 20 компонента и 40 модула е симулирана при същите параметри като системата с 10 компонента (т. 3.2.2). Изследвани са 11 разпределения на модулния излишък в компонентите. Влиянието на постоянните неизправности е разгледано при две различни интензивности на неизправностите, за да се изследват различни условия на работната среда на системата. Интензивността на постоянните неизправности представлява честотата, с която системата понася постоянни неизправности, които са резултат както от вътрешни, така и от външни причини. Външните причини са свързани със средата на приложение (контекста на системата, разгледан в *Глава 1*). Избраните интензивности на неизправностите са илюстративни. Те представляват среди с различно въздействие върху системната гарантоспособност. Допускането е, че системите, които функционират при по-неблагоприятни условия, търпят повече неизправности, което в модела се изразява с по-голяма интензивност на постоянните неизправности $\lambda_p=10^{-3}$ 1/h. По-малката интензивност на неизправностите $\lambda_p=10^{-4}$ 1/h моделира среди, където неизправностите настъпват по-рядко, но системата трябва да е в състояние да ги толерира.

Системата с настройваема надеждност има различни разпределения на излишъка в компонентите. Тяхната надеждност, R_d, във функция от времето е показана на *Фигура 3-33*,

където интензивността на постоянните неизправности е $\lambda_p=10^{-4}$ 1/h. С цел по-голяма яснота на изобразяването на фигурата са показани само пет от единадесетте възможни системи. Надеждността на не изобразените графики попада между двете крайни системи – система (7,6,7) с най-малка надеждност (кафявата крива на *Фигура 3-33*) и система (3,14,3) с най-голяма надеждност (жълтата крива на *Фигура 3-33*). Има системи с много близки графики на надеждността и затова само една от тях е показана на *Фигура 3-33*.



Фигура 3-33. Надеждност на системи с различни разпределения на структурния излишък, $\lambda_p=10^{-4}$ 1/h, $\mu_p=\mu_{sys}=0.1$ 1/h



Фигура 3-34. Надеждност на системи с различен брой триплирани компоненти, $\lambda_p=10^{-4}$ 1/h, $\mu_p=\mu_{sys}=0.1$ 1/h

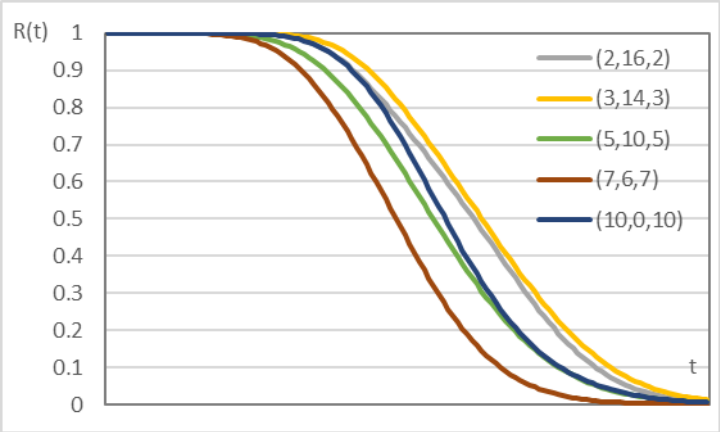
Системите със сравнително малък брой единични и триплирани компоненти демонстрират висока надеждност (системи (3,14,3) в жълто и (2,16,2) в сиво на *Фигура 3-33*). Не може да се твърди обаче, че това е тенденция. Системата, състояща се само от единични и триплирани компоненти (система (10,0,10) в тъмно синьо на *Фигура 3-33*) има близка надеждност.

За да се покаже влиянието на триплираните компоненти, на *Фигура 3-34* е изобразена надеждността на системи (1,18,1), (5,10,5) и (10,0,10). Въвеждането на повече единични и триплирани компоненти в системата (система (10,0,10) в тъмно синьо на *Фигура 3-34*) подобрява надеждността и удължава периода, през който системата поддържа висока надеждност. Функционирането с по-малко единични и ТМИ компоненти (система (1,18,1) в оранжево на *Фигура 3-34*) обаче не влошава значително системната надеждност. Система (5,10,5), която има най-ниската надеждност от трите системи на *Фигура 3-34*, все пак има добра надеждност в сравнение с останалите системи, както се вижда от *Фигура 3-33*.

Не може да се изведе ясна зависимост между разпределението на структурния излишък и надеждността (*Фигура 3-33* и *Фигура 3-34*). Разпределението на излишъка влияе върху системната надеждност и някои разпределения са по-благоприятни за системата от други. Изборът на системна конфигурация в зависимост от изискванията на приложението може да доведе до подобряване на надеждността и готовността на системата.

Системите са изследвани по-задълбочено в т. 3.3, където е представен подходът на настройваема надеждност.

Отказоустойчивата система с настройваема надеждност е симулирана за по-голяма интензивност на постоянните неизправности, за да се провери дали разпределението на излишъка влияе по различен начин върху нейната надеждност (*Фигура 3-35*). В сравнение с надеждността на системи при $\lambda_p=10^{-4}$ 1/h (*Фигура 3-33*) надеждността при $\lambda_p=10^{-3}$ 1/h е подобна и изследваните системи са подредени по същия начин според тяхната надеждност (*Фигура 3-35*). Настройването на системната надеждност чрез промяна на разпределението на структурния излишък не зависи от интензивността на неизправностите.



Фигура 3-35. Надеждност на системи с различно разпределение на структурния излишък, $\lambda_p=10^{-3}$ 1/h, $\mu_p=\mu_{sys}=0.1$ 1/h

3.3 Подход на настройваема надеждност

Критичните за безопасността приложения изискват много висока надеждност, за да предоставят гарантоспособната услуга, за която са предназначени. При проектирането им се задава желаната системна надеждност и останалите ѝ характеристики се съобразяват с това изискване. Изследването на отказоустойчивата система с настройваема надеждност е разширено, за да може в повече дълбочина да се проследи нейното поведение и да се предложат възможности за постигане на висока желана надеждност. В представения дисертационен труд тази надеждност се нарича *обща надеждност* и се означава с R_{total}. Въз основа на изследванията, представени в т. 3.2.2 и 3.2.3 и [75], [76], е разработен подход на настройваема надеждност за определяне на системите, които постигат R_{total}.

R_{total} се определя при задаването на спецификациите на проектираната система според изискванията на приложението (на етапа на проектиране на системата, при определяне на системните изисквания – *Глава 1*, т. 1.3.1). По време на проектирането се дефинират режимите на неизправност и отказ, коефициентите на покритие на средствата за самопроверка, средните

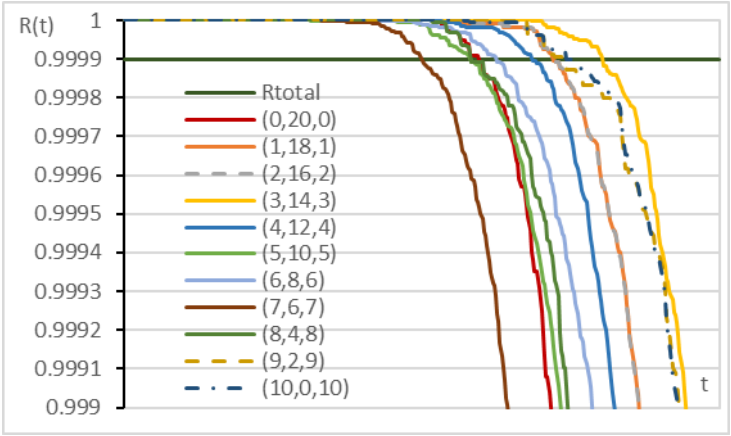
времена до отказ и до ремонт и т.н. (*Глава 2* и *Приложение А*). За отказоустойчивата система с настройваема надеждност се определят: интензивност на постоянните неизправности, интензивност на локалните ремонти, интензивност на системните ремонти, време на живот/мисия и обща надеждност. При дадена R_{total} се определят и изследват всички възможни разпределения на модулния излишък, *система (i, j, k)*, за N компонента и M модула. Системите (i, j, k), т.е. системните конфигурации, се симулират, както е описано в т. 3.1, и се получават техните графики на надеждността във функция от времето. Надеждността на всяка конфигурация R_d се сравнява с R_{total}. След това се определят системите с $R_d(t) \geq R_{total}$. За всяка система се определя и периодът на висока надеждност.

Резултатите за системите, описани в т. 3.2.2 и 3.2.3, са показани на *Фигура 3-36* за R_{total}=0.9999 за система с N=20. Всички изследвани разпределения на структурния излишък постигат R_{total}, но поддържат тази надеждност за различни периоди. (Данните са за 1000 итерации за изчисляване на надеждността на всяка от единадесетте системи със стъпка 100.)

Стойността R_{total}=0.9999 е избрана само за целите на симулацията. Действителните изисквани стойности за високонадеждни системи са много по-високи (например от порядъка на $1\text{-}10^{-9}$ [17]. Резултатите от симулирането показват взаимоотношението между общата надеждност и разпределението на структурния излишък и илюстрират подхода на настройваема надеждност.

Система (3,14,3) има най-висока надеждност (жълтата крива на *Фигура 3-36*), следвана от система (10,0,10) (тъмно синята крива на *Фигура 3-36*) и система (1,18,1) (оранжевата крива на *Фигура 3-36*). Системата, изградена само от дублирани компоненти, система (0,20,0), има най-ниска надеждност (червената крива на *Фигура 3-36*).

Разпределението на излишъка в компонентите на системата влияе върху системната надеждност (*Фигура 3-33, Фигура 3-35* и *Фигура 3-36*). Общата надеждност е по-висока за някои разпределения на структурния излишък, например система (3,14,3) (жълтата крива на *Фигура 3-36*), система (1,18,1) (оранжевата крива) и система (10,0,10) (тъмно синята крива), и е по-ниска за други, като система (7,6,7) (кафявата крива), система (5,10,5) (светло зелената крива) и (6,8,6) (светло синята крива на *Фигура 3-36*).



Фигура 3-36. Постигане на надеждност R_{total}=0.9999, $\lambda_p=10^{-4}$ 1/h, $\mu_p=\mu_{sys}$=0.1 1/h

Не може да се изведе ясна зависимост между разпределението на излишъка и системната надеждност. Ако броят на компонентите с коефициент на покритие C_3 (т.е. с ТМИ) е по-голям от броя на компонентите с коефициент на покритие C_2 (т.е. с ДМИ), това не означава непременно, че системната надеждност ще се повиши. Въвеждането на единични компоненти, от друга страна, не води до значително намаляване на системната надеждност. В някои случаи, например система (10,0,10) (тъмно синята крива на *Фигура 3-36*), надеждността е по-добра отколкото при системи с по-малък брой единични компоненти, като система (6,8,6) (светло синята крива на *Фигура 3-36*).

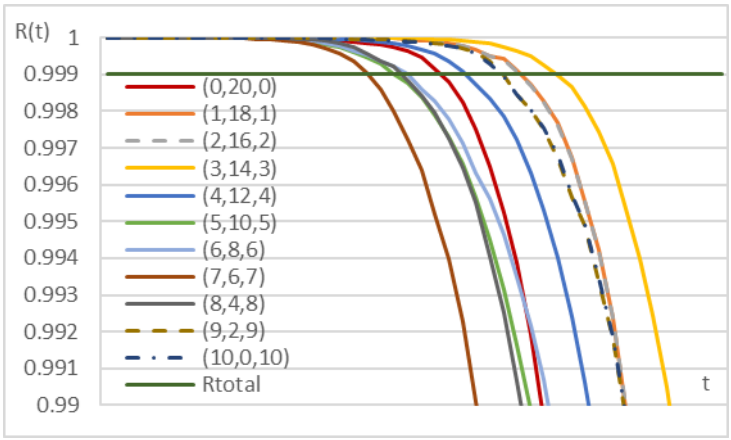
Таблица 3-3 показва периодите, през които надеждността R_d на всяка от системите с настройваема надеждност надвишава R_{total}=0.9999, като се започва от системата с най-дълъг период. Системата (3,14,3) поддържа своята надеждност над R_{total} за най-дълъг период от време в сравнение с останалите системи. Система (0,20,0) без настройваема надеждност има сравнително по-кратък период на надеждност над R_{total}=0.9999.

Таблица 3-3. Периоди на работа на системи (i, j, k) с надеждност $R_d \geq R_{total}$=0.9999, $\lambda_p=10^{-4}$ 1/h, $\mu_p=\mu_{sys}$=0.1 1/h, N=20

Система	$R_d \geq R_{total}$	Време [h]
(3,14,3)	0.999902	32300
(9,2,9)	0.999904	29800
(10,0,10)	0.999922	29800
(1,18,1)	0.999905	28900
(2,16,2)	0.999909	28900
(4,12,4)	0.999902	27500
(6,8,6)	0.999907	25100
(0,20,0)	0.9999	24000
(8,4,8)	0.999912	23400
(5,10,5)	0.999902	23400
(7,6,7)	0.999903	20100

Подобрената системна надеждност за съответните разпределения на структурния излишък води до по-дълъг период на функциониране на системата с надеждност над R_{total} (*Таблица 3-3*).

При симулиране на системите за $\lambda_p=10^{-3}$ 1/h (*Фигура 3-37*) подреждането на графиките на надеждността на различните конфигурации на системата е приблизително същото като при интензивност на неизправностите $\lambda_p=10^{-4}$ 1/h (*Фигура 3-36*). Отново най-висока надеждност има система (3,14,3) (жълтата крива на *Фигура 3-37*), а най-ниска – система (7,6,7) (кафявата крива на *Фигура 3-37*). Системата без настройваема надеждност (0,20,0) (червената крива на *Фигура 3-37*) показва средна надеждност.



Фигура 3-37. Постигане на надеждност $R_{total}=0.999$, $\lambda_p=10^{-3}$ 1/h, $\mu_p=\mu_{sys}=0.1$ 1/h

Периодите на работа на конфигурациите с надеждност $R_{total}\geq0.999$ за $\lambda_p=10^{-3}$ 1/h са показани в *Таблица 3-4*. Както личи и от графиките на *Фигура 3-37*, система (3,14,3) най-дълго поддържа желаната надеждност R_{total}, а най-кратък е периодът за система (7,6,7). Система (0,20,0) без настройваема надеждност има сравнително по-кратък период на поддържане на R_{total} в сравнение с повечето от останалите системи.

Таблица 3-4. Периоди на работа на системи *(i, j, k)* с надеждност $R_d\geq R_{total}=0.999$, $\lambda_p=10^{-3}$ 1/h, $\mu_p=\mu_{sys}=0.1$ 1/h, $N=20$

Система	$R_d\geq R_{total}$	Време [h]
(3,14,3)	0.999242	3300
(1,18,1)	0.9991	3100
(2,16,2)	0.99908	3100
(9,2,9)	0.999357	2900
(10,0,10)	0.999295	2900
(4,12,4)	0.999098	2700
(0,20,0)	0.99914	2500
(6,8,6)	0.999004	2300
(5,10,5)	0.999022	2200
(8,4,8)	0.999212	2200
(7,6,7)	0.999035	2000

Резултатите от симулирането показват, че разпределянето на системните ресурси в зависимост от тяхната важност за приложението може да даде предимство за системната надеждност. Например, един съвременен автомобил е оборудван с реалновременни разпределени системи, които управляват различни блокове, като двигател, окачване, скоростна кутия, врати, седалки и др. Тези блокове на свой ред са подсистеми, състоящи се от други модули, които работят заедно в изпълнение на конкретна задача. Една такава подсистема може да използва подхода на настройваема надеждност, за да постигне надеждността, диктувана от приложението. Определяйки R_{total} и знаейки броя на компонентите в подсистемата и техните надеждностни характеристики, могат да бъдат изведени и симулирани разпределенията на структурния излишък, за да се сравни тяхната надеждност. По този начин може по-нататък да се изследва и разработи модулното разпределение с най-висока надеждност, отчитайки особеностите на проектираната подсистема.

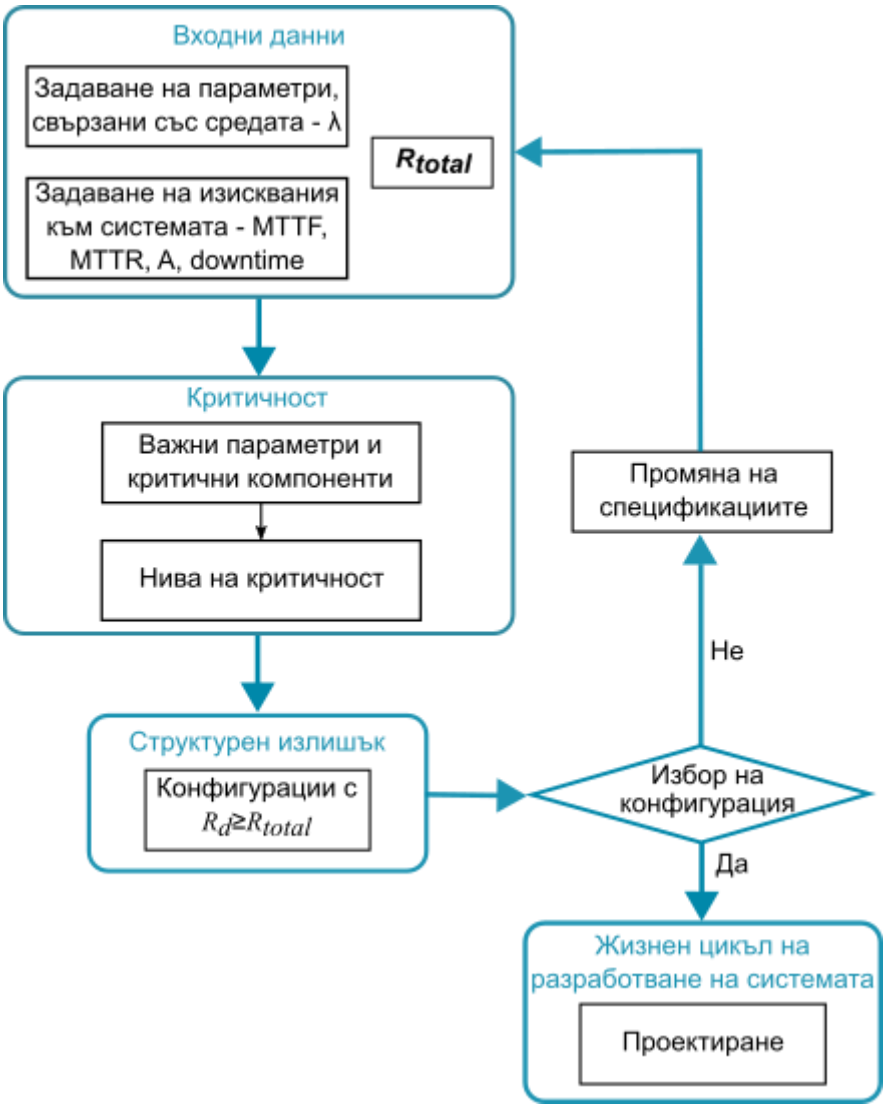
Представените резултати от симулационното моделиране (т. 3.2) показват, че има разпределения на структурния излишък, при които системата с настройваема надеждност постига по-висока надеждност R_{total} и я поддържа за по-дълъг период от време в сравнение със система без настройване на надеждността (*Фигура 3-36* и *Таблица 3-3, Фигура 3-37* и *Таблица 3-4*). При какви условия се случва това обаче е трудно да се установи, тъй като няма ясна зависимост между системната надеждност и разпределението на излишъка. Затова в дисертационния труд се предлага *подход на настройваема надеждност*, чрез който да се определят конфигурациите на структурния излишък, удовлетворяващи изискването за системна надеждност на приложението.

Подходът на настройваема надеждност може да се опише със следните действия:

1. Задаване на R_{total},
2. Задаване на параметрите, описващи средата – интензивност на неизправностите,
3. Задаване на изискванията към системата – средно време до отказ, средно време до ремонт, готовност, средно време на престой, възможности за локален и системен ремонт и съответно интензивности на локалните и системните ремонти,
4. Определяне на важните контролирани параметри и на критичните компоненти,
5. Определяне на нивата на критичност,
6. Извеждане на системните конфигурации, при които настройваемата надеждност е равна или по-голяма от изискваната обща надеждност R_{total},
7. Проверка коя от възможните конфигурации отговаря най-добре на изискванията на приложението,

8. При липса на подходяща конфигурация се променят входните спецификации и процедурата се повтаря.

Описаните действия са онагледени на *Фигура 3-38*. Част от тях могат да се изпълняват едновременно. При обсъждането на системните спецификации в процеса на проектиране на системата се определят параметрите на средата, т.е. видовете хардуерни неизправности и тяхната интензивност, както и изискванията за гарантоспособност към самата система - R_{total}, средни времена до отказ и ремонт, готовност и т.н. Важните контролирани параметри описват особеностите на обекта на управление, а критичните компоненти се избират в зависимост от тежестта на последствията от техен отказ. На тях се присвояват съответните нива на критичност.



Фигура 3-38. Описание на подхода на настройваема надеждност

В зависимост от броя на компонентите и модулите на системата се определят разпределенията на структурния излишък. Те се симулират, за да се получат надеждностните им характеристики. Надеждността им се съпоставя с R_{total}. Ако повече от възможните конфигурации на системата с настройваема надеждност изпълняват изискването за системна надеждност при зададените спецификации, подходящата за приложението конфигурация може да се избере според допълнителни критерии, като брой единични, дублирани или триплирани компоненти, желани нива на критичност, най-висока обща надеждност, най-голям период с висока обща надеждност и т.н. Избраната конфигурация след това преминава към фазата на проектиране от жизнения цикъл на разработване на системата.

Ако нито една конфигурация на системата с настройваема надеждност не удовлетворява изискването за R_{total} на приложението, е необходимо да се преразгледат системните спецификации, включително и изискването за обща надеждност.

3.4 Изводи и резултати

В *Глава 3* е представена симулационната програма, разработена според изискванията в т. 3.1.1. Представени са основната структура и блоковата схема на програмата, която е описана в *Приложение В*.

Изследван е компонент на отказоустойчивата система с настройваема надеждност (т. 3.2.1) в зависимост от коефициента на покритие на блока за самопроверка C и коефициента на възстановяване от случайна неизправност C_r. Симулирани са компоненти с двоен и троен модулен излишък при работа с и без локален ремонт. Оценено е влиянието на C и C_r върху надеждността на компонентите, тяхната готовност, *MTTF, MTTS, MTBF, MTBS* и средното време за престой.

Според предвидения изследователски протокол са проведени експерименти (т. 3.2.2 и т. 3.2.3) за системи с различен брой компоненти (N=10 и N=20), различна интензивност на постоянните неизправности λ_p и различни стойности на коефициентите на покритие C_1, C_2 и C_3. Получени са данни за: $R(t)$, *A, MTTF, MTTR, MTTS, downtime*. Определени са възможните разпределения на структурния излишък при зададените N и M.

Изследвано е влиянието на броя компоненти, коефициентите на покритие и интензивността на неизправностите върху надеждностните характеристики на отказоустойчивата система с настройваема надеждност и на системи без разпределение на структурния излишък. Определени са конфигурациите на системата, удовлетворяващи изискването за обща надеждност.

Създаден и представен е авторски подход на настройваема надеждност (т. 3.3), чрез който да се определя кои разпределения на излишъка в компонентите удовлетворяват желаната обща системна надеждност. Въз основа на изискванията на приложението подходът намира системните конфигурации, които могат да постигнат общата надеждност.

Налага се изводът, че може да се постигне висока надеждност чрез разпределение на структурния излишък, като в някои случаи тя надвишава надеждността на системи без разпределение на излишъка. Конфигурациите, при които това е изпълнено, могат да бъдат определени чрез подхода на настройваема надеждност.

Постигнатите резултати имат научно-приложен и приложен характер. Научно-приложните резултати са:

1. Разработена е симулационна програма, реализираща метода за симулационно моделиране на гарантоспособни разпределени системи;
2. Формулиран е подход на настройваема надеждност.

Приложните резултати са свързани с провеждане на експерименти със симулационната програма с цел да се изследват надеждностните характеристики на отказоустойчива разпределена система с и без настройваема надеждност.

Чрез изследванията, представени в *Глава 3*, са изпълнени задачи 3 и 4 на дисертацията.

Част от резултатите в тази глава са публикувани в:

1. Djambazova, E. (2022). Achieving system reliability using reliability adjustment. International Conference on Computer Systems and Technologies 2022 (CompSysTech ’22), Ruse, Bulgaria. ACM, New York, NY, USA, pp. 64-68. DOI: 10.1145/3546118.3546129.
2. Djambazova, E. (2012). Adjusting reliability of a fault-tolerant distributed process control system – Preliminary results. International Conference “Automatics and Informatics 2012”, Sofia, Bulgaria, pp. 175-178.
3. Djambazova, E. (2009). Node reliability of a fault-tolerant distributed process control system – Simulation results. International Conference “Automatics and Informatics’ 2009”, Sofia, Bulgaria, pp. I-131 – I-134.

Глава 4 Обсъждане и анализ на резултатите

Гарантоспособните разпределени системи, които са обект на дисертацията, се разработват за приложения, които са критични по отношение на безопасността. Те се изграждат със специализирани или с готови компоненти, като и двата подхода имат своите предимства и недостатъци. Специализираните компоненти отразяват по-адекватно особеностите на приложението, дават възможност за постигане на висока надеждност с методи и техники, съобразени с конкретната работна среда и контекст на системата и са добре верифицирани и валидирани. Това обаче изисква време и усилия, които имат своята цена. Такива системи се проектират и внедряват по-бавно, което, от една страна, ги прави твърде трудни за адаптиране към промени в средата на функциониране на системата и, от друга страна, оскъпява крайната им реализация.

Прилагането на готови компоненти скъсява периода между проектиране и внедряване на разпределената система и намалява цената. Той има недостатък, че внася допълнителни рискове за надеждността на системата. Готовите компоненти не дават гаранции за изпълнение на изискванията за отказоустойчивост. Затова се търсят методи и средства за компенсиране на ниската собствена надеждност на готовите компоненти.

В посока на постигане на повече гъвкавост и съобразяване с изискванията на приложението са разработените системи, които внасят гарантоспособност на ниво мидълуер, и системи, въвеждащи нива на критичност на компонентите.

Анализът на гарантоспособните разпределени системи от гледна точка на структурния излишък, направен в *Глава 1*, показва, че основните подходи за постигане на повече гъвкавост по отношение на изискванията на приложението са промяна на софтуерния структурен излишък (при статично разпределение на хардуерния структурен излишък) и въвеждане на нива на критичност.

Отказоустойчивата разпределена система с настройваема надеждност предлага и изследва подход, който да притежава гъвкавостта на гарантоспособните системи с готови компоненти, да отчита нивата на критичност на системите със смесена критичност, да дава възможности за настройване на надеждностни характеристики и в същото време да отговаря на изискванията за висока надеждност. Всички изброени видове системи реализират отказоустойчивостта си посредством структурен излишък. Те прилагат равномерно разпределен хардуерен излишък на компонентите и съобразяват с приложението разпределението на софтуерния структурен излишък. Предложената система с настройваема надеждност внася гъвкавост чрез разпределение на хардуерния структурен излишък.

При изграждане на идеята за ОРС с настройваема надеждност са взети под внимание концептуалният модел на подход за вземане на решения във връзка с гарантоспособността и класификацията на гарантоспособните разпределени системи според възможностите им за определяне на структурния излишък в зависимост от приложението (*Глава 1*, *Фигура 1-3* и *Фигура 1-4*). Предложената система (*Глава 2*, *Фигура 2-1*) е изградена от отказоустойчиви компоненти с различна степен на репликиране, която дава възможност да бъдат изследвани надеждностните характеристики на различни конфигурации на структурния излишък. Отказоустойчивостта на компонентите се определя от наличието на блок за самопроверка на всеки модул и средства за сравнение на неговите резултати (*Глава 2, Фигура 2-2*). Представените Марковски вериги на ОРС с настройваема надеждност (*Глава 2*, т. 2.4) моделират поведението на системата при отсъствие или наличие на възможности за възстановяване на компонент и ремонт на системата. Тези модели стоят в основата на симулационната програма (*Приложение В*), чрез която са проведени експериментите в дисертацията. Избраният изследователски подход на симулационно моделиране предлага възможност за представяне на поведението на системата по отношение на неизправностите и отказите и изследване на нейните надеждностни характеристики, дефинирани в *Глава 2*, т. 2.1.2. Симулацията позволява моделиране на система с много компоненти и състояния и задаване на различни параметри за изследване на тяхното влияние върху системната надеждност.

Представените в *Глава 3* резултати от симулационното моделиране на отказоустойчивата система с настройваема надеждност показват, че изследваната система има добри надеждностни характеристики. Направен е анализ на резултатите, за да се провери дали се потвърждава хипотезата на дисертационния труд, а именно постига ли се висока надеждност и гъвкавост на системата чрез настройване на надеждността според изискванията на приложението.

При симулирането на компонент на системата (*Глава 3*, т. 3.2.1) са разгледани варианти на компонент с двоен и троен модулен излишък. Изследвано е влиянието на коефициента на покритие на блока за самопроверка и коефициента на възстановяване след случайна неизправност. Резултатите за надеждността, готовността, *MTTF, MTTR, MTTS, MTBF, MTBS* и времето за престой (*Глава 3*, *Фигура 3-3 – Фигура 3-22*) показват, че добавянето на блокове за самопроверка към всеки модул от компонента подобрява значително надеждностните характеристики на системата, особено когато тя работи без възможност за локален ремонт.

Симулационното моделиране на ОРС с настройваема надеждност с 10 компонента (*Глава 3*, т. 3.2.2, *Фигура 3-23 – Фигура 3-32*) не очертава ясна зависимост между системната

надеждност и разпределението на структурния излишък. От една страна, системата без настройваема надеждност (0,10,0) показва най-висока надеждност (*Глава 3*, т. 3.2.2, *Фигура 3-32*). От друга страна, при по-нисък коефициент на покритие C_2 някои конфигурации с разпределение на структурния излишък имат по-висока или близка до нейната надеждност (*Глава 3*, т. 3.2.2, *Фигура 3-31*). Влиянието на C_1 върху надеждността е незначително – повишаването му в рамките на изследвания интервал не води до промени в надеждността (*Глава 3*, т. 3.2.2, *Фигура 3-24*). Коефициентите на покритие C_2 и C_3 подобряват значително общата надеждност на системата (*Глава 3*, т. 3.2.2, *Фигура 3-25* - *Фигура 3-30*). Това е логично, предвид по-голямата им стойност.

Наблюдението, че системна конфигурация с настройваема надеждност показва по-висока надеждност от тази на система без разпределение на структурния излишък (*Глава 3*, т. 3.2.2, *Фигура 3-31*) дава основание да се задълбочат изследванията и да се провери как влияе структурният излишък върху надеждността при система с 20 компонента.

При ОРС с 20 компонента (*Глава 3*, т. 3.2.3) най-висока надеждност показва системата (3,14,3) (*Глава 3*, т. 3.2.3, *Фигура 3-33* - *Фигура 3-35*). Най-ниска надеждност има система (7,6,7) (*Глава 3*, т. 3.2.3, *Фигура 3-35*). Конфигурациите с малко единични и малко триплирани компоненти, като (1,8,1), (2,16,2) и (3,14,3), имат сравнително висока надеждност (*Глава 3*, т. 3.2.3, *Фигура 3-35*). Увеличаването на броя на триплираните компоненти може да повиши надеждността (*Глава 3*, т. 3.2.3, *Фигура 3-34*), но това не винаги е така. Системната конфигурация (7,6,7), например, има най-ниска надеждност (*Глава 3*, т. 3.2.3, *Фигура 3-35*).

При проведените симулационни изследвания за N=10 и N=20 различните конфигурации на системата с настройваема надеждност не показват постоянно поведение. Надеждността им се влияе от броя на компонентите, интензивността на постоянните неизправности, коефициентите на покритие на средствата за самопроверка. При някои конфигурации общата надеждност е по-висока от тази на система без настройваема надеждност, при други не е. За да се определи подходящата конфигурация на ОРС с настройваема надеждност според изискванията на приложението, е създаден подход на настройваема надеждност (*Глава 3*, т. 3.3). Той предвижда последователност от действия за избор на разпределение на структурния излишък в зависимост от изискването за обща системна надеждност на приложението. Подходът определя всички системни конфигурации, които постигат желаната надеждност, и представя възможност за избор на онази от тях, която в най-голяма степен удовлетворява зададените системни спецификации.

Промяната на степента на репликиране на компонентите води до промяна на системната надеждност (*Глава 3, Фигура 3-33, Фигура 3-35* и *Фигура 3-36*). Общата

надеждност е по-висока за някои разпределения на структурния излишък, например системи (3,14,3), (1,18,1) и (10,0,10), и е по-ниска за други, като системи (7,6,7), (5,10,5) и (6,8,6) (*Глава 3*, т. 3.3, *Фигура 3-36*). Не може да се изведе ясна зависимост между разпределението на излишъка и системната надеждност. Ако броят на компонентите с коефициент на покритие C_3 (т.е. с ТМИ) е по-голям от броя на компонентите с коефициент на покритие C_2 (т.е. с ДМИ), това не означава непременно, че системната надеждност ще се повиши. Въвеждането на единични компоненти, от друга страна, не води до значително намаляване на системната надеждност. В някои случаи, например система (10,0,10) (*Глава 3*, т. 3.3, *Фигура 3-36*), надеждността е по-добра отколкото при системи с по-малък брой единични компоненти, като система (6,8,6) (*Глава 3*, т. 3.3, *Фигура 3-36*).

Изследванията при по-висока интензивност на неизправностите $\lambda_p=10^{-3}$ 1/h (*Глава 3*, т. 3.3, *Фигура 3-37*) показват, че някои системи с настройваема надеждност са по-подходящи за работа при такива условия. Системи (1,18,1) и (2,16,2) имат втората по големина надеждност, докато при интензивност $\lambda_p=10^{-4}$ 1/h имат по-ниска надеждност (*Глава 3*, т. 3.3, *Фигура 3-36*). По-добре работи при висока интензивност и системата без настройваема надеждност (0,20,0) (*Глава 3*, т. 3.3, *Фигура 3-37*). Това поведение на изследваните системи предполага гъвкавост при избора на подходяща система за конкретно приложение.

Системите с най-висока обща надеждност имат и най-дълги периоди, през които я поддържат (*Глава 3*, т. 3.3, *Таблица 3-3* и *Таблица 3-4*). Това е мярка за тяхната готовност. Ако този показател е важен за приложението, той трябва да се вземе под внимание при избора на конфигурация с настройваема надеждност.

От проведените експерименти със симулационната програма могат да се направят няколко извода. Конфигурациите с настройваема надеждност постигат висока системна надеждност, съпоставима и в някои случаи по-висока от тази на системи без разпределение на структурния излишък. Подходът на настройваема надеждност дава възможност да се определят системните конфигурации, които най-добре изпълняват изискването за обща системна надеждност на приложението. Това внася гъвкавост при проектирането на отказоустойчиви разпределени системи да бъде избрано разпределение на структурния излишък, което най-добре отразява нуждите на конкретната реализация.

Тези изводи потвърждават хипотезата на дисертационния труд, че може да се постигне гъвкавост и висока надеждност на отказоустойчивите разпределени системи чрез разпределение на хардуерния структурен излишък според изискванията на приложението.

4.1 Изводи и резултати

Направено е обсъждане и анализ на резултатите, представени в дисертационния труд и на тази основа те са групирани, както следва:

Научни резултати:

1. Представен е нов архитектурен модел на отказоустойчива разпределена система с настройваема надеждност, като са дефинирани изискванията към нейните компоненти и системата като цяло;
2. Създаден е симулационен модел на предложената отказоустойчива разпределена система с настройваема надеждност;
3. Предложен е синтез на класификация и класификация на гарантоспособни разпределени системи със структурен излишък.

Научно-приложни резултати:

1. Създаден е концептуален модел на подход за вземане на решения при осигуряване на гарантоспособност;
2. Формулиран е подход на настройваема надеждност;
3. Разработена е симулационна програма, реализираща метода за симулационно моделиране на гарантоспособни разпределени системи.

Приложни резултати:

1. Създадената симулационна програма може да се използва за моделиране и изследване на надеждностните характеристики и на други отказоустойчиви системи. С нея може да се изследва влиянието и на случайни хардуерни неизправности.

Получените в дисертационното изследване резултати показват, че поставените задачи са изпълнени.

Приложните резултати доказват твърденията, поддържащи научната хипотеза на дисертацията: Може да се постигне висока надеждност и гъвкавост на разпределението на системните ресурси посредством настройваема надеждност, реализирана с разпределение на хардуерния структурен излишък. Това се доказва със следните резултати:

1. Представената отказоустойчива разпределена система с настройваема надеждност постига висока обща надеждност, съпоставима с надеждността на системи без разпределение на структурния излишък.
2. Отказоустойчивата разпределена система с настройваема надеждност има конфигурации на разпределението на структурния излишък, които притежават по-

добри надеждностни характеристики от тези на системи без разпределение на структурния излишък.

3. Подходът на настройваема надеждност дава възможност за определяне на конфигурациите на структурния излишък, които удовлетворяват изискването за обща надеждност на приложението, включително и тези с по-добра обща надеждност от системите без настройваема надеждност.

Заключение и бъдеща работа

В дисертационния труд са изследвани надеждностните характеристики на отказоустойчива разпределена система с настройваема надеждност. Системата е предложена като възможност за постигане на гъвкавост в проектирането на гарантоспособни разпределени системи. При направения обзор на гарантоспособни разпределени системи за работа в реално време са очертани две основни направления на изграждане на такива системи – с използване на специализирани компоненти и с използване на готови компоненти. Системите, изградени със специализирани компоненти, изискват повече усилия и време за проектиране и внедряване, дават по-малка възможност за повторно използване на вече създадени подсистеми, не са достатъчно гъвкави в разпределението на ресурсите и при добавяне на нови блокове към тяхната конфигурация. Техните предимства идват от предсказуемото им поведение. Те работят при строги времеви ограничения, което позволява ефективно използване на компонентите и изготвяне на разписания на задачите, прилагане на точни алгоритми за синхронизация и работа в режим „времеделене и множествен достъп“. От друга страна, системите, използващи готови компоненти, позволяват по-бързо разработване и внедряване на гарантоспособните разпределени системи, възможност за повторно използване на вече създадени компоненти и лесна разширимост на системата. Това предполага намаляване на разходите за проектиране и ускоряване на процеса на внедряване. Системите от този вид също постигат висока надеждност, като прилагат средства за самопроверка към готовите компоненти, разнообразни схеми на репликиране, разработване на подходящи интерфейси за връзка с обекта на управление и т.н. Те са по-гъвкави от системите със специализирани компоненти, но не винаги постигат тяхното предвидимо поведение.

Гарантоспособните разпределени системи и от двата вида постигат отказоустойчивост посредством разнообразни начини за въвеждане на излишък. Прилагат се структурен, времеви и функционален излишък. Структурният излишък добавя хардуерни и софтуерни елементи към системната архитектура. Най-често той се реализира като еднакво репликирани хардуерни компоненти, които изпълняват различно репликирани софтуерни задачи.

Резултатите могат да се обобщят по следния начин:

В дисертационния труд е направен е обзор на гарантоспособни разпределени системи за работа в реално време от гледна точка на управлението на въведения в тях излишък. Разработен е концептуален модел на подход за вземане на решение, направен е синтез на класификация на гарантоспособни разпределени системи на базата на модела за разработване на системи.

Предложени са архитектура и модел на отказоустойчива разпределена система за работа в реално време, наречена от автора система с настройваема надеждност. Тя предлага настройване на хардуерния структурен излишък за постигане на обща надеждност според изискванията на приложението. Системата е изследвана посредством метода на симулационно моделиране.

Проектиран и създаден е програмен продукт за симулационно моделиране на отказоустойчиви разпределени системи. Той реализира създадения модел на системата с настройваема надеждност, като моделира поведението на системите във времето при допускане за експоненциално разпределение и постоянна интензивност на неизправностите и в зависимост от коефициентите на покритие на техните средства за самопроверка. В резултат от изпълнението му се получава функция на надеждността, както и данни за надеждностните характеристики на системата. Симулационният продукт е написан на език за програмиране C++ и с него могат да се изследват отказоустойчиви системи с и без разпределение на структурния излишък.

Резултатите от проведените експерименти показват добра обща надеждност на системата с настройваема надеждност. Съществуват разпределения на структурния излишък, при които системната надеждност е по-висока от тази на система с равномерно разпределен излишък. При някои от конфигурациите се наблюдава стохастично подреждане, т.е. кривите на надеждността не се пресичат, което означава, че изборът на архитектурно решение не зависи от съответните стойности на коефициентите на покритие. Има случаи, при които различните архитектурни решения са неразличими, и други, при които не се наблюдава стохастично подреждане. Това прави избора на конкретно инженерно решение не очевиден и зависим от по-задълбоченото познаване на стойностите на коефициентите на покритие. Разликите във функцията на надеждността на изследваните конфигурации показват, че при проектиране на системата трябва да се използват средства, които да дадат количествена оценка на вариантите на разпределение на структурния излишък, за да бъде избрано най-подходящото за приложението решение.

Това мотивира и създаването от автора на подход, наречен подход на настройваема надеждност, който определя при какви конфигурации на структурния излишък системата постига надеждността, изисквана от приложението.

Получените при моделирането на отказоустойчивата разпределена система с настройваема надеждност резултати показват, че системата има предимства по отношение на разпределянето на структурния излишък според изискванията на приложението, които могат да се използват при проектирането на гарантоспособни разпределени системи.

Настройваемата надеждност е подходяща за използване в системи със смесена критичност на компонентите, в компактни системи, където множество възли са разположени в ограничено пространство, в системи с високи изисквания за надеждност, които позволяват работа с готови компоненти и др. под. Програмният продукт за симулационно моделиране на отказоустойчиви системи с разпределение на структурния излишък може да се използва и за моделиране на други гарантоспособни разпределени системи с добавяне на модули, които описват техните характеристики.

Резултатите, описани в дисертацията, са публикувани в 5 научни публикации, в това число 3 доклада на международни конференции (една от които с SJR ранг), 1 доклад на национална научна конференция и 1 статия в списание.

Насоки за бъдеща работа

Отказоустойчивата система с настройваема надеждност може да се изследва за различни приложения, изискващи нива на критичност, висока надеждност и разпределение на структурния излишък. Подходът на настройваема надеждност може да се усъвършенства, за да включва съобразяване и на други изисквания на приложението на гарантоспособни разпределени системи.

Моделът на отказоустойчивата разпределена система с настройваема надеждност може да се разшири за моделиране на софтуерна надеждност и изследване на ефекта на софтуерните неизправности върху отказоустойчивостта на системата. Подходът и моделът на настройваема надеждност могат да се приложат за конкретни системи.

Симулационната програма подлежи на усъвършенстване, като се ускори нейното изпълнение чрез прилагане на техники за паралелна обработка. В нея могат да се включат още блокове за изследване на влиянието на други фактори върху надеждностните характеристики на дадена система. Програмният продукт може да се развие и за изследване на други видове разпределени системи.

Списък на публикациите по дисертацията

1. Djambazova, E., & Andreev, R. (2023). Redundancy management in dependable distributed real-time systems. *Problems of Engineering Cybernetics and Robotics*. (под печат)
2. Djambazova, E. (2022). Achieving system reliability using reliability adjustment. International Conference on Computer Systems and Technologies 2022 (CompSysTech ’22), Ruse, Bulgaria. ACM, NewYork, NY, USA, pp. 64-68. DOI: 10.1145/3546118.3546129. SJR(SCOPUS) 2020: 0,18
3. Djambazova, E. (2012). Adjusting reliability of a fault-tolerant distributed process control system – Preliminary results. International Conference “Automatics and Informatics 2012”, Sofia, Bulgaria, pp. 175-178.
4. Djambazova, E. (2009). Node reliability of a fault-tolerant distributed process control system – Simulation results. International Conference “Automatics and Informatics’ 2009”, Sofia, Bulgaria, pp. I-131 – I-134.
5. Джамбазов, К., & Ананиева, Е. (1995). Управляващи системи с модулно настройване на отказоустойчивостта. Национална конференция с международно участие „Автоматика и информатика ‘95”, София, стр. 247-250.

Апробация на резултатите

Основните резултати, получени в дисертационното изследване, са представени на международни и национални научни конференции в 4 доклада, в една статия в реферирано списание с отворен достъп и на семинари на секция „Комуникационни системи и услуги“ на ИИКТ-БАН. Една от международните конференции е индексирана в Scopus, SJR 2020: 0,18.

Част от разработките са включени в работата по два национални проекта:

1. Моделиране и изследване на интелигентни системи за обучение и сензорни мрежи“ (ИСОСеМ) – Договор № КП-06-Н 47/4 от 2020 г., финансиран от ФНИ (текущ).
2. Информационни и комуникационни технологии за единен цифров пазар в науката, образованието и сигурността (ИКТ в НОС) – Д01-205/2018 г., финансиран от МОН.

Основни научни и научно-приложни резултати

Научни резултати:

1. Предложен е нов архитектурен модел на отказоустойчива разпределена система с настройваема надеждност.
2. Създаден е симулационен модел на отказоустойчива разпределена система с настройваема надеждност.
3. Синтезирана е класификация на гарантоспособни разпределени системи според възможностите им за определяне на структурния излишък в зависимост от приложението.

Научно-приложни резултати:

4. Направен е критичен анализ на гарантоспособни разпределени системи, на базата на който е разработен концептуален модел на подход за вземане на решения при осигуряване на гарантоспособност.
5. Идентифицирани са основните направления на управление на структурния излишък в гарантоспособни разпределени системи и са очертани изследователски възможности.
6. Проектиран и реализаран е софтуерен продукт за симулационно моделиране на изследваната система.
7. След сравнителен анализ на отказоустойчивата система с настройваема надеждност със системи без разпределение на структурния излишък е разработен и приложен подход на настройваема надеждност.

Приложни резултати:

8. Създадената симулационна програма може да се използва за моделиране и изследване на надеждностните характеристики и на други отказоустойчиви системи. С нея може да се изследва влиянието и на случайни хардуерни неизправности.

Декларация за оригиналност

Декларация за оригиналност на резултатите

Декларирам, че настоящата дисертация съдържа оригинални резултати, получени при проведени от мен научни изследвания. Резултатите, които са получени, описани и/или публикувани от други учени, са надлежно и подробно цитирани в библиографията.

Настоящата дисертация не е прилагана за придобиване на научна степен в друго висше училище, университет или научен институт.

Подпис:

(Едита Джамбазова)

Благодарности

Идеята за тази дисертация тръгна от проучването на възможността да се конструират гарантоспособни системи от компоненти с общо предназначение. От общата картина се очертаха много разнообразни аспекти на изследване – постигане на надеждност на единичен компонент, който няма заложени механизми за защита от неизправности, отказоустойчива синхронизация на компонентите в разпределена система, моделиране на компонент с добавени средства за самопроверка, архитектура на гарантоспособни разпределени системи и т.н., - по които съм работила през годините. Постепенно изкристализира общата конструкция и отделните парчета на пъзела се подредиха в системата, описана в дисертацията.

За последния и решаващ тласък към завършване на започнатото дължа голяма благодарност на моя научен ръководител доц. Румен Андреев, дискусиите с когото ми помогнаха да организирам изследванията си и да структурирам дисертационния труд. Винаги добронамерен и отзивчив, той ми даде нужната увереност и спокойствие за работа. Ценните му методически съвети и умение да вижда главното допринесоха изследването да придобие завършен вид.

Благодаря на Диана Георгиева, която точно и своевременно ми обясняваше процедурните правила и ми помагаше да спазвам сроковете. След един, на пръв поглед неангажиращ, разговор с нея осъзнах, че имам всичко необходимо, за да доведа дисертацията до край.

Благодаря на колегите, с които започнах професионалния си път в секциите, занимаващи се с отказоустойчиви компютърни и комуникационни системи, на ИКС-БАН и ИККС-БАН. Академичната атмосфера и научните дискусии с тях оформиха отношението ми към научната работа и отговорността към изследователските резултати. В трудните времена на липса на финансиране за научните изследвания с тях успяхме да преведем тънката нишка на научното познание до по-добрите съвременни възможности и да създадем приятелство, което надхвърля тесните професионални рамки.

Имах удоволствието и късмета тази колегиална обстановка да се прехвърли и в секция „Комуникационни системи и услуги“ на ИИКТ-БАН, където разнообразните ни научни интереси се съчетаха в един много добър проект и това подпомогна завършването на работата ми. Благодаря на колегите за подкрепата и приятелството.

Този труд е посветен на човека, с когото започна цялото приключение – Красимир Джамбазов, мой пръв научен ръководител и впоследствие съпруг. На него дължа задълбоченото си запознаване с областта на гарантоспособните системи и на научните изследвания въобще. Той се придържаше към високи професионални стандарти, коректно отношение към научните изследвания и поставяне на амбициозни цели. Взискателното му отношение към резултатите и вниманието му към детайлите налагаха качество на публикациите и търсене на сравнение с водещите научни колективи в областта. За съжаление не стигнахме заедно до крайния резултат, но той има огромен принос за моето израстване като изследовател и човек. Вярвам, че където и да си, Краси, се радваш и си доволен от резултата, който постигнах.

Искам да благодаря от цялото си сърце на моето любимо семейство – Анна, Петър и Христо, които са моята опора и вдъхновение и които дадоха смисъл на това усилие. Те са едни от малцината, които вярваха безрезервно, че ще се справя и нито за момент не престанаха да ме насърчават.

Тази дисертация е само един етап от пътя на научните ми занимания, където ме очакват нови предизвикателства. Нямам търпение да ги посрещна.

Библиография

[1] J.-C. Laprie, "Dependable computing and fault tolerance: Concepts and terminology," in *15th IEEE International Symposium on Fault-Tolerant Computing (FTCS-15)*, 1985.

[2] J.-C. Laprie, Dependability: Basic Concepts and Terminology. Dependable Computing and Fault-Tolerant Systems, vol 5., Vienna: Springer, 1992.

[3] A. Avižienis, J.-C. Laprie, B. Randell and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, p. 11–33, 2004.

[4] J.-C. Laprie, Dependability — Its attributes, impairments and means. In: Randell, B., Laprie, J.C., Kopetz, H., Littlewood, B. (eds) Predictably Dependable Computing Systems. ESPRIT Basic Research Series, Berlin, Heidelberg: Springer, 1995.

[5] A. M. Johnson and M. Malek, "Survey of software tools for evaluating reliability, availability, and serviceability," *ACM Computing Surveys*, vol. 20, no. 4, pp. 227-269, 1988.

[6] К. Джамбазов, „Изследване на отказоустойчиви изчислителни системи,“ ИКС-БАН, София, 1992.

[7] E. Djambazova, "Construct Integrity Checking: An assigned-signature technique," in *International Conference “Automatics and Informatics’2000”*, Sofia, Bulgaria, 2000.

[8] E. Djambazova, "Comparative analysis of signature-monitoring error detection techniques," in *International Conference “Automatics and Informatics’02”*, Sofia, Bulgaria, 2002.

[9] K. Wilken and J. P. Shen, "Continuous signature monitoring: efficient concurrent-detection of processor control errors," in *The 1988 International Conference on Test: New Frontiers in Testing (ITC'88)*, USA, 1988.

[10] S. S. Yau and F.-C. Chen, "An approach to concurrent control flow checking," *IEEE Transactions on Software Engineering*, Vols. SE-6, no. 2, pp. 126-137, 1980.

[11] K. Djambazov and E. Ananieva, "Analytical assessment of the coverage factor of watchdog timers," in *National Conference “Automatics and Informatics ‘96”*, Sofia, Bulgaria, 1996.

[12] K. Djambazov and E. Djambazova, "Modeling of methods for error detection in microprocessor systems," in *National Conference “Automatics and Informatics ‘99”*, Sofia, Bulgaria, 1999.

[13] D. J. Lu, "Watchdog processors and structural integrity checking," *IEEE Transactions on Computers*, vol. 31, no. 7, pp. 681-685, 1982.

[14] I. Majzik, A. Pataricza, M. Dal Cin, W. Hohl, J. Honig and V. Sieh, "Hierarchical Checking of Multiprocessors Using Watchdog Processors," in *First European Dependable Computing Conference*, 2002.

[15] A. Mahmood and E. J. McCluskey, "Concurrent error detection using watchdog processors - a survey," *IEEE Transactions on Computers*, vol. 37, no. 2, pp. 160-174, 1988.

[16] A. Pataricza, I. Majzik, W. Hohl and J. Honig, "Watchdog processors in parallel systems," *Microporocessing and Microprogramming*, vol. 39, no. 2-5, pp. 69-74, 1993.

[17] H. Kopetz, Real-Time Systems, Design Principles for Distributed Embedded Applications. 2nd ed. Real-Time Systems Series, Springer, 2011.

[18] R. Stetter, Fault-tolerant design and control of automated vehicles and processes. Insights for the synthesis of intelligent systems. Studies in Systems, Decision, and Control 201, Switzerland AG: Springer Nature, 2020.

[19] W. Hohl, E. Michel and A. Pataricza, "Hardware support for error detection in multiprocessor systems - A case study," *Microprocessors and Microsystems,* vol. 17, no. 4, pp. 201-206, 1993.

[20] J. Rushby, "Systematic Formal Verification for Fault-Tolerant Time-Triggered Algorithms," in *The 6th Working Conference on Dependable Computing for Crotocal Applications*, 1997.

[21] M. Barranco, S. Derasevic and J. Proenza, "An architecture for highly reliable fault-tolerant adaptive distributed embedded systems," *Computer,* vol. 53, p. 38–46, 2020.

[22] K. P. Birman, Reliable distributed systems technologies, Web services, and applications, NY: Springer New York, 2005.

[23] E. Dubrova, Fault-tolerant design, Springer Science+Business Media New York, 2013.

[24] K. Erciyes, Distributed real-time systems, Theory and practice. Computer Communications and Networks Series, Springer Cham, 2019.

[25] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabl, C. Senft and R. Zailinger, "Distributed fault-tolerant real-time systems: The Mars approach," *IEEE Micro,* vol. 9, no. 1, pp. 25-40, 1989.

[26] D. Powell, G. Bonn, D. Seaton, P. Verissimo and F. Waeselynck, "The Delta-4 approach to dependability in open distributed computing systems," in *The Eighteenth International Symposium on Fault-Tolerant Computing*, 1988.

[27] B. Rostamzadeh, H. Lonn, R. Snedsbol and J. Torin, "DACAPO: A distributed computer architecture for safety-critical control applications," in *Intelligent Vehicles '95 Symposium*, 1995.

[28] P. Verissimo, A. Casimiro, L. M. Pinho, F. Vasques, L. Rodrigues and E. Tovar, "Distributed computer-controlled systems: The DEAR-COTS approach," *IFAC Proceedings Volumes,* vol. 33, no. 30, pp. 113-120, 2000.

[29] R. Isermann, Fault diagnosis systems. An Introduction from fault detection to fault tolerance, New York: Springer, 2006.

[30] R. Obermaisser and H. Kopetz, Eds., GENESYS: A candidate for an ARTEMIS cross-domain reference architecture for embedded systems, Suedwestdeutscher Verlag fuer Hochschulschriften, 2009.

[31] P. Verissimo and L. Rodrigues, Distributed systems for system architects, Boston, MA: Springer, 2001.

[32] Mobileye, „Safety first for automated driving," 2019. [Онлайн]. Available: https://static.mobileye.com/website/corporate/media/Intel-Safety-First-for-Automated-Driving.pdf. [Отваряно на 2023].

[33] J.-C. Geffroy and G. Motet, Design of dependable computing systems, Springer Netherlands, 2002.

[34] I. Koren and C. M. Krishna, Fault-Tolerant Systems (2nd ed.), Elsevier Inc., 2021.

[35] P. A. Lee and T. Anderson, Fault tolerance: principles and practice, 2nd ed., Berli, Heidelberg: Springer-Verlag, 1990.

[36] D. K. Pradhan, Fault-tolerant computer system design, USA: Prentice-Hall, Inc., 1996.

[37] M. L. Shooman, Reliability of computer systems and networks: Fault tolerance, analysis, and design, USA: John Wiley & Sons, Inc., 2002.

[38] D. J. Sorin, Fault tolerant computer architecture, Morgan and Claypool Publishers, 2009.

[39] A. Babay и et al., „Deploying Intrusion-Tolerant SCADA for the Power Grid," в *49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, Portland, OR, USA, 2019 .

[40] H. Kopetz, Simplicity is Complex. Foundations of Cyber-Physical System Design, Springer Nature Switzerland AG, 2019.

[41] H. Kopetz, Simplicity is Complex. Foundations of Cyber-Physical System Design, Springer Cham, 2019.

[42] International Electrotechnical Commission (IEC), „Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 1: General requirements (IEC 61508-1:2010),“ 2010. [Онлайн]. Available: https://webstore.iec.ch/publication/5515.

[43] International Organization for Standardization (ISO), „Road vehicles — Functional safety — Part 1: Vocabulary (ISO 26262-1:2018),“ 2018. [Онлайн]. Available: https://www.iso.org/standard/68383.html.

[44] E. Djambazova and R. Andreev, "Redundancy management in dependable distributed real-time systems (in print)," *Problems Of Engineering Cybernetics And Robotics,* 2023.

[45] R. Bloomfield и J. Rushby, „Assurance 2.0: A Manifesto,“ 2020.

[46] P. D. T. O'Connor and A. Kleyner, Practical reliability engineering. Fifth Edition, John Wiley & Sons, Ltd., 2011.

[47] T. Dumitras, D. Srivastava and P. Narasimhan, "Architecting and implementing versatile dependability," in *Architecting Dependable Systems III. LNCS 3549*, R. de Lemos, C. Gacek and A. Romanovsky, Eds., Berlin, Heidelberg, Springer, 2005.

[48] P. Narasimhan, T. A. Dumitras, A. M. Paulos, S. M. Pertet, C. F. Reverte, J. G. Slember and D. Srivastava, "MEAD: Support for real-time fault-tolerant CORBA," *Concurrency and Computation: Practice and Experience,* vol. 17, pp. 1527-1545, 2005.

[49] B. Randell, J.-C. Laprie, H. Kopetz and B. Littlewood, Predictably dependable computing systems, Springer, 1995.

[50] B. Randell and J. Xu, "The evolution of the recovery block concept," in *Software Fault Tolerance*, M. (. Lyu, Ed., Wiley, 1995, pp. 1-21.

[51] A. Avizienis and L. Chen, "On the implementation of N-version programming for software fault tolerance during execution," in *Proceedings of the IEEE COMPSAC 77 Conference*, 1977.

[52] A. Avizienis and J. P. J. Kelly, "Fault tolerance by design diversity: Concepts and experiments," *Computer,* vol. 17, no. 8, pp. 67-80, 1984.

[53] H. Kopetz, "The time-triggered model of computation," in *Real-Time Systems Symposium (RTSS98)*, Madrid, Spain, 1998.

[54] D. P. Siewiorek and R. S. Swarz, Reliable computer systems: design and evaluation. 3rd ed., A. K. Peters, Ltd., 1998.

[55] A. S. Tanenbaum and M. van Steen, Distributed systems: Principles and paradigms. 2nd ed., Pearson Prentice Hall, 2017.

[56] D. Powell, A generic fault-tolerant architecture for real-time dependable systems, Kluwer Academic Publishers, 2001.

[57] D. Powell, J. Arlat, L. Beus-Dukic, A. Bondavalli, P. Coppola, A. Fantechi, E. Jenn, C. Rabejac and A. Wellings, "GUARDS: A generic upgradable architecture for real-time dependable systems.," *IEEE Transactions on Parallel and Distributed Systems, Special issue on Dependable Real-Time Systems,* vol. 10, no. 6, pp. 580-599, 1999.

[58] W. Herzner, R. Schlick, M. Schlager, B. Leiner, B. Huber, A. Balogh, G. Csertan, A. LeGuennec, T. LeSergent, N. Suri and S. Islam, "Model-based development of distributed

embedded real-time systems with the DECOS tool-chain," SAE Technical Paper 2007-01-3827, 2007.

[59] R. Obermaisser, P. Peti, B. Huber and C. El Salloum, "DECOS: an integrated time-triggered architecture," *Elektrotech. Inftech.*, vol. 123, pp. 83-95, 2006.

[60] A. Burns and R. I. Davis, "Mixed criticality systems - A review," University of York, UK, 2022. [Online]. Available: https://www-users.york.ac.uk/~ab38/review.pdf.

[61] S. Islam, R. Lindstrom and N. Suri, "Dependability driven integration of mixed criticality SW components," in *Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'06)*, Gyeongju, South Korea, 2006.

[62] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proceedings of the IEEE,* vol. 91, no. 1, pp. 112-126, 2003.

[63] D. Powell, Delta-4 - A generic architecture for dependable distributed computing, ESPRIT Research Reports. Springer-Verlag, 1991.

[64] D. Powell, "Failure mode assumptions and assumption coverage," in *FTCS-22: The Twenty-Second International Symposium on Fault-Tolerant Computing*, Boston, MA, USA, 1992.

[65] D. Powell, "Distributed fault tolerance – Lessons learnt from Delta-4. Hardware and Software Architectures for Fault Tolerance. Experiences and Perspectives," *Lecture Notes in Computer Science,* vol. 774, pp. 199-217, 1994.

[66] L. M. Pinho, F. Vasques and A. Wellings, "Replication management in reliable real-time systems," *Real-Time Systems,* vol. 26, no. 3, pp. 261-296, 2004.

[67] P. Verissimo, A. Casimiro and C. Fetzer, "The timely computing base: Timely actions in the presence of uncertain timeliness," in *International Conference on Dependable Systems and Networks. DSN 2000*, New York, NY, USA, 2000.

[68] H. Kopetz and G. Grunsteidl, "TTP - A time-triggered protocol for fault-tolerant real-time systems," in *FTCS-23 The Twenty-Third International Symposium on Fault-Tolerant Computing*, Toulouse, France, 1993.

[69] R. Maier, G. Bauer, G. Stoger and S. Poledna, "Time triggered architecture: A consistent computing platform," *IEEE Micro,* vol. 22, no. 4, pp. 36-45, 2002.

[70] H. Kopetz, A. Kruger, D. Millinger and A. Schedl, "A synchronization strategy for a time-triggered multicluster real-time system," in *The 14th Symposium on Reliable Distributed Systems*, Bad Neuenahr, Germany, 1995.

[71] M. Cukier and et al., "AQuA: An adaptive architecture that provides dependable distributed objects," in *Seventeenth IEEE Symposium on Reliable Distributed Systems (Cat. No.98CB36281)*, West Lafayette, IN, USA, 1998.

[72] Z. T. Kalbarczyk, R. K. Iyer, S. Bagchi and K. Whisnant, "Chameleon: A software infrastructure for adaptive fault tolerance," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 6, pp. 560-579, 1999.

[73] К. Джамбазов и Е. Ананиева, „Управляващи системи с модулно настройване на отказоустойчивостта,“ в *Национална конференция с международно участие „Автоматика и информатика ‘95”*, София, 1995.

[74] E. Djambazova, "Adjusting reliability of a fault-tolerant distributed process control system – Preliminary results," in *International Conference “Automatics and Informatics’ 2012”*, Sofia, Bulgaria, 2012.

[75] E. Djambazova, "A fault-tolerant real-time system with adjustable reliability," in *ACM International Conference Proceeding Series, CompSysTech'21*, Ruse, Bulgaria, 2021.

[76] E. Djambazova, "Achieving system reliability using reliability adjustment," in *ACM International Conference Proceeding Series, International Conference on Computer Systems and Technologies 2022 (CompSysTech ’22)*, Ruse, Bulgaria, 2022.

[77] K. Trivedi and A. Bobbio, Reliability and Availability Engineering: Modeling, Analysis, and Applications, Cambridge: Cambridge University Press, 2017.

[78] B. Huber and R. Obermaisser, "Model-based development of integrated computer systems: Modeling the execution platform," in *2007 Fifth Workshop on Intelligent Solutions in Embedded Systems*, Leganes, Spain, 2007.

[79] K. S. Trivedi, J. Bechta Dugan, R. Geist and M. Smotherman, "Hybrid reliability modeling of fault-tolerant computer systems," *Comput. Electr. Eng.*, vol. 11, no. 2-3, pp. 87-108, 1984.

[80] K. S. Trivedi, Probability and Statistics with Reliability, Queuing, and Computer Science Applications, 1984.

[81] L. Portinale and D. C. Raiteri, Modeling and analysis of dependable systems: A probabilistic graphical model perspective, USA: World Scientific Publishing Co., Inc., 2015.

[82] S. Distefano and A. Puliafito, Dynamic reliability block diagrams: Overview of a methodology. Risk, Reliability and Societal Safety, A. &. V. (eds), Ed., London: Taylor & Francis Group, 2007.

[83] M. Rausand, Reliability of Safety-Critical Systems: Theory and Applications (1st. ed.), Wiley Publishing, 2014.

[84] R. A. Sahner, K. S. Trivedi and A. Puliafito, Performance and Reliability Analysis of Computer Systems, 1996.

[85] N. Limnios, Fault Trees, Wiley-ISTE, 2013.

[86] C. Schneeweiss, Hierarchies in Distributed Decision Making, Berlin, Heidelberg: Springer, 1999.

[87] C. Beounes, M. Aguera, J. Arlat, S. Bachmann, C. Bourdeau, J.-E. Doucet, K. Kanoun, J.-C. Laprie, S. Metge, J. Moreira de Souza, D. Powell and P. Spiesser, "SURF-2: A program for dependability evaluation of complex hardware and software systems," in *FTCS-23 The Twenty-Third International Symposium on Fault-Tolerant Computing*, Toulouse, France, 1993.

[88] R. W. Butler, "An abstract language for specifying Markov reliability models. , (), .," *IEEE Transactions on Reliability*, Vols. R-35, no. 5, pp. 595-601, 1986.

[89] J. B. Dennis, Petri Nets, D. E. o. P. C. Padua, Ed., Springer, Boston, MA, 2011.

[90] J. L. Peterson, Petri Net theory and the modeling of systems, USA: Prentice Hall PTR, 1981.

[91] J. B. Dugan, K. S. Trivedi, R. Geist and V. F. Nicola, "Extended stochastic Petri nets: Applications and analysis," in *International Symposium on Computer Modeling, Measurement and Evaluation*, 1984.

[92] W. H. Sanders and J. F. Meyer, "Stochastic Activity Networks: Formal Definitions and Concepts.," *Lectures on Formal Methods and Performance Analysis. EEF School 2000. Lecture Notes in Computer Science in Computer Science*, vol. 2090, 2001.

[93] M. A. Marsan, G. Conte and G. Balbo, "A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems," *ACM Transactions on Computer Systems*, vol. 2, no. 2, pp. 93-122, 1984.

[94] E. Djambazova, "Node reliability of a fault-tolerant distributed process control system – simulation results," in *International Conference “Automatics and Informatics’ 2009”*, Sofia, Bulgaria, 2009.

[95] E. Djambazova and K. Djambazov, "Time-dependent coverage factor model," in *International Conference "Automatics and Informatics'2000"*, Sofia, Bulgaria, 2000.

[96] E. Djambazova and K. Djambazov, "Processor control-flow error-detection techniques - Model and evaluation tool," *Cybernetics and Information Technologies,* vol. 1, no. 2, pp. 3-18, 2001.

[97] H. Madeira and et al., "Time behavior monitoring as an error detection mechanism," in *Third IFIP Working Conference on Dependable Computing for Critical Applications (DCCA-3),* 1992.

[98] Д. Е. Форсайт, М. А. Малкълм и К. Б. Молър, Компютърни методи за математически пресмятания, София: Наука и изкуство, 1986.

[99] Т. Калоянов, Статистика, София: Тракия-М, 2004.

[100] И. А. Ушаков, Надежность технических систем, Москва: Радио и связь, 1985.

[101] B. Gnedenko and I. Ushakov, Probabilistic Reliability Engineering, John Wiley & Sons, Inc., 1995.

[102] T. F. Arnold, "The concept of coverage and its effect on the reliability model of repairable systems," *IEEE Transactions on Computers,* vol. 22, no. 6, pp. 251-254, 1973.

Приложение А

Математическото изразяване на атрибутите на гарантоспособността, както и изследваните надеждностни характеристики на отказоустойчивите системи се основават на теорията на вероятностите и математическата статистика. В приложението са изброени и дефинирани основните характеристики, които са използвани в дисертационния труд според няколко източника [77], [100], [101]. Посочени са вероятностно и статистическо представяне на надеждностните характеристики. Вероятностното представяне дава математическа дефиниция, която изразява непрекъснатостта на явленията в отказоустойчивите разпределени системи. То е удобно при аналитични пресмятания на надеждността. Статистическото представяне е подходящо при симулационното моделиране на системите.

1) *Надеждност* [100]

Системната надеждност се представя с функция на разпределение на t_F – времето до първи отказ. Тя се изразява математически като вероятност за безотказна работа на системата в интервала (0,*t*) [100].

a) Вероятностно представяне

$$R(t) = P(t_F \geq t) \qquad (П1)$$

b) Статистическо определяне

$$\bar{R}(t) = \frac{N(t)}{N(0)} = 1 - \frac{n(t)}{N(0)} \qquad (П2)$$

n(t) – брой отказали компоненти в момент *t*,

N(0) – брой работоспособни компоненти в началото на интервала [0,*t*],

N(t) – брой изправни компоненти в момент *t*.

2) Вероятност за отказ на системата в интервал (0,*t*)

Допълва до единица вероятността за безотказна работа.

$$Q(t) = 1 - R(t) \qquad (П3)$$

$$\bar{Q}(t) = 1 - \bar{R}(t)$$

3) Разпределение на времето до първи отказ

Показателят се дефинира с вероятността

$$F(t) = P(t_F \leq t) \qquad (П4)$$

и е числено равен на *Q(t).*

4) *Средно време до отказ* (Mean Time To Failure – MTTF)

Друга важна характеристика на гарантоспособните системи е средното време до отказ.

a) Вероятностно определяне

Средното време до отказ представлява математическото очакване на t_F.

$$MTTF = E[t_F] = \int_0^\infty R(t)\,dt \qquad (П5)$$

b) Статистическо определяне

$$\overline{MTTF} = \frac{1}{N(0)} \sum_{i=1}^{N(0)} t_F^{(i)} \qquad (П6)$$

където

$t_F^{(i)}$ – време до първия отказ на *i*-тата изпитвана система (случайна величина).

5) *Средно време между отказите* (Mean Time Between Failures - MTBF)

Дефинира се при системи с ремонт. *MTBF* е математическото очакване на граничната стойност на времето на работа между отказите за стационарен процес.

a) Вероятностно определяне

$$MTBF = T_\infty = \lim_{k\to\infty} E[T_k] = \lim_{k\to\infty} \frac{1}{k} \sum_{j=1}^{k} T_j \qquad (П7)$$

където:

T_k е средното време за работа на системата от завършването на *(k-1)*-ия ремонт до *k*-тия отказ и се определя като:

$$T_k = E\left[t_F^{(k)}\right] = \int_0^\infty R_k(t)\,dt,$$

което е математическото очакване (средна стойност) на времето за нормална работа на системата от момента на завършване на *(k-1)*-вия отказ.

b) Статистическо определяне

$$\overline{MTBF} = \bar{T}_k = \frac{1}{N(0)} \sum_{i=1}^{N(0)} \left\{ \theta_k^{(i)} \middle| k \gg 1 \right\} \qquad (П8)$$

$\bar{T}_k$ е средно аритметична стойност на времето за нормална работа до *k*-тия отказ при достатъчно голяма стойност на *k*.

6) *Стационарен коефициента на готовност на системата* (availability)

Готовността *A* е вероятността системата да се намира в работоспособно състояние за стационарен случаен процес (т.е. в произволен достатъчно отдалечен момент от време) или математическото очакване на отношението на времето (за стационарен случаен процес), в

течение на което системата се намира в работоспособно състояние в определен интервал от време към общата продължителност на интервала.

a) Вероятностно определяне

$$A = \lim_{t\to\infty} A(t) = \lim_{t\to\infty} A^*(t) \qquad (П9)$$

За произволни *MTBF* и времена за ремонт, имащи крайна средна стойност T_∞ и τ, може да се напише:

$$A = \frac{T_\infty}{T_\infty + \tau} \qquad (П10)$$

където

τ – средно време за ремонт

b) Статистическо определяне

$$\overline{A} = \frac{N(t_\infty)}{N(0)} = 1 - \frac{n(t_\infty)}{N(0)} \qquad (П11)$$

7) *Средно време на престой* (Mean Down Time - MDT)

Времето на престой (downtime) на системата се определя от времето, през което системата се намира в състояние на неработоспособност.

Средното време на престой е математическото очакване на времето за престой на системата.

$$MDT = \lim_{k\to\infty} E\left[t_D^{(k)}\right] \qquad (П12)$$

където

$t_D^{(k)}$ е времето за престой след *k*-тия отказ на системата.

8) *Средно време за ремонт* (Mean Time To Repair – MTTR)

Средното време за ремонт е математическото очакване на времето за ремонт на система.

a) Вероятностно определяне

$$MTTR = \tau = E[\eta] = \int_0^\infty [1 - G(t)]\, dt \qquad (П13)$$

където

η е времето за ремонт (случайна величина),

G(t) – разпределение на времето за ремонт.

b) Статистическо определяне

$$\overline{MTTR} = \bar{\tau} = \frac{1}{N(0)} \sum_{i=1}^{N(0)} \eta^{(i)} \quad \text{(П14)}$$

9) *Интензивност на ремонт/възстановяване на системата в момент t*

Интензивността на ремонта е условната плътност на вероятността за възстановяване на системата в момент *t*, отчитано от началото на възстановяването, при условие че в момент *t* не е направено възстановяване.

a) Вероятностно определяне

$$\mu(t) = \frac{g(t)}{1-G(t)} \quad \text{(П15)}$$

където

g(t) е интензивността на разпределение на *G(t)*.

b) Статистическо определяне

$$\bar{\mu}(t) = \frac{n_B(t+\Delta t) - n_B(t)}{N_B(t)\Delta t} = \frac{N_B(t+\Delta t) - N_B(t)}{N_B(t)\Delta t} = \frac{\Delta n_B(t,t+\Delta t)}{N_B(t)\Delta t} \quad \text{(П16)}$$

където

$n_B(t)$ – брой елементи от системата, ремонтът на които е извършен до момент *t*,

$N_B(t)$ – брой елементи от системата, чийто ремонт се извършва след момент *t*,

$\Delta n_B(t, t+\Delta t)$ - брой елементи на системата, чийто ремонт се извършва след момент *t* и преди момент $t + \Delta t$.

10) *Покритие* (coverage)

Покритието е мярка за работата на механизмите за защита от грешки в системата [6], [102]. Коефициентът на покритие се определя като условната вероятност системата да се възстанови при възникване на активен отказ.

$$C = P_{ck} P_B \quad \text{(П17)}$$

където

P_{ck} е показател за качеството на системите за контрол, а P_B – вероятност за коректно възстановяване.

$$P_{ck} = 1 - \prod_{j=1}^{\nu} \left(1 - P_j\right) \quad \text{(П18)}$$

$$P_{ck} = \prod_{j=1}^{n} P_j P_{j0} \quad \text{(П19)}$$

11) *Интензивност на неизправности* [100]

a) Вероятностно определяне

$$f(t) = \frac{dF(t)}{dt} = \frac{dQ(t)}{dt} = -\frac{dR(t)}{dt} \qquad (П20)$$

където

Q(t) е вероятността за възникване на неизправност в елемент от системата за интервал от време (0,t) и е равна на *1-R(t)*.

b) Статистическо определяне

$$\bar{f}(t) = \frac{N(t+\Delta t)-N(t)}{N(0)\Delta t} = \frac{\Delta n(t,t+\Delta t)}{N(0)\Delta t} \qquad (П21)$$

където

Δn е броят на отказалите елементи в интервала $(t,t+\Delta t)$.

12) *Интензивност на неизправностите в момент t*

a) Вероятностно определяне

$$\lambda(t) = \frac{1}{1-F(t)}.\frac{dF(t)}{dt} = \frac{f(t)}{P(t)} \qquad (П22)$$

където

F(t) е функция на разпределение на времето до първата неизправност,

P(t) – вероятност за безотказна работа в интервала (0,*t*),

f(t) – плътност на разпределение на *F(t)*.

b) Статистическо определяне

$$\bar{\lambda}(t) = \frac{N(t+\Delta t)-N(t)}{N(t)\Delta t} = \frac{\Delta n(t,t+\Delta t)}{N(t)\Delta t} \qquad (П23)$$

$\bar{\lambda}(t)$ е отношението на броя на неизправностите в интервала $[t,t+\Delta t]$ към произведението на броя на изправните системи в момент *t* и продължителността на интервала Δt.

13) *Интензивност на неизправностите* [100] при постоянна плътност на потока на неизправностите

a) Вероятностно определяне

$$\lambda = \frac{1}{T} \qquad T = MTBF \qquad (П24)$$

където

λ е математическото очакване на броя на неизправностите в системата с ремонт за единица време, при установен процес на експлоатация, т.е. при постоянна плътност на потока на неизправностите.

b) Статистическо определяне

$$\bar{\lambda} = \frac{1}{\bar{T}} \qquad (П25)$$

където

$\bar{\lambda}$ е средният брой неизправности в системата с ремонт за единица време.

14) *Детектируемост на неизправност* (fault detectability)

Определя се от вероятността възникналата неизправност да принадлежи към класа на детектируемите неизправности.

$$C_{fm} = Pr\{fault\ is\ detectable | fault\ occurs\} \qquad (П26)$$

Ако възникването на всички неизправности е равномерно, C_{fm} се определя като

$$C_{fm} = \frac{D}{F} \qquad (П27)$$

където:

D – детектируеми неизправности,

F – общо количество неизправности.

15) Коефициент на покритие

Коефициентът на покритие на даден механизъм в най-общ вид е функция на C_{fm} (П27) и *t*.

$$C_t(t) = F\left(C_{fm}, t\right) \qquad (П28)$$

При експоненциално разпределение:

$$C_t(t) = C_{fm}\left(1 - e^{-\delta t}\right) \qquad (П29)$$

```
// NMRSIM1.cpp : Simulating structural redundancy

#include "stdafx.h"
#include "stdio.h"
#include "stdlib.h"
#include <string.h>

typedef enum el_states {normal, detected, undetected};
typedef enum sys_states {snormal, stop, failure};
typedef enum faults {none, permanent, transient, repair};

typedef struct {
  unsigned int number;          //Number of modules in a node
  el_states  state;             //Node's state
  long double ftime[3];         //Times of fault: PF, TF, repair
  long double lambda[3];        //Fault rates
  unsigned long lastf[3];       //Last fault
  faults  fault;                //Fault flag
  bool curr_proc;               //Processor fault in the current cycle
  long double C0;               //Node's coverage
  unsigned long lastC;
} element;

void input(void);
extern long double urande(unsigned long &);     //Random number generator
extern long double time_val(unsigned long &,long double);   //Time values
extern void adjust(long double &);                  //Measures adjustment
long double min_time(void);                     //Minimal time
void init(void);                                //Initialize
void fault_time(void);                          //Fault occurrence
void el_faults(faults &, unsigned int);         //Element faults
void el_state(unsigned int);                    //Element's state
void st_count(unsigned int []);                 //States count
sys_states system_state(void);                  //System state
void el_repair(unsigned int);                   //Element repair
void system_level(unsigned int);                //Elelment's state at system level
void fault_repair(unsigned int, unsigned int);  //Fault repair
void el_coverage(void);                         //Initial values of the coverage
long double sqr(long double);                   //calculates square
long double sum_sqr(long double *, long double *, unsigned int);            //sum of
squares
void stats(long double *, long double&, long double, unsigned int, long double&);
//standard deviation and confidence
void modules (unsigned int, unsigned int &);            //Combination of modules
void constr(char [], unsigned int);             //Concatenate strings

element *pr_ptr;
unsigned int N /*number of nodes*/, Nc /*number of coverage values for the study*/;
long double Cmod[3];                    //Coverage for 1, 2, and 3 modules resp.
long double* cov_factor[3];             //array to store values of C1, C2 and C3
long double srepair;                    //system repair rate
long double maxt, uptime;               //max time; max uptime in repairable system
unsigned int iter, itr, it;             //Number of iterations
unsigned int pp, pp_rel, num_stops, num_failures, *stops;       //Reliability points;
number of stops; number of failures
long double stepT;                      //Step of the time
unsigned int cnt[3];
long double cycle;                      //current cycle
long double downtime;                   //total downtime in an iteration
long double s;                          //for the RNG and time_val
unsigned long m2 = 0;                   //for the RNG
sys_states sstate;                      //System state
long double *ttf, *rel, mttf, /*MTF[MAXC]*/ *rel1[35], *ttr, mttr /*MTR[MAXC]*/;
//Times to failure & reliability of each combination, time to repair
long double *tbf, mtbf /*MBF[MAXC]*/;           //Time between failures
long double *tts, mtts, /*MTS[MAXC],*/ *tbs, mtbs /*MBS[MAXC]*/;            //time to
stop & mean time to stop; mean time between stops
long double *dwt, /*MDWT[MAXC],*/ downt;        //Collecting downtime
unsigned long step;
unsigned long lastR;                    //last value for system repair
```

```
66  FILE *ff, *ff1;                          //Output files
67  long double adj;                         //Time adjustment factor
68  bool coverage;                           //for the cycle of C or Cr, if needed
69  unsigned long lastX; /*for delta in time_val*/
70  unsigned long last_cov;                  //for determining C of the nodes
71  long double stdev_ttf, stdev_tbf, stdev_tts, stdev_tbs, stdev_down, stdev_rel, sum_rel
    ,
72              conf_rel, conf_ttf, conf_tbf, conf_tts, conf_tbs, conf_down;
73  long double availability, R_total, mission_time, R_time;
74  float sum_stops;                         //number of stops
75  unsigned int maxM, comb[35][3], *node_mod[35], Ncomb, combin;       //maximum number
    of modules, combinations of nodes with 1, 2, and 3 modules
76
77
78
79
80  int _tmain(int argc, _TCHAR* argv[])
81  {
82  unsigned int i, j, pf /**points of failure in non-repairable system**/, ii, jj, kk,
    COMB /*number of combinations studied*/;
83  char str[120];
84  char str1[120];
85  errno_t err;
86  char ans;
87
88  printf("Number of nodes: ");
89  scanf("%u", &N);
90  if ((pr_ptr = (element *) calloc(N,sizeof(element))) == NULL) {
91    printf("Not enough memory to allocate buffer\n");
92    exit(1);
93  }
94
95  input();
96  last_cov = 356;
97      printf("Maximal number of modules: ");
98      scanf("%u", &maxM);
99      maxM = 2 * N;
100     modules(maxM, Ncomb);
101
102     if (Ncomb == 1) Nc = 1000;
103     else Nc = 10;                    //number of coverage values
104 for (i = 0; i < 3; i++)
105     if ((cov_factor[i] = (long double*)calloc(Nc, sizeof(long double))) == NULL) {
106         printf("Not enough memory to allocate buffer\n");
107         exit(1);
108     }
109
110 for (ii = 0; ii < Nc; ii++) {        //iterations for number of coverage values
111     el_coverage();                   //Define coverage for 1, 2, 3 modules
112     for (i = 0; i < 3; i++) {
113         cov_factor[i][ii] = Cmod[i];
114     }
115 }
116
117 it = 0;
118
119 if (Ncomb == 1) COMB = 1;
120 else COMB = Nc;
121
122 for (ii=0; ii<COMB; ii++) {          //iterations for C1
123     for (jj=0; jj<Nc; jj++) {        //iterations for C2
124         for (kk=0; kk<COMB; kk++) { //iterations for C3
125
126
127 for (combin=0; combin<Ncomb; combin++) {            //all combinations of nodes with
    maxM modules
128     for (i = 0; i<N; i++) {
129         pr_ptr[i].number = node_mod[combin][i];
130         switch (pr_ptr[i].number) {
131         case 1: pr_ptr[i].C0 = cov_factor[pr_ptr[i].number - 1][ii]; break;
```

```
132           case 2: pr_ptr[i].C0 = cov_factor[pr_ptr[i].number - 1][jj]; break;
133           case 3: pr_ptr[i].C0 = cov_factor[pr_ptr[i].number - 1][kk]; break;
134           }
135           //printf("Combination %u of %u module(s) C0= %Lf\n", combin + 1,
              pr_ptr[i].number, pr_ptr[i].C0);
136       }
137
138   lastX = 777;  lastR = 23;
139
140   for (itr=0; itr<iter; itr++) {
141     init();           //initial values for lastC and pr_ptr
142     ttf[itr] = 0.0;       tbf[itr] = 0.0;     ttr[itr] = 0.0;
143     tts[itr] = 0.0;       tbs[itr] = 0.0;     dwt[itr] = 0.0;
144     num_stops = 0;        stops[itr] = 0;
145     sstate = snormal;
146     fault_time();
147
148     do {              //until failure
149       cycle = min_time();
150       if (uptime != 0.0 && srepair != 0.0) {      //for repairable systems
151           if (cycle <= uptime) sstate = snormal;
152           else sstate = failure;
153       }
154       for (i=0; i<N; i++) {
155         if (pr_ptr[i].curr_proc)
156             el_state(i);               //according to C determines node's state
157       }
158
159       if (sstate != failure) {
160         st_count(cnt);
161         for (i=0; i<N; i++)
162             if (pr_ptr[i].state != normal && pr_ptr[i].curr_proc)
163               system_level(i);
164         st_count(cnt);
165         sstate = system_state();
166       }
167       if (sstate == failure) {
168   //      printf("\nFailure[%u] at cycle %Lf. State %d %d
      %d.\n",itr,cycle,cnt[0],cnt[1],cnt[2]);
169           st_count(cnt);
170           if (sstate == system_state()) {
171             ttf[itr] = cycle;
172             if (ttf[itr] == 0.0)
173                 printf("\nTTF = %Lf at iteration %Lf. State %d %d %d.\n", ttf[itr], itr,
                    cnt[0], cnt[1], cnt[2]);
174             if (srepair != 0.0) {
175               tbf[itr] = cycle/(num_stops+1);
176             }
177           }
178           else if (srepair != 0.0 && num_stops != 0 && num_stops != 1)  {
179               tbf[itr] = tbf[itr]/num_stops;
180           }
181           if (srepair != 0.0) {
182             if (num_stops != 0 && num_stops != 1) {
183                 ttr[itr] = ttr[itr]/num_stops;
184                 tbs[itr] = tbs[itr]/num_stops;
185             }
186           }
187       }
188       if (sstate == stop) {
189   //      printf("\nStop[%u] at cycle %Lf. State %d %d
      %d.\n",itr,cycle,cnt[0],cnt[1],cnt[2]);
190         num_stops++;
191         stops[itr]++;
192         if (srepair != 0.0) {
193           tts[itr] = cycle;
194           tbs[itr] = cycle;
195           ttf[itr] = cycle;
196           tbf[itr] = cycle;       //stops are failures with possible repair
197           downtime = time_val(lastR,srepair);
```

```
198           cycle = cycle + downtime;
199           dwt[itr] += downtime;
200           ttr[itr] += downtime;
201
202           for (i=0; i<N; i++) {
203             for (j=0; j<3; j++) {
204                 if (pr_ptr[i].fault != none) {        //Operational nodes don't obtain
                    new fault times!
205                   pr_ptr[i].ftime[j] = 0.0;
206                   if (pr_ptr[i].lambda[j] != 0.0 && j != 2)
207                   pr_ptr[i].ftime[j] = cycle + time_val(pr_ptr[i].lastf[j], pr_ptr[i].
                      lambda[j]);
208               }
209             }
210             pr_ptr[i].fault = none;
211             pr_ptr[i].curr_proc = false;
212             pr_ptr[i].state = normal;
213           }
214         }
215         else {
216           ttf[itr] = cycle;
217           tts[itr] = cycle;
218           sstate = failure;
219         }
220       }
221       cycle = 0.0;
222     } while (sstate != failure);
223
224   }   //for itr
225
226   mttf = 0.0;     mttr = 0.0;     downt = 0.0;    mtts = 0.0;     mtbs = 0.0;     mtbf =
       0.0;
227   pf = 0;     num_stops = 0;      num_failures = 0;   sum_stops = 0.0;
228   for (i=0; i<iter; i++) {
229       if (srepair != 0.0) {
230           if (ttf[i] != 0.0) {            //!!! Check!
231               adjust(ttf[i]);     mttf += ttf[i];
232               adjust(tbf[i]);     mtbf += tbf[i];
233           }
234           else {
235               num_failures++;
236               printf("\nTTF = 0.0 %d times.\n", num_failures);
237           }
238           if (tts[i] != 0.0) {            //!!! Check!
239               adjust(tts[i]);     mtts += tts[i];
240               adjust(ttr[i]);     mttr += ttr[i];
241               adjust(dwt[i]);     downt += dwt[i];
242               adjust(tbs[i]);     mtbs += tbs[i];
243           }
244           else num_stops++;
245       }
246       else {
247           adjust(ttf[i]);     adjust(tbf[i]);
248           mttf += ttf[i];     mtbf += tbf[i];
249           if (tts[i] != 0.0) {
250               adjust(tts[i]);     mtts += tts[i];
251           }
252           else pf++;
253       }
254       sum_stops += stops[i];
255   }
256   stats(ttf, stdev_ttf, mttf, iter, conf_ttf);
257   stats(tts, stdev_tts, mtts, iter, conf_tts);
258   if (srepair != 0.0) {
259       stats(tbf, stdev_tbf, mtbf, iter, conf_tbf);
260       stats(tbs, stdev_tbs, mtbs, iter, conf_tbs);
261       stats(dwt, stdev_down, downt, iter, conf_down);
262   }
263
264   if (num_failures != iter) {
```

```
265         mttf = mttf/(iter-num_failures);        mtbf = mtbf/(iter-num_failures);
266     }
267     else {
268         mttf = mttf/iter;   mtbf = mtbf/iter;
269     }
270     if (num_stops != iter) {
271         mttr = mttr/(iter-num_stops);       downt = downt/(iter-num_stops);
272         mtbs = mtbs/(iter-num_stops);
273     }
274     else {
275         mttr = mttr/iter;       downt = downt/iter;
276         mtbs = mtbs/iter;
277     }
278     if (srepair == 0.0) mtts = mtts/(iter-pf);
279     else    if (num_stops != iter) mtts = mtts/(iter-num_stops);
280             else mtts = mtts/iter;
281     sum_stops /= iter;
282
283     printf("\nMTTF = %Lf\t",mttf);      printf("\tMTTF between %Lf and %Lf\n",mttf-
        conf_ttf, mttf+conf_ttf);
284     printf("\tStandard deviation of MTTF %Lf \n", stdev_ttf);
285     printf("\nMTTS = %Lf\t",mtts);      printf("\tMTTF between %Lf and %Lf\n",mtts-
        conf_tts, mtts+conf_tts);
286
287
288
289     if (srepair != 0.0) {
290         if (mtbf != 0.0 && mttr != 0.0) availability = mtbf/(mttr+mtbf);
291         else availability = 0.0;
292         printf("\nMTTR = %Lf\t",mttr);      printf("\tA = %Lf\n",availability);     printf
            ("\tMTTF = %Lf\n", mttf);
293     }
294
295
296
297     pp_rel = 1000;          //pp=100
298     step = ceil(maxt/pp_rel);       //step to calculate rel; maxt=3e5
299     if (it == 0) {
300         if ((rel = (long double *) calloc(pp_rel,sizeof(long double))) == NULL) {
301         printf("Not enough memory to allocate buffer\n");
302         exit(1);
303     }
304
305     if ((rel1[combin] = (long double *) calloc(pp_rel,sizeof(long double))) == NULL) {
306         printf("Not enough memory to allocate buffer\n");
307         exit(1);
308     }
309     }
310
311     for (i=0; i<pp_rel; i++) rel[i] = 0.0;
312     sum_rel = 0.0;
313     for (i=0; i<pp_rel; i++) {
314       for (j=0; j<iter; j++)
315         if (ttf[j]<=step*(i+1)) rel[i]++;
316       rel[i] = 1.0-(rel[i]/iter);
317       sum_rel +=rel[i];
318     }
319
320     stats(rel, stdev_rel, sum_rel, pp_rel, conf_rel);
321     div_t x;
322     x = div(pp_rel, 2);
323     printf("\nR = %Lf\t",rel[x.quot]);      printf("\tAverage R between %Lf and %Lf\n",
        sum_rel/pp_rel-conf_rel, sum_rel/pp_rel+conf_rel);   //rel 28.11.2013
324     printf("\tStandard deviation of R %Lf\n", stdev_rel);
325
326
327     for (i=0; i<pp_rel; i++){
328         if (Ncomb == 1) {
329             if (ii == 0 && jj == 0 && kk == 0) rel1[combin][i] = rel[i] / Nc;
330             else rel1[combin][i] += rel[i] / Nc;
```

```
331         }
332         else {
333             if (ii == 0 && jj == 0 && kk == 0) rel1[combin][i] = rel[i] / (Nc * Nc * Nc);
334             else rel1[combin][i] += rel[i] / (Nc * Nc * Nc);
335         }
336     }
337 
338 
339         step = 1;   pp=150;         //if pp>iter only values up to iter are calculated
340         for (i=0; i<iter; i++) ttr[i] = 0.0;        //normalised number of stops, by
            intervals; just using ttr
341         for (i = 0; i < pp; i++) {          //used to be up to (pp+1). Shoud be checked!
342             for (j = 0; j < iter; j++) {
343                 if (i < iter) {
344                     if (stops[j] == i) ttr[i]++;
345                 }
346             }
347             if (i < iter) ttr[i] = ttr[i] / iter;   //if (i == 0) ttr[i+1] =
                ttr[i+1]/iter;
348         }
349 
350 
351         pp = 500;               //if pp>iter only values up to iter are calculated
352         step = ceil(maxt / pp);     //step to calculate ttf distribution; maxt=3e5
353         for (i=0; i<iter; i++) {
354             tbs[i] = 0.0;           //normalised distribution of ttf & tts, just using
                tbs & tbf arrays resp. for data storage, by intervals
355             tbf[i] = 0.0;
356         }
357         for (i=0; i<pp; i++) {
358             for (j=0; j<iter; j++) {
359                 if (i == 0) {
360                     if (ttf[j]<=step*(i+1)) tbs[i]++;
361                     else if (ttf[j]>step*(i+1) && ttf[j]<=step*(i+2)) tbs[i+1]++;
362                     if (tts[j]<=step*(i+1)) tbf[i]++;
363                     else if (tts[j]>step*(i+1) && tts[j]<=step*(i+2)) tbf[i+1]++;
364                 }
365                 else {
366                     if (i < iter) {
367                         if (ttf[j] > step* (i + 1) && ttf[j] <= step * (i + 2)) {
368                             tbs[i]++;
369     //                      printf("TTF = %Lf.", ttf[j]); system("pause");
370                         }
371                         if (tts[j] > step* (i + 1) && tts[j] <= step * (i + 2)) tbf[i]++;
372                     }
373                 }
374             }
375             if (i < iter) {
376                 tbs[i] = tbs[i] / iter; if (i == 0) tbs[i + 1] = tbs[i + 1] / iter;
377                 tbf[i] = tbf[i] / iter; if (i == 0) tbf[i + 1] = tbf[i + 1] / iter;
378             }
379         }
380 
381 
382         strcpy(str, "C:\\Results\\mttf");       //Results for ttf distribution
383         constr(str, combin);
384         if ((ff = fopen(str,"w+")) == NULL) {
385             printf("Error!!!");
386             exit(1);
387         }
388         for (i=0; i<pp+1; i++) {
389             if (ii==0 && jj==0 && kk==0) tbs[i] = tbs[i]/(Nc*Nc*Nc);
390             else tbs[i] += tbs[i]/(Nc*Nc*Nc);
391             fprintf(ff,"%Lf\n",tbs[i]);
392         }
393         fclose(ff);
394     }           //for combin
395     it++;
396     printf("\nIteration %u\n", it);     //Only counts the values of C1, C2, C3
397 
```

```
398   }       //for kk
399   }       //for jj
400   }       //for ii
401
402
403
404   for (combin=0; combin<Ncomb; combin++) {
405       strcpy(str, "C:\\Results\\rel");
406       constr(str, combin);
407       if ((ff = fopen(str,"w+")) == NULL) {
408           printf("Error!!!");     exit(1);
409       }
410       for (j = 0; j < 3; j++)
411           fprintf(ff, "%u\n", comb[combin][j]);
412       for (i=0; i<pp_rel; i++) fprintf(ff, "%Lf\n", rel1[combin][i]);
413       fclose(ff);
414   }
415
416   for (i = 0; i < 3; i++) free(cov_factor[i]);
417
418   free(pr_ptr);    free(ttf);      free(tbf);
419   free(tts);       free(tbs);      free(ttr);
420   free(dwt);       free(stops);    free(rel);
421
422   for (i=0; i<35; i++) free (rel1[i]);
423
424
425   return 0;
426   }
427
428   //_____________________________________________________________________________
429   void input (void) {
430   unsigned int i, j;
431   char ans;
432   long double lam;
433
434   stepT = 1.0;
435   adj = stepT/3600;                    //Convert rates to seconds
436   printf("Types of fault:\n");
437   printf("\t-permanent\n");
438   printf("\t LP[1/h] = ");
439   scanf("%Lf", &lam);
440   adjust(lam);
441
442   for (i=0; i<N; i++) pr_ptr[i].lambda[0] = lam;
443   printf("\t-repair\n");
444   printf("\t Mu[1/h] = ");
445   scanf("%Lf", &lam);
446   adjust(lam);
447   for (i=0; i<N; i++) pr_ptr[i].lambda[2] = lam;
448
449   printf("\nIterations: ");
450   scanf("%u", &iter);
451
452   printf("\nTotal reliability: ");
453   scanf("%Lf", &R_total);
454
455   printf("\nMission time: ");
456   scanf("%Lf", &mission_time);
457
458
459   if ((ttf = (long double *) calloc(iter,sizeof(long double))) == NULL) {
460     printf("Not enough memory to allocate buffer\n");
461     exit(1);
462   }
463
464   if ((tbf = (long double *) calloc(iter,sizeof(long double))) == NULL) {
465     printf("Not enough memory to allocate buffer\n");
466     exit(1);
467   }
```

```
468
469   if ((tts = (long double *) calloc(iter,sizeof(long double))) == NULL) {
470     printf("Not enough memory to allocate buffer\n");
471     exit(1);
472   }
473
474   if ((tbs = (long double *) calloc(iter,sizeof(long double))) == NULL) {
475     printf("Not enough memory to allocate buffer\n");
476     exit(1);
477   }
478
479   if ((ttr = (long double *) calloc(iter,sizeof(long double))) == NULL) {
480     printf("Not enough memory to allocate buffer\n");
481     exit(1);
482   }
483
484   if ((dwt = (long double *) calloc(iter,sizeof(long double))) == NULL) {
485     printf("Not enough memory to allocate buffer\n");
486     exit(1);
487   }
488
489   if ((stops = (unsigned int *) calloc(iter,sizeof(unsigned int))) == NULL) {
490     printf("Not enough memory to allocate buffer\n");
491     exit(1);
492   }
493
494   printf("System repair after stop (Y/N)?\n");
495   ans = _getch();
496   ans = toupper(ans);
497   if (ans == 'Y') {
498     printf("Repair rate[1/h] = ");
499     scanf("%Lf", &srepair);   adjust(srepair);
500     printf("\nMax uptime [h]: ");     //Maximum time for system operation in case of
      repairable system in hours
501     scanf("%Lf", &uptime);    uptime = uptime/adj;
502   }
503   else {
504       srepair = 0.0;      uptime = 0.0;
505   }
506
507   maxt = pr_ptr[0].lambda[0];
508
509   for (i = 0; i<N; i++)
510       if (pr_ptr[0].lambda[i] < maxt && pr_ptr[0].lambda[i] != 0.0)   //according to PF
      rate
511           maxt = pr_ptr[i].lambda[0];
512   if (srepair < maxt && srepair != 0.0)                          //looking for the minimum
      rate
513       maxt = srepair;
514
515
516    maxt = 3.0e5;
517   }
518
519   //_____________________________________________________________________________
520   void init(void) {
521   unsigned int i, j;
522
523   if (N == 3) {
524       pr_ptr[0].lastC = 92;   pr_ptr[1].lastC = 56;   pr_ptr[2].lastC = 17;
525   }
526   else if (N == 5) {
527       pr_ptr[0].lastC = 92;   pr_ptr[1].lastC = 56;   pr_ptr[2].lastC = 17;
528       pr_ptr[3].lastC = 111;  pr_ptr[4].lastC = 865;
529   }
530   else if (N == 10) {
531       pr_ptr[0].lastC = 92;   pr_ptr[1].lastC = 56;   pr_ptr[2].lastC = 17;
532       pr_ptr[3].lastC = 111;  pr_ptr[4].lastC = 865;  pr_ptr[5].lastC = 235;
533       pr_ptr[6].lastC = 8;    pr_ptr[7].lastC = 1354; pr_ptr[8].lastC = 999;
534       pr_ptr[9].lastC = 736;
```

```
535   }
536   else if (N == 20) {
537       pr_ptr[0].lastC = 92;   pr_ptr[1].lastC = 56;       pr_ptr[2].lastC = 17;
538       pr_ptr[3].lastC = 111;  pr_ptr[4].lastC = 865;      pr_ptr[5].lastC = 235;
539       pr_ptr[6].lastC = 8;    pr_ptr[7].lastC = 1354;     pr_ptr[8].lastC = 999;
540       pr_ptr[9].lastC = 736;  pr_ptr[10].lastC = 3214;    pr_ptr[11].lastC = 12;
541       pr_ptr[12].lastC = 352; pr_ptr[13].lastC = 34;      pr_ptr[14].lastC = 1040;
542       pr_ptr[15].lastC = 62;  pr_ptr[16].lastC = 7518;    pr_ptr[17].lastC = 35;
543       pr_ptr[18].lastC = 205; pr_ptr[19].lastC = 222;
544   }
545
546
547   for (i=0; i<N; i++) {
548     pr_ptr[i].state = normal;
549     pr_ptr[i].fault = none;
550     pr_ptr[i].curr_proc = false;
551     for (j=0; j<3; j++)
552       pr_ptr[i].ftime[j] = 0.0;
553   }
554   }
555
556   //_____________________________________________________________________________
557   void fault_time(void) {
558   unsigned int i, j;
559   unsigned long dum = 1;
560
561   for (i=0; i<N; i++) {
562       if (itr == 0) {
563           pr_ptr[i].lastf[0] = i+1;
564           pr_ptr[i].lastf[2] = i+3;
565       }
566
567     for (j=0; j<3; j++)
568       if (pr_ptr[i].lambda[j] != 0.0 && j!=2) {
569           pr_ptr[i].ftime[j] += time_val(pr_ptr[i].lastf[j],pr_ptr[i].lambda[j]);
570           if (pr_ptr[i].ftime[j] == 0.0)
571           pr_ptr[i].ftime[j] += time_val(pr_ptr[i].lastf[j],pr_ptr[i].lambda[j]);
572       }
573   }
574   }
575
576   //_____________________________________________________________________________
577   void el_faults(faults &el, unsigned int ft) {
578
579   switch (ft) {
580     case 1: el = permanent; break;
581     case 2: el = transient; break;
582     case 3: el = repair; break;
583   }
584   }
585
586
587   //_____________________________________________________________________________
588   long double min_time(void)
589   {
590   unsigned int i, j;
591   long double min;
592
593   for (i=0; i<N; i++)
594   //  for (j=0; j<3; j++)
595       if (pr_ptr[i].ftime[0] != 0.0) {
596           min = pr_ptr[i].ftime[0];  break;
597       }
598
599   for (i = 0; i<N; i++)
600     for (j=0; j<3; j++)
601       if (pr_ptr[i].ftime[j]<min && pr_ptr[i].ftime[j] != 0.0)
602         min = pr_ptr[i].ftime[j];
603
604   for (i=0; i<N; i++) {
```

```
    for (j=0; j<3; j++)
      if (pr_ptr[i].ftime[j] == min) {
        pr_ptr[i].curr_proc = true;
        el_faults(pr_ptr[i].fault,j+1);
        break;
      }
  }
  return (min);
  }


  void el_coverage(void) {
  long double C, rangeCmin, rangeCmax;
  int decimal_digits;
  double y;
  unsigned int i;

  decimal_digits = 4;
  for (i=0; i<3; i++) {
      if ((i+1) == 1) {rangeCmin = 0.8; rangeCmax = 0.9;}
      else if ((i+1) ==2) {rangeCmin = 0.9; rangeCmax = 0.94;}
      else if ((i+1) == 3) {rangeCmin = 0.97; rangeCmax = 1.0;}

  do {
      C = urande(last_cov);
  } while (C<rangeCmin || C>=rangeCmax);
  y = pow(10.0,decimal_digits);
  C *= y;
  C = (unsigned int)(C)/y;
  Cmod[i] = C;
  printf("%u module(s) C[%u] = %Lf\n", i+1, i+1, Cmod[i]);
  }
  }


  void el_state(unsigned int i) {
  long double C;
  unsigned long lastCov;

  if (pr_ptr[i].fault == repair) pr_ptr[i].state = detected;
  else {
      if (pr_ptr[i].state != detected) {
          if (pr_ptr[i].C0 != 0.0) {
              if (pr_ptr[i].C0 == 1.0) pr_ptr[i].state = detected;
              else {
                  lastCov = pr_ptr[i].lastC;
                  C = urande(lastCov);
                  pr_ptr[i].lastC = lastCov;
                  if (C <= pr_ptr[i].C0) pr_ptr[i].state = detected;
                  else pr_ptr[i].state = undetected;
              }
          }
          else pr_ptr[i].state = undetected;
      }
  }
  }

  //_____________________________________________________________________________
  void st_count(unsigned int num[]) {
  unsigned int i;

  for (i=0; i<3; i++) num[i] = 0;
  for (i=0; i<N; i++)
    switch (pr_ptr[i].state) {
      case normal : num[0]++;  break;
      case detected : num[1]++;  break;
      case undetected : num[2]++;  break;
    }
  }
```

```
675  //_____________________________________________________________________________
676  sys_states system_state(void) {
677  div_t x;
678
679  x = div(N,2);
680  if (cnt[0] > x.quot) return(snormal);           //majority normal
681  else if (cnt[2] > cnt[1]) return(failure);      //majority undetected
682       else return(stop);                         //majority detected
683  }
684
685
686  //_____________________________________________________________________________
687  void fault_repair(unsigned int k, unsigned int jj) {
688  //unsigned long last_repair;
689
690
691  if (pr_ptr[k].lambda[2] != 0.0) {
692    pr_ptr[k].fault = repair;
693    pr_ptr[k].ftime[2] = pr_ptr[k].ftime[jj]+time_val(pr_ptr[k].lastf[2],pr_ptr[k].
     lambda[2]); //repair from PF
694    for (unsigned int i=0; i<3; i++)
695      if (i!=2) {
696        pr_ptr[k].ftime[i] = 0.0;
697      }
698  }
699  else                                                //no repair available; detected
     and stopped
700    for (unsigned int i = 0; i<3; i++) {
701      pr_ptr[k].ftime[i] = 0.0;
702    }
703  }
704
705
706  //_____________________________________________________________________________
707  void system_level(unsigned int k) {
708
709  if (pr_ptr[k].state == detected) {
710      switch (pr_ptr[k].fault) {
711          case permanent: fault_repair(k,0);
712               pr_ptr[k].curr_proc = false;
713          break;
714          case repair: if (pr_ptr[k].curr_proc)
715                       el_repair(k);
716          break;
717      }
718  }
719  else if (pr_ptr[k].state == undetected){        //no repair
720      for (unsigned int i=0; i<3; i++)
721          pr_ptr[k].ftime[i] = 0.0;
722      pr_ptr[k].curr_proc = false;
723  }
724  }
725
726  //_____________________________________________________________________________
727  void el_repair(unsigned int j) {
728
729  for (unsigned int i = 0; i<3; i++) {
730      if (pr_ptr[j].lambda[i]!= 0.0 && i!=2) {
731          pr_ptr[j].ftime[i] = pr_ptr[j].ftime[2]+time_val(pr_ptr[j].lastf[i],pr_ptr[j].
           lambda[i]);
732      }
733  }
734
735  pr_ptr[j].ftime[2] = 0.0;
736  pr_ptr[j].curr_proc = false;
737  pr_ptr[j].state = normal;
738  pr_ptr[j].fault = none;
739  }
740
741  //_____________________________________________________________________________
```

```
742  long double sqr(long double x) {
743      return (x*x);
744  }
745
746
747  long double sum_sqr(long double *arr, long double *arr1, unsigned int n) {  //sum of
     squares
748  unsigned int i;
749  long double temp;
750
751  temp = 0.0;
752  for (i = 0; i<n; i++) temp += arr[i]*arr1[i];
753  return (temp);
754  }
755
756
757  void stats(long double *arr, long double &stdev, long double summ, unsigned int n,
     long double &conf) {
758  long double sum_sq, variance, stoch_err;
759
760  sum_sq = sum_sqr(arr, arr, n);
761  variance = (n*sum_sq - sqr(summ))/(n*(n-1));            //according to the textbook
     of Kaloyanov
762  stdev = sqrt(variance);
763  stoch_err = stdev/sqrt((long double)n);                 //margin of error or
     stochastic error
764  conf = 1.96*stoch_err;                                  //confidence for 95%
     probability
765  }
766
767
768  void modules (unsigned int maxM, unsigned int &appr) {
769  unsigned int i, j, k, z;
770
771  z = 0;  appr = 0;
772  for (i=0; i<N+1; i++) {
773      for (j=0; j<N+1; j++) {
774          for (k=0; k<N+1; k++) {
775              if (i+2*j+3*k == maxM && i+j+k == N) {
776                  appr++;
777                  comb[z][0] = i; comb[z][1] = j; comb[z][2] = k;
778                  z++;
779              }
780          }
781      }
782  }
783
784  for (i = 0; i < appr; i++)
785      if ((node_mod[i] = (unsigned int*)calloc(N, sizeof(unsigned int))) == NULL) {
786          printf("Not enough memory to allocate buffer\n");
787          exit(1);
788      }
789
790  for (z=0; z<appr; z++) {            //All appropriate combinations
791      for (i=0; i<comb[z][0]; i++)
792          node_mod[z][i] = 1;
793      for (i=comb[z][0]; i<comb[z][0]+comb[z][1]; i++)
794          node_mod[z][i] = 2;
795      for (i=comb[z][0]+comb[z][1]; i<N; i++)
796          node_mod[z][i] = 3;
797  }
798  }
799
800  //_____________________________________________________________________________
801  void constr(char *str, unsigned int i) {
802  char str1[120];
803
804      sprintf(str1, "%d", i);
805      strcat(str, str1);
806      strcat(str, ".dat");
```

```
807   }
808
809
810   //_____________________________________________________________________________
811   //_____________________________________________________________________________
812   //Program for identifying the configurations that satisfy Rtotal
813
814
815   #include <iostream>
816   using std::cerr;
817   using std::cout;
818   using std::endl;
819   #include <fstream>
820   using std::ifstream;
821   using std::ofstream;
822   #include <cstdlib> // for exit function
823   #include <string>
824
825   long double R_total;
826   int points, i, Ncomb, combin, Max_points;
827   std::string str = "";
828   std::string str1 = "";
829
830   int main()
831   {
832       ifstream indata;
833       long double num1, num2; // variables for 2 input values
834       ofstream outdata;
835
836       printf("Rtotal = ");
837       scanf_s("%Lf", &R_total);
838       printf("Max points = ");
839       scanf_s("%d", &Max_points);
840
841       Ncomb = 11;
842
843       for (combin = 0; combin < Ncomb; combin++) {
844           str = "C://Results/rel" + std::to_string(combin) + ".dat";
845           str1 = "C://Results/combin" + std::to_string(combin) + ".dat";
846
847           indata.open(str); // opens the file for reading
848           if (!indata) {
849               cerr << "Error: file could not be opened" << endl;
850               exit(1);
851           }
852           outdata.open(str1); // opens the file for writing
853           if (!outdata) {
854               cerr << "Error: file could not be opened" << endl;
855               exit(1);
856           }
857           i = 0;  points = 0;
858           if (indata && outdata) {
859               indata >> num1;
860               while (!indata.eof()) {
861                   if (i<3) outdata << num1 << endl;
862                   else {
863                       indata >> num2;
864                       if ((num1 >= R_total) || (points<=Max_points)) {
865                           cout << "Value above Rtotal " << num1 << endl;
866                           points++;
867                           outdata << num1 << " " << num2 << endl;
868                       }
869                   }
870                   i++;
871                   indata >> num1;
872               }
873               cout << "Points = " << points << endl;
874               indata.close();
875               cout << "End-of-IN-file reached." << endl;
876           }
```

```
877             outdata.close();
878             cout << "End-of-OUT-file reached." << endl;
879         }
880 
881         return 0;
882     }
```