ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут економіки, управління, права та інформаційних технологій
Кафедра інформаційних систем та технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти магістр

на тему: **«Імітаційна модель функціонування інформаційної системи в умовах атак на її готовність»**

Виконав: здобувач вищої освіти
за освітньою програмою
Інформаційні управляючі системи та технології
спеціальності 126 Інформаційні системи та технології
ступеня вищої освіти магістр
групи 126ІСТ_мд_2023
Шпіньов Павло Віталійович
Керівник: Уткін Юрій Вікторович
Рецензент: Муравльов Володимир Вячеславович

Полтава – 2024 року

ВСТУП

Актуальність теми. Протягом останніх десятиліть хмарні технології стали однією з найдинамічніших галузей IT, адже вони фактично перетворилися на стандарт для інтеграції різних додатків. Інформаційні системи (ІС) та вебдодатки нині широко застосовуються в різних сферах життя та промисловості, зокрема у системах бронювання та продажу туристичних послуг, інтернет-магазинах, онлайн-банкінгу, електронному бізнесі та інших галузях [1].

Платформа Windows Azure є моделлю PaaS (Платформа як Сервіс), яка дозволяє запускати додатки на серверах, що під'єднані до мережевої інфраструктури у центрах обробки даних Microsoft і мають доступ до Інтернету [2].

Сучасні плани впровадження хмарних архітектур відповідають новим викликам у моделюванні та оцінці їх готовності й доступності. Недоліками існуючих підходів є складність реалізації архітектури для бізнес-критичних застосувань, відсутність уніфікованих процедур забезпечення відмовостійкості та моделей для оцінки надійності інформаційних систем.

Розвитком методів і моделей оцінювання надійності та готовності інформаційних систем упродовж останніх десятиліть активно займалися дослідники, зокрема В. Харченко [3], А. Горбенко [4], К. Тріведі [1] та А. Боярчук [6]. У їхніх дослідженнях аналізуються аналітичні моделі інформаційних систем у контексті спеціалізованих засобів моделювання.

Зв'язок роботи з науковими програмами, темами. Робота відповідає дослідженням в межах науково-дослідної ініціативної тематики «Організаційно-методологічні аспекти впровадження інформаційно- комунікаційних систем і технологій в управлінні діяльністю сучасних організацій та підприємств за умов переходу до цифрової економіки» (ДРН 0123U105060, 2023-2028 рр.), що реалізується на кафедрі інформаційних систем та технологій, тематиці досліджень навчально-дослідної лабораторії інтелектуальних систем, комп'ютерних мереж та інтернет речей кафедри інформаційних систем та технологій Полтавського державного аграрного університету.

Метою кваліфікаційної роботи є оцінювання адекватності результатів аналітичного моделювання функціонування інформаційних систем та розробка рекомендацій щодо застосування імітаційних моделей.

Завданнями кваліфікаційної роботи є:

– аналіз існуючих методів і засобів розробки, моделювання, оцінки характеристик і забезпечення надійності та кібербезпеки інформаційних систем архітектури Microsoft Azure Web Sites;

– розробка і дослідження марковської моделі для оцінки готовності ІС;

– розробка і дослідження імітаційної моделі функціонування ІС з врахуванням атак на її компоненти.

Об'єктом дослідження є процеси моделювання функціонування інформаційних систем в умовах прояву відмов та атак на їх компоненти.

Предметом дослідження є аналітичні та імітаційні моделі оцінки готовності інформаційних систем.

Методи дослідження – проведені в роботі дослідження базуються на методах теорії ймовірності, системного і марковського аналізу, систем масового обслуговування, які використовувалися при розробці комплексу марковських та імітаційних моделей інформаційних систем.

Інформаційна база кваліфікаційної роботи складається з наукових статей, міжнародних аналітичних видань і звітів, матеріалів наукових конференцій інтернет-ресурсів, що місять інформацію про моделі функціонування інформаційних систем в умовах атак, чинні нормативні документи, що регулюють вимоги до надійності та кібербезпеки інформаційних систем.

Елементи наукової новизни полягають у дослідженні аналітичних та імітаційних моделей функціонування інформаційних систем в умовах прояву дефектів апаратних і програмних засобів, атак на компоненти архітектури Microsoft Azure Web Sites, проведення оновлень програмного забезпечення, загальних та роздільних обслуговувань.

Практична значущість роботи полягає в можливості повторного застосування та модифікації розроблених моделей. Отримані результати можуть бути корисними для ІТ фахівців при моделюванні розподілених інформаційних систем.

Апробація результатів дослідження відбувалася шляхом оприлюднення доповідей на наукових конференціях, семінарах.

Публікації. За результатами проведеного дослідження опубліковано тези: «Методи аналізу трафіку в корпоративних комп'ютерних мережах», Матеріали науково-практичної конференції за підсумками проходження виробничої практики здобувачів вищої освіти ступеня вищої освіти «Магістр», 16 жовтня 2024 року, Полтава; «Імітаційне моделювання в галузі кібербезпеки та застосування систем диференціальних рівнянь для оцінки стійкості інформаційних систем»: Студентські роботи за науковою тематикою кафедри інформаційних систем та технологій: матеріали XXI щорічного міждисциплінарного семінару, 20 листопада 2024 р. Полтава.

Структура та обсяг кваліфікаційної роботи логічно пов’язані з задачами досліджень. Робота містить перелік умовних позначень, вступ, три розділи основної частини, висновки, список використаних джерел, додатки. Загальний обсяг текстової частини дипломної роботи складає 62 сторінки формату А4. Вона містить 25 рисунків і 3 таблиці. У роботі використано 45 науково-технічних джерел.

РОЗДІЛ 1
АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ І ЗАСОБІВ РОЗРОБКИ ТА ОЦІНКИ ГОТОВНОСТІ АРХІТЕКТУРИ MICROSOFT AZURE

1.1 Архітектура Windows Azure Platform

Платформа Windows Azure є прикладом моделі PaaS (Платформа як Сервіс), яка дозволяє запускати додатки на серверах, інтегрованих у мережеву інфраструктуру, розташовану в центрах обробки даних Microsoft, з доступом до Інтернету. Azure включає масштабовану операційну систему, фабрику зберігання даних та додаткові сервіси для доставки через фізичні або логічні (віртуалізовані) екземпляри Windows Server [9]. Інструментарій для розробників Windows Azure (SDK) забезпечує можливості для створення хмарних сервісів. Інструменти та прикладні інтерфейси програмування (API), необхідні для розробки, розгортання та управління масштабованими сервісами в середовищі Windows Azure, включають шаблони додатків Azure для різних версій Visual Studio.

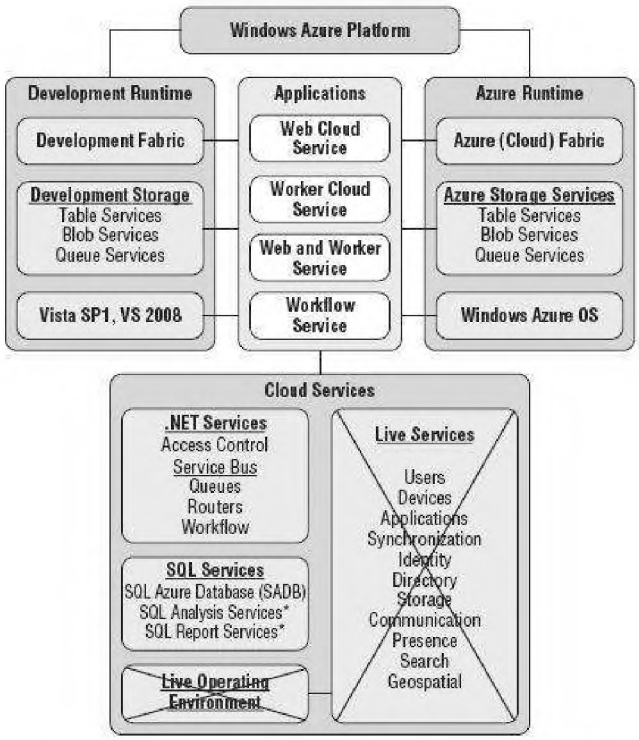


Рисунок 1.1 – Компоненти платформи Windows Azure

На рис. 1.1 представлені основні компоненти хмарної платформи, а також інструменти для розробників. Microsoft стала однією з провідних компаній, яка займалася розробкою стандарту SOAP [10]. У 2002 році, разом із випуском ASP.NET 1.0, було представлено технологію ASMX (Active Server Method Extended), яка дозволила розробникам у Visual Studio легко створювати та запускати веб-сервіси на основі SOAP. На порталі MSDN ця технологія відома під назвою XML Web Services [11]. У ті часи SOAP лише починав набирати популярність у веб-розробках. W3C затвердив SOAP 1.1 у 2000 році, а SOAP 1.2 – у 2003, з подальшими доповненнями у 2007 році. Було важливо створити технологію, яка була б простою для освоєння та застосування, і Microsoft досягла цієї мети. Завдяки ASMX розробникам навіть не потрібно було знати XML, SOAP або WSDL для роботи з веб-сервісами [12].

Протягом наступних років технологія ASMX стала дуже популярною та отримала широке визнання. Microsoft також із самого початку постачала додатковий пакет Web Services Enhancements (WSE), що дозволяв впроваджувати різні специфікації безпеки WS-*, такі як WS-Security, WS-Policy та WS-ReliableMessaging. Остання версія WSE 3.0 вийшла у 2005 році. У 2007 році разом із виходом .NET 3.0 Microsoft представила Windows Communication Foundation (WCF), яка офіційно замінила ASMX. Попри те, що ASMX більше не розвивається, вона продовжує підтримуватися останніми версіями .NET Framework і все ще активно використовується [13].

Платформа Azure дозволила розробникам .NET, починаючи з Visual Studio 2008 та вище, створювати ASP.NET веб-додатки та сервіси Windows Communication Framework (WCF). Веб-додатки розгортаються в ізольованому середовищі Internet Information Services (IIS). Веб-програми та сервіси працюють у частково довіреному середовищі безпеки, яке обмежує доступ коду до ресурсів комп'ютера за допомогою механізму Code Access Security, що відповідає середньому рівню довіри. Це середовище обмежує доступ до певних ресурсів операційної системи. Комплект розробника Windows Azure дозволяє мати повний

доступ до ресурсів комп'ютера для виконання .NET-коду та використання бібліотек, що вимагають повної довіри. Також доступна обробка взаємодій за допомогою програмних каналів (Pipe). Microsoft також обіцяє підтримку для запуску програм на Ruby, PHP та Python на своїй хмарній платформі [14].

Спочатку платформа Azure була обмежена середовищем програмування Visual Studio 2008 та вище, але згодом була запланована підтримка інструментів Eclipse. Платформа підтримує веб-стандарти та протоколи, такі як SOAP, HTTP, XML, Atom та AtomPub.

Для розробників точкою входу для створення та розміщення ASP.NET додатків у хмарі є портал Windows Azure. Для доступу до порталу необхідно увійти за допомогою облікового запису Windows Live ID. Версії платформи Azure (Community Technical Previews, CTPs) вимагали різних токенів для [15]:

1. Windows Azure, що включає: – Azure Hosted Service; – Storage Accounts;

2. SQL Azure Live служб, що включають Live Framework (попередня версія) та Live Services (існуючі API).

З початку 2009 року токени Windows Azure надають право на один обліковий запис Hosted Service та два облікові записи для Storage. Сховище Windows Azure Storage забезпечує розробникам можливість зберігати дані в хмарі. Програми можуть отримувати доступ до даних у будь-який час і з будь-якої точки світу. Обсяги збережених даних можуть бути необмеженими, а дані зберігаються надійно, без ризику пошкодження або втрати. Windows Azure Storage пропонує широкий набір абстракцій даних [16]:

– Windows Azure Table для зберігання структурованих станів сервісу,

– Windows Azure Blob для зберігання великих даних,

– Windows Azure Queue для диспетчеризації асинхронних завдань і реалізації обміну даними між сервісами.

Таким чином, платформа Windows Azure забезпечує потужні можливості для розробників у створенні масштабованих і безпечних хмарних сервісів з використанням сучасних стандартів і технологій.

1.2 Аналіз хмарної архітектури із застосуванням Microsoft Azure Web Sites

Azure Web Sites пропонує кілька варіантів хостингу: Free, Shared, Basic та Standard. Плани Free та Shared використовують спільну інфраструктуру, тобто ресурси вебсайтів розподіляються між кількома користувачами. Натомість плани Basic і Standard забезпечують виділену інфраструктуру для кожного сайту [17].

На цих рівнях можна налаштувати індивідуальний план хостингу з використанням одного або кількох екземплярів віртуальних машин (VM). Ці плани підтримують різні типи екземплярів: малі, середні та великі. Постачальник послуг Azure керує цими віртуальними машинами, тому не потрібно турбуватися про оновлення операційної системи, безпеку чи конфігурацію.

Для запуску інформаційних систем у середовищі виробничого рівня в Azure Web Sites рекомендується використовувати плани Basic або Standard, в залежності від масштабів додатка і обсягу трафіку, який надходить на вебсайт. На рисунку 1.2 показано, як можна розробити архітектуру вебсайту та залежних від нього компонентів [18].

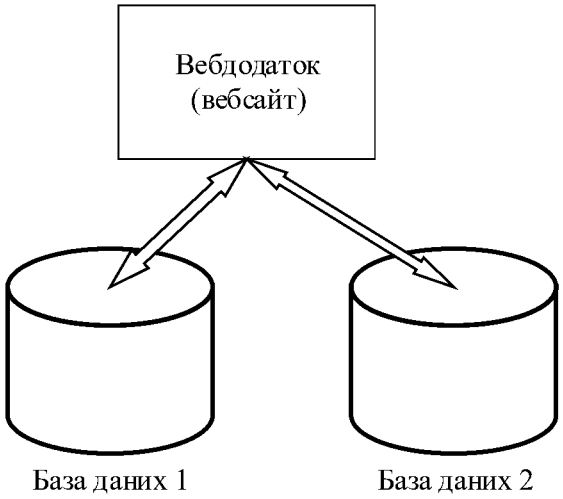


Рисунок 1.2. Типова архітектура вебсайту

Типова архітектура вебсайту в Azure Web Sites включає вебдодаток, базу даних, сховище Azure і форму кешування. Всі ці компоненти взаємодіють таким чином, щоб уникнути ситуації, коли одна з них стає єдиною точкою збою (single

point of failure, SPOF) [19]. Це один з головних принципів проектування хмарних рішень. Azure Web Sites гарантує рівень доступності (SLA) 99,9%, що означає, що можна очікувати до 10 хвилин простою на тиждень через планові оновлення чи розгортання. Однак навіть такі короткі простої можуть суттєво вплинути на бізнес. Тому важливо проектувати систему таким чином, щоб мінімізувати вплив простоїв і забезпечити стабільне обслуговування клієнтів, знижуючи можливі ризики. Приклад вебархітектури з високою доступністю наведений на рисунку 1.3 [20].

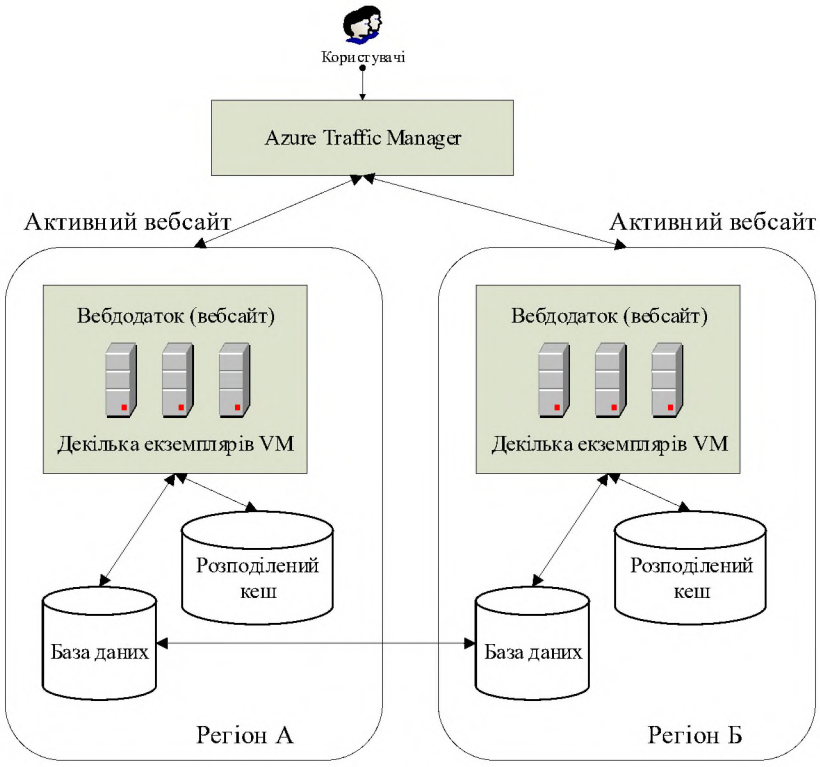


Рисунок 1.3. Приклад вебархітектури з високою доступністю

Azure Web Sites забезпечує хмарне рішення для інформаційних систем без підтримки станів. Тому при розробці архітектури високої доступності для інформаційних систем потрібно враховувати кілька важливих аспектів [21]:

1. Використовувати режим Standard для вебсайтів і налаштувати роботу мінімум двох екземплярів вебсайту.

2. В залежності від передбачуваного трафіку імітувати навантаження за допомогою інструментів тестування, таких як Visual Studio або Apache JMeter

(jmeter.apache.org), для визначення оптимального розміру та кількості екземплярів, необхідних для стабільної роботи вебсайту.

3. Централізувати зберігання власних та сеансових даних у базі даних або розподіленому кеші. У режимі Standard усі віртуальні машини в Azure Web Sites використовують один файловий сервер.

4. Реплікувати рівень вебдодатка щонайменше у два різні регіони, підтримувані Azure Web Sites.

5. Користувацький контент, такий як медіафайли та документи, що завантажуються на сайт, слід зберігати на акаунті Azure Storage. Важливо переконатися, що цей обліковий запис знаходиться у тому ж регіоні, що й вебсайт, для зменшення затримок.

6. Завжди дотримуватися принципів безпечного кодування, щоб захистити вебдодаток від можливих атак.

Дотримуючись цих рекомендацій, можна забезпечити високу доступність та надійність інформаційних систем, мінімізуючи ризики простоїв і покращуючи якість обслуговування клієнтів.

1.3 Визначення мікросервісів та їх застосування

Мікросервіси стали популярним підходом до створення програмних систем, і хоча існує багато визначень цього терміна, в Інтернеті можна знайти безліч ресурсів з різними точками зору. Однак можна виділити кілька ключових характеристик мікросервісів, які часто згадуються [22]. По-перше, мікросервіси інкапсулюють конкретні клієнтські або бізнес-сценарії. Вони зазвичай розробляються невеликими командами, які можуть використовувати будь-які мови програмування та платформи, що надає гнучкість у виборі технологій. Кожен компонент, що входить до складу мікросервісу, містить код і, за потреби, стан, який можна окремо розгортати, масштабувати і керувати його версіями. Мікросервіси взаємодіють між собою через чітко визначені інтерфейси та протоколи, маючи

унікальні імена (наприклад, у формі URL-адрес), які дозволяють визначити їхнє розташування в системі. До того ж вони забезпечують стабільність та доступність навіть у разі збоїв.

Монолітні додатки, хоча й мають свої переваги, часто не можуть забезпечити необхідну гнучкість та масштабованість. При монолітному підході всі компоненти додатка тісно пов'язані між собою на різних рівнях, що дозволяє швидко здійснювати виклики між компонентами. Розробка таких додатків зазвичай вимагає менше зусиль, оскільки всі користувачі тестують один продукт, що дозволяє ефективніше використовувати ресурси. Проте головним недоліком є неможливість масштабувати окремі частини додатка і залежність від тестування всього продукту, що може затримувати впровадження змін [23].

Мікросервіси вирішують ці проблеми, дозволяючи створювати додатки, які більш точно відповідають бізнес-вимогам. Основними перевагами мікросервісів є можливість незалежного масштабування, тестування, розгортання і управління окремими частинами системи. Кожна мікрослужба зазвичай представляє просту бізнес-функцію і може масштабуватися незалежно від інших служб. Завдяки цьому команди розробників можуть зосередитися на реалізації бізнес-сценаріїв, не витрачаючи багато часу на узгодження технологічних рішень. Це особливо корисно під час розробки багаторівневих систем, коли невеликі команди створюють мікросервіси відповідно до клієнтських сценаріїв, використовуючи ті технології, які вважають найбільш доцільними.

У цьому підході відсутня потреба в стандартизації технологій на рівні всієї організації. Кожна команда може самостійно вирішувати, які інструменти і платформи використовувати, базуючись на власному досвіді і знаннях. Водночас рекомендується використовувати набір рекомендованих технологій, наприклад, для вибору сховищ NoSQL або платформ веб-додатків. Це дозволяє уникнути технологічної фрагментації, зберігаючи при цьому гнучкість для кожної окремої команди [24].

Попри значні переваги, мікросервіси мають і певні недоліки. Один із них полягає у зростанні кількості служб, якими потрібно керувати, а також у

складнощах з розгортанням і управлінням версіями. Збільшення мережевого трафіку між мікросервісами може призводити до затримок у передачі даних. Крім того, надмірна кількість вузькоспеціалізованих служб може стати причиною зниження продуктивності системи, якщо їхня робота не оптимізована належним чином. Брак засобів для виявлення залежностей між мікросервісами може ускладнити загальний огляд системи, що робить її обслуговування складнішим.

Щоб забезпечити ефективну роботу мікросервісів, важливо дотримуватися стандартів обміну даними, а не суворих контрактів, яких потрібно дотримуватися беззаперечно. Під час розробки таких систем важливо заздалегідь визначити правила обміну даними між службами, оскільки кожна служба може оновлюватися незалежно. Це дозволяє підвищити гнучкість і забезпечити можливість незалежного розвитку кожної частини системи.

Однією з важливих характеристик мікросервісної архітектури є її сервісно-орієнтована природа (SOA) [25], що дозволяє розробляти системи з невеликих, незалежно керованих модулів. Кожна мікрослужба може бути змінена без впливу на інші частини системи, що значно спрощує її обслуговування і масштабування. У процесі створення додатків для хмарних платформ стає очевидним, що розбиття великих монолітних систем на окремі мікросервіси є оптимальним підходом, який забезпечує гнучкість і масштабованість у довгостроковій перспективі.

Порівняння монолітних додатків та мікросервісів можна описати так: монолітні системи зазвичай складаються з функціональних блоків, що тісно пов'язані між собою. Вони часто поділяються на рівні, такі як мережевий, бізнес-рівень та рівень даних. Масштабування монолітних додатків зазвичай досягається шляхом їх клонування на кілька серверів, віртуальних машин або контейнерів. На відміну від цього, мікросервіси дозволяють масштабувати окремі служби незалежно одна від одної, створюючи екземпляри цих служб на різних серверах чи в контейнерах [26].

Підхід із використанням мікросервісів не є універсальним і залежить від конкретних вимог проекту. У деяких випадках доцільніше використовувати монолітний підхід, наприклад, коли немає можливості переписати код на етапі

впровадження мікросервісів. Проте найчастіше розробники починають із монолітної архітектури, а з часом поступово розділяють її на мікросервіси, починаючи з тих функціональних блоків, що потребують найбільшої гнучкості та масштабованості.

Таким чином, використання мікросервісної архітектури дозволяє створювати додатки з кількох невеликих, незалежно керованих служб, що запускаються в контейнерах і розгортаються в кластерах серверів. Кожна мікрослужба відповідає за окремий бізнес-сценарій і може тестуватися, розгортатися та масштабуватися незалежно від інших служб, забезпечуючи гнучкість у розвитку додатка в цілому.

1.4 Аналіз загроз для надійності та кібербезпеки інформаційних систем

Загрози для надійності та безпеки інформаційних систем можна поділити на кілька ключових аспектів: помилки, несправності та відмови. Помилки є станами системи, що можуть призвести до її відмови. Відмова відбувається, коли помилка потрапляє до системного коду, змінюючи його поведінку. Несправність може бути причиною виникнення помилки, при цьому вона може бути або прихованою, або активною, якщо безпосередньо спричиняє помилку.

Несправності можна класифікувати на три основні групи [2]: ті, що виникають через помилки розробки, фізичні несправності та несправності, пов'язані з взаємодією. До несправностей, які виникають внаслідок зловмисних дій, належать ті, що спричинені шкідливим програмним забезпеченням (троянські коні, віруси, хробаки), а також прямі атаки на систему. Зловмисники можуть скористатися внутрішніми недоліками системи або здійснювати атаки через зовнішні фактори, наприклад, перехоплення трафіку чи відключення живлення. У контексті моделювання систем важливо враховувати наявність несправностей, які виникають через такі атаки.

Для забезпечення стабільної роботи інформаційних систем критично важливо уникати єдиної точки відмови, зокрема на рівні баз даних. Платформа Azure Web Sites підтримує роботу з Azure SQL Database та сервісом MySQL

ClearDB, які пропонують рішення для різних типів навантажень – від простих до преміум-рівня [29].

Azure SQL Database Premium забезпечує стабільнішу продуктивність і більше ресурсів для хмарних програм, що використовують платформу Azure Web Sites. Окрім цього, вона пропонує фіксовані резерви ємності для баз даних, що включають вбудовані засоби реплікації. Такі резерви допомагають вирішувати кілька критичних завдань:

– пікове навантаження: вебдодатки, що потребують значних ресурсів процесора, пам'яті або інтенсивного введення-виведення, можуть використовувати базу Premium для стабільної роботи;

– багато одночасних запитів: стандартні редакції баз даних Azure SQL Database мають обмеження до 180 одночасних запитів? для програм, що потребують більшої кількості підключень, слід використовувати Premium-варіанти з додатковими ресурсами для обробки запитів;

– мінімальна затримка: критичні вебдодатки мають гарантувати швидкий відгук бази даних, що забезпечується Premium-версією, яка дозволяє повертати результат запиту за 20 мс у 99% випадків.

Резервні архітектури, які використовують кілька регіонів, можна поділити на два типи: «активний-активний» і «активний-пасивний». У конфігурації «активний-активний» кілька вебсайтів у різних регіонах одночасно обслуговують один додаток, розподіляючи між собою трафік. У режимі «активний-пасивний» один вебсайт виконує основні функції, тоді як інший залишається резервним і починає роботу лише у випадку збою на основному сайті, що значно знижує наслідки відмови.

1.5 Аналіз математичного апарату для моделювання функціонування інформаційних систем

Відомо кілька методів моделювання інформаційних систем, серед яких експериментальне тестування реальних сервісів, метод Монте-Карло, аналітичний підхід на основі Байєсових моделей та Марковські процеси. Далі розглянемо

детальніше кожен із цих методів та математичні моделі, що використовуються для опису функціонування інформаційних систем.

Системи масового обслуговування (СМО) є однією з основних математичних моделей для аналізу процесів взаємодії запитів у ІС. У таких системах запити користувачів називаються вимогами, а елементи, що обслуговують ці запити, позначаються як канали обслуговування або обслуговуючі пристрої. Головна мета СМО полягає в задоволенні вимог, тобто виконанні запитів користувачів [31].

Залежно від кількості обслуговуючих пристроїв, системи масового обслуговування поділяються на одноканальні (з одним обслуговуючим пристроєм) та багатоканальні (із кількома пристроями). У багатоканальних системах пристрої можуть мати різну продуктивність.

Щодо часу очікування на обслуговування, СМО класифікуються на такі групи [32]:

1. Системи з необмеженим часом очікування, де заявки можуть перебувати в черзі, поки не будуть оброблені.

2. Системи з відмовами, де заявки відхиляються, якщо всі обслуговуючі пристрої зайняті.

3. Системи зі змішаним типом, де встановлюються обмеження на довжину черги або час перебування заявки в системі.

Крім того, системи можуть працювати за певною дисципліною обслуговування, коли заявки обробляються за чергою або відповідно до пріоритетів.

Основними елементами СМО є:

– Вхідний потік вимог: потік запитів користувачів на обслуговування.

– Черга вимог: система накопичення заявок, що очікують на обробку.

– Канали обслуговування: пристрої або сервіси, що виконують запити.

– Вихідний потік: результат виконання запитів.

Марковські випадкові процеси є ефективним методом для вирішення проблеми розмірності розподілених систем обслуговування. Використання цього підходу дає змогу моделювати складні системи інформаційних систем, враховуючи

широкий спектр впливів на працездатність та якість обслуговування. В основі методу лежить декомпозиція структури системи на окремі компоненти, що дозволяє детально оцінити різні параметри.

Декомпозиція буває двох типів [33]:

1. Вертикальна декомпозиція, коли система розбивається на функціональні модулі.

2. Функціональна декомпозиція, коли кожен модуль ділиться на структурні елементи.

Підхід передбачає використання вкладених марковських процесів для моделювання складних станів системи. Це дозволяє враховувати композитність структури інформаційних систем, зовнішні впливи та динамічні зміни параметрів. Для отримання коректних результатів аналізу надійності застосовується апарат диференціальних рівнянь, оскільки алгебраїчні рівняння не можуть вирішити проблеми неергодичності марковських систем.

Імітаційне моделювання інформаційних систем у складних умовах, зокрема під впливом різних зовнішніх факторів, часто здійснюється за допомогою методу Монте-Карло [34]. Цей метод застосовують тоді, коли аналітичне моделювання ускладнюється через складність системи. Суть методу полягає в тому, що замість аналітичного опису випадкових явищ проводяться багаторазові імітації випадкових подій, що дозволяє отримати статистичні результати.

Метод Монте-Карло є ефективним при моделюванні систем з великою кількістю елементів, у яких випадкові фактори взаємопов'язані. Окрім того, його використовують для перевірки достовірності результатів, отриманих іншими методами. Основні аспекти побудови імітаційної моделі полягають у виборі показників ефективності, способу формалізації моделі та визначенні ключових параметрів системи.

Аналіз існуючих методів моделювання сервіс-орієнтованих систем показує, що найкращими для таких задач є методи, засновані на марковських або напівмарковських процесах [36]. Ці методи дозволяють систематизувати процес

моделювання, визначаючи множину можливих станів системи, інтенсивності переходів між ними та інші характеристики.

Проте існує ряд проблем, що обмежують використання цього підходу для інформаційних систем. Найбільшим викликом є відсутність уніфікованої технології моделювання, яка б враховувала специфіку різних архітектур сервіс-орієнтованих систем, а також їх поведінку під час відмов або збоїв. Для адекватного опису подібних систем необхідно враховувати складність архітектури, кількість факторів, що впливають на працездатність, та можливість комбінування простих сервісів у більш складні структури.

Для побудови моделей, що точно відображають реальні системи, слід виконати такі дії:

1. Регуляризація структури моделі та впливів на неї.

2. Застосування методів вкладених марковських процесів для опису внутрішніх процесів у складних станах системи.

3. Використання імітаційного моделювання для перевірки результатів і достовірності моделі.

Таким чином, методи математичного моделювання інформаційних систем, зокрема марковські процеси та метод Монте-Карло, є ключовими інструментами для аналізу складних систем. Вони дозволяють враховувати широкий спектр факторів, впливів і динамічних змін, що відбуваються у процесі функціонування інформаційних систем, а також підвищують точність і надійність моделювання.

Висновки до розділу 1

У першому розділі розглянуто основні аспекти архітектури хмарної платформи Windows Azure та її компонентів, що забезпечують створення та підтримку інформаційних систем. Особливу увагу приділено структурі платформи, її інструментам для розробки, а також підтримуваним стандартам і протоколам, таким як SOAP, HTTP, XML та Atom. Проаналізовано історичний розвиток

технологій для вебсервісів, починаючи від ASMX до сучасного Windows Communication Foundation (WCF), що замінив попередні рішення, зберігаючи сумісність із ними. Виконано аналіз хмарної архітектури Azure Web Sites, який охопив доступні варіанти хостингу (Free, Shared, Basic та Standard), їх особливості та рекомендації для налаштування високої доступності та продуктивності інформаційних систем. Детально розглянуто засоби зберігання даних у Windows Azure Storage, зокрема таблиці, блоби та черги, а також їх роль у забезпеченні масштабованості та стійкості до відмов.

Також проаналізовано мікросервісний підхід до розробки програмних систем, його переваги та недоліки. Встановлено, що мікросервісна архітектура надає значну гнучкість у масштабуванні та розгортанні окремих компонентів додатків, хоча її впровадження пов'язане зі зростанням складності управління службами та їх взаємодії.

Розглянуто основні загрози для надійності та кібербезпеки інформаційних систем, включаючи помилки, несправності та відмови. Встановлено, що важливу роль у забезпеченні надійності відіграє уникнення єдиної точки відмови та впровадження резервних архітектур. Описано методи захисту від збоїв, такі як реплікація даних та використання преміум-версій Azure SQL Database для стабільної роботи під час пікових навантажень. Виконано аналіз математичних методів моделювання функціонування інформаційних систем. Детально описано застосування систем масового обслуговування (СМО), марковських процесів і методу Монте-Карло для дослідження надійності інформаційних систем. Встановлено, що комбінація аналітичних і імітаційних підходів є ефективним інструментом для оцінки складних систем, які працюють у динамічних умовах.

Таким чином, розділ закладає основи для подальшого дослідження та моделювання функціонування інформаційних систем з урахуванням сучасних підходів до архітектури, забезпечення надійності та безпеки в умовах використання хмарних технологій.

РОЗДІЛ 2
РОЗРОБКА І ДОСЛІДЖЕННЯ МАРКОВСЬКОЇ МОДЕЛІ ДЛЯ ОЦІНКИ ГОТОВНОСТІ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Принципи і послідовність розробки марковських моделей інформаційних систем

Першим кроком у розробці та дослідженні моделей інформаційних систем є створення базових моделей їх функціонування, зокрема відмовостійких систем. Особливістю цих моделей є можливість врахування різних сценаріїв поведінки ІС у випадку відмов, що дозволяє побудувати типові моделі та оцінити такі показники, як безвідмовність, доступність і економічна ефективність запланованих сервісів.

Для цього необхідно визначити стани роботи ІС та виділити серед них групи станів, такі як готовність, активна готовність, відмова й відновлення. На наступному етапі потрібно встановити взаємозв'язки між цими станами, застосовуючи розмітку переходів між ними.

Далі проводять ідентифікацію станів, що належать до різних груп, і визначають переходи між ними. Для цього аналізується логіка функціонування системи та описується її поведінка на верхньому рівні декомпозиції. Кожен стан аналізується детально, виконуючи його вертикальну декомпозицію.

Об'єднавши всі ці етапи, формують модель системи зі станами, розкладеними до потрібного рівня деталізації. Якщо модель виявляється надто складною, виконують її спрощення, зберігаючи при цьому чутливість до вхідних потоків, впливів і параметрів процедур відновлення. Після цього, позначивши переходи між станами, отримують орієнтований граф функціонування системи.

Для розробки та дослідження марковських моделей оцінки надійності інформаційних систем здійснюється перехід від функціонального графа до марковської моделі, де вершинами є стани системи, а орієнтованими дугами – переходи між ними. Якщо умови марковості порушуються, проводять додаткову декомпозицію станів. На основі раніше проведеного аналізу визначають

інтенсивності переходів між усіма станами. Таким чином, формується розмічений марковський граф системи з дискретними станами та безперервним часом.

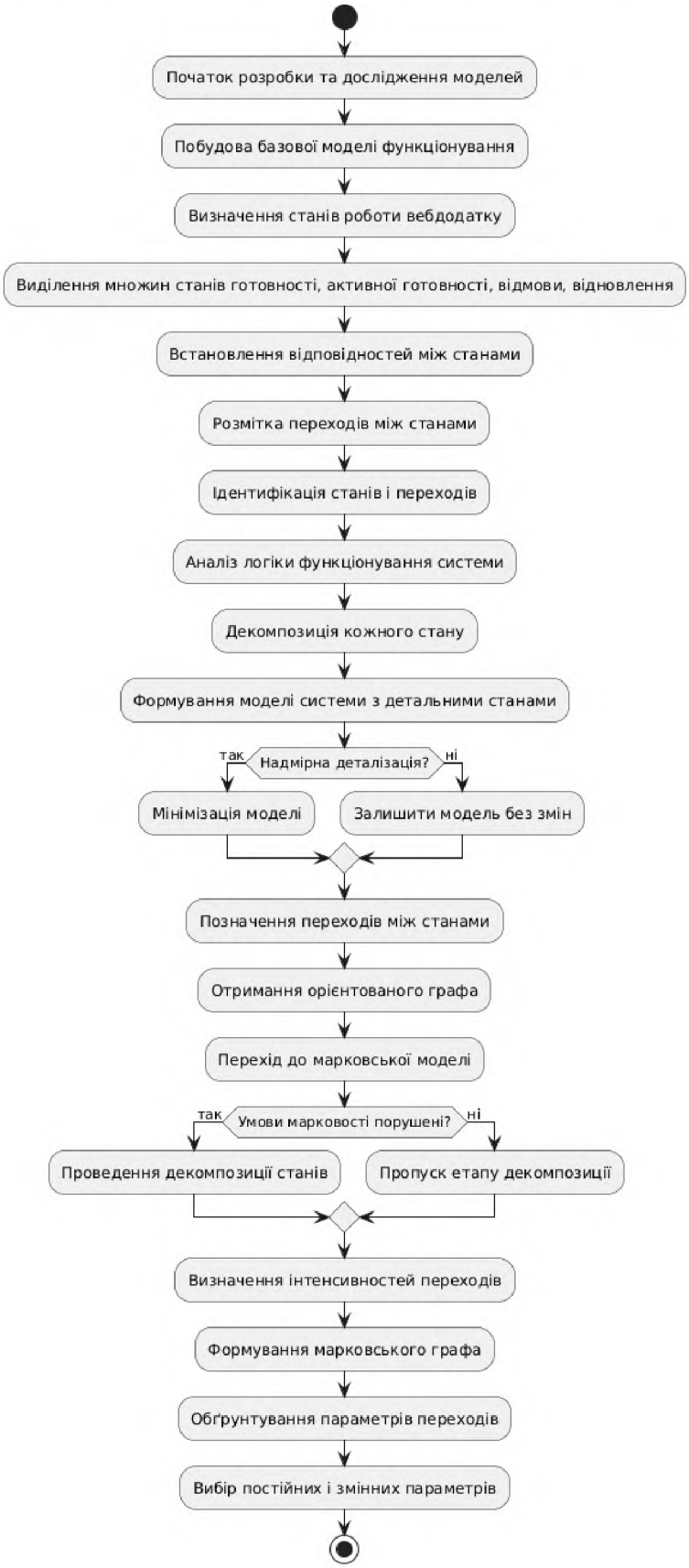


Рисунок 2.1 – Діаграма активностей при побудові марковських моделей

Останнім етапом є обґрунтування та вибір значень постійних і змінних параметрів інтенсивності переходів, використовуючи дані з вітчизняних та зарубіжних джерел, результати експериментів і закономірності, підтверджені в попередніх дослідженнях.

2.2 Модель функціональних (справних) станів інформаційної системи на прикладі вебдодатку для бронювання квитків

Першим кроком у процесі розробки (структурного синтезу) моделі є визначення станів системи, які входять до різних множин, та ідентифікація переходів між ними (відображень). Для цього важливо проаналізувати логіку функціонування системи, щоб зрозуміти, як вона поводитиметься на різних етапах. На верхньому рівні декомпозиції система має чотири основних стани, які визначають її реакцію на вхідний потік заявок. Розглянемо ці стани більш детально.

Стан 1: Ініціалізація системи. Цей стан є початковим, в який система переходить одразу після увімкнення. У цьому режимі виконується кілька важливих операцій:

а) запуск усіх модулів системи;

б) встановлення з'єднання з базою даних (БД);

в) формування початкового набору цільових компонентів вебдодатку;

г) налаштування системи під конкретний режим роботи;

д) ініціалізація черги запитів від клієнтів.

Ці дії гарантують, що система готова до прийому заявок і подальшого виконання своєї функції.

Стан 2: Отримання запиту з черги. Після ініціалізації система переходить до другого стану, де всі модулі вже запущені, а цільові компоненти вебдодатку зареєстровані в системі. У цей момент черга запитів ще порожня, і система чекає надходження перших клієнтських запитів. Як тільки з'являються перші заявки,

вони автоматично потрапляють у чергу, де система починає їх обробляти, розбиваючи на складові для подальшої декомпозиції.

Стан 3: Декомпозиція запитів. Запити, отримані з черги, надходять до блоку декомпозиції, який визначає структуру та категорію запиту. Наприклад, якщо йдеться про туристичне агентство, запит може містити декілька підзапитів, як-от бронювання авіаквитків, готелю чи оренда автомобіля. Відповідно, система аналізує наявні вебдодатки, зареєстровані в системі, і вибирає ті, які можуть виконати запитані дії. Цей процес є важливою частиною забезпечення коректної взаємодії між різними сервісами.

Стан 4: Обробка запитів. На цьому етапі система безпосередньо виконує обробку запитів. Згідно з раніше розглянутою структурою композитного сервісу, запити декомпозуються на заявки для різних сервісів на кожному рівні «вкладеності» компонентних серверів. Ці WSCA-заявки передаються на обробку до відповідних цільових сервісів. Далі система визначає, чи задоволений користувач отриманими результатами. Якщо відповідь позитивна, система формує остаточну відповідь і надсилає її клієнту. Якщо ж результат не задовольняє користувача, система відбраковує його і повідомляє про неможливість виконати запит.

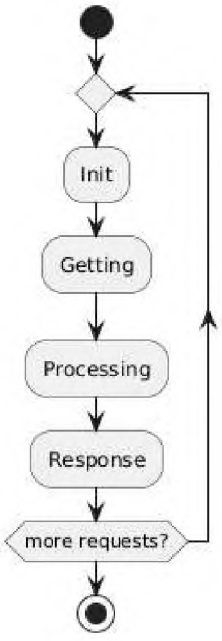


Рисунок 2.2 – Послідовність виконання операцій на верхньому рівні декомпозиції

На рисунку 2.2 показана схема послідовності виконання операцій на верхньому рівні декомпозиції. Вона наочно ілюструє процес від прийому заявки до її остаточної обробки. Щоб краще зрозуміти поведінку системи, виконаємо вертикальну декомпозицію кожного з описаних станів.

Процес ініціалізації (Init)складається з таких етапів:

– Start – запуск модулів системи;

– ConnectDB – підключення до системи керування базами даних (СУБД);

– ServList – формування початкового набору цільових сервісів;

– SysConf – конфігурація системи з урахуванням необхідного режиму роботи;

– Qinit – ініціалізація черги запитів.

Кожен з етапів виконується послідовно, і система переходить до наступного стану лише після успішного виконання попереднього. У випадку виникнення помилки на будь-якому етапі, система намагається повторити його виконання.

Декомпозиція стану отримання клієнтського запиту з черги містить такі етапи:

– Wait – очікування надходження заявки (періодичний огляд черги);

– Get – отримання клієнтського запиту;

– Decompose – декомпозиція запиту на компоненти.

Блок обробки клієнтських запитів є одним з основних функціональних елементів системи. Хоча безпосередня обробка запитів не моделюється, описано три ключових стани цього блоку:

– SendR – відправлення запитів на обробку цільовим сервісам;

– CollectR – збір результатів обробки;

– CheckR – перевірка результатів на валідність.

У цьому блоці можливі три типи переходів:

– послідовний перехід в наступний стан після успішного виконання операції;

– циклічний перехід назад до збору результатів, якщо вони ще не всі отримані;

– повторне відправлення запитів на обробку у разі, якщо результат не пройшов перевірку на валідність.

Завершальний модуль моделі сервісу – це відправка валідного результату клієнту. У цьому блоці є два стани:

– IsOk – перевірка прийнятності результату клієнтом;

– SendOk – відправка результату клієнтові у разі його схвалення.

Тут можливі три переходи:

– послідовний перехід зі стану перевірки в стан відправки;

– повернення до стану повторної обробки, якщо клієнт не задоволений результатом;

– очікування нового запиту, якщо результат було прийнято.

2.3 Проєктування моделі у вигляді розміченого графа

Систему можна представити як марковський процес із дискретними станами та неперервним часом. На рисунку 2.3 зображено модель у вигляді графа станів, де вершини представляють стани системи, а орієнтовані дуги – переходи між ними.

Модель має 24 стани, зокрема:

S0 – ініціалізація черги запитів;

S1 – очікування нової заявки;

S2 – отримання і декомпозиція запиту;

S3 – відправка запитів на обробку;

S4 – збір і перевірка результатів;

S5 – відправка результату клієнту;

F1-F8 – стани виявлених відмов системи;

F* – стан прихованої відмови;

P – профілактика системи.

Для розмітки марковського графа використовуються інтенсивності переходів між станами, що відповідають логіці функціонування моделі.

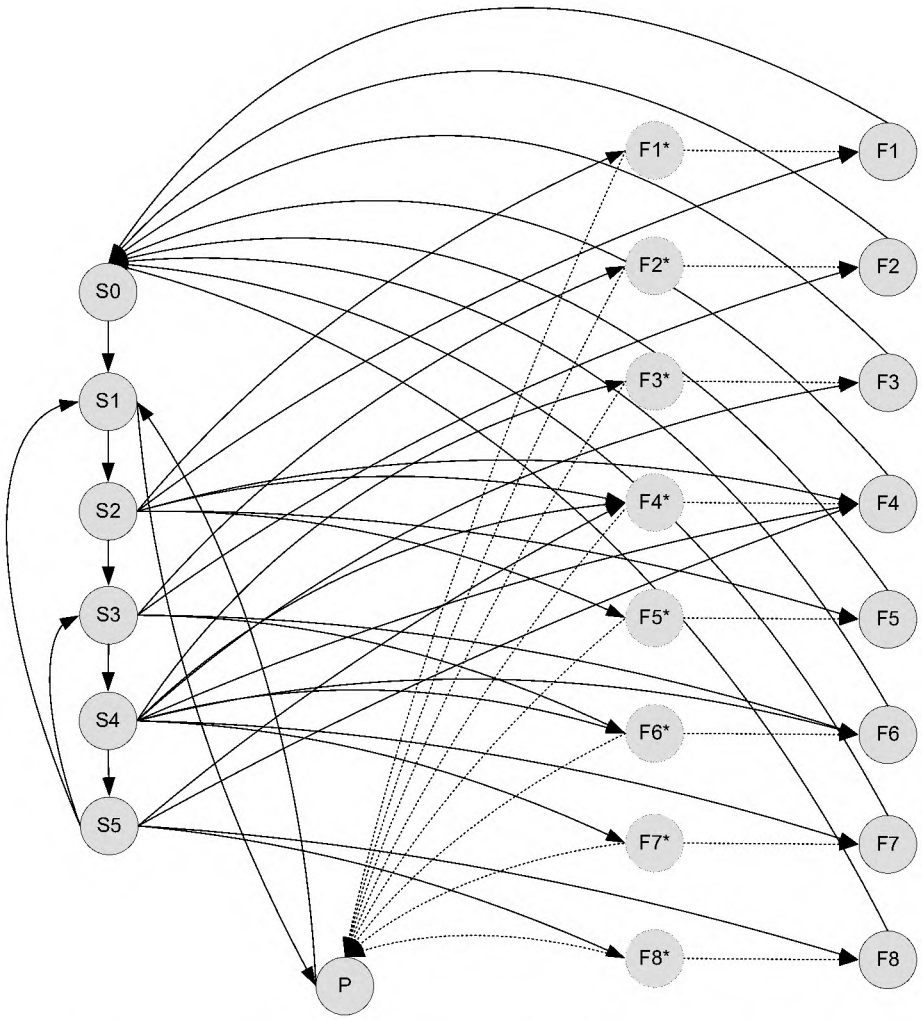
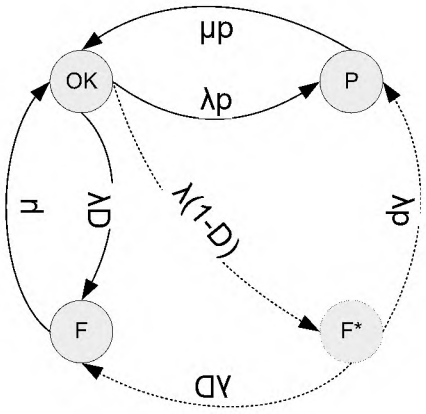


Рисунок 2.3 – Модель вебдодатку як нерозмічений граф станів системи

На рисунку 2.4 показано фрагмент розміченого графа з основними станами: працездатність (OK), профілактика (P) і відмова (F), де λ представляє інтенсивність переходу між станами.



OK - працездатний стан;

P - стан профілактики;

F - стан відмови;

λ - інтенсивність переходу.

Рисунок 2.4 – Розмічений фрагмент марковского графа

Модель враховує припущення щодо незалежності заявок, імовірностей відмов, а також інтенсивностей переходів. Як відомо, найпростіший потік характеризується такими властивостями:

– стаціонарність;

– ординарність;

– відсутність післядії.

Додамо два припущення, що описують цільову модель:

1. Заявки, отримані від різних джерел, є незалежними одна від одної.

2. Заявки, отримані від одного джерела, можуть бути залежними лише в тому випадку, якщо цільовий сервіс не надав послугу. Проте ймовірність такої події є надзвичайно малою.

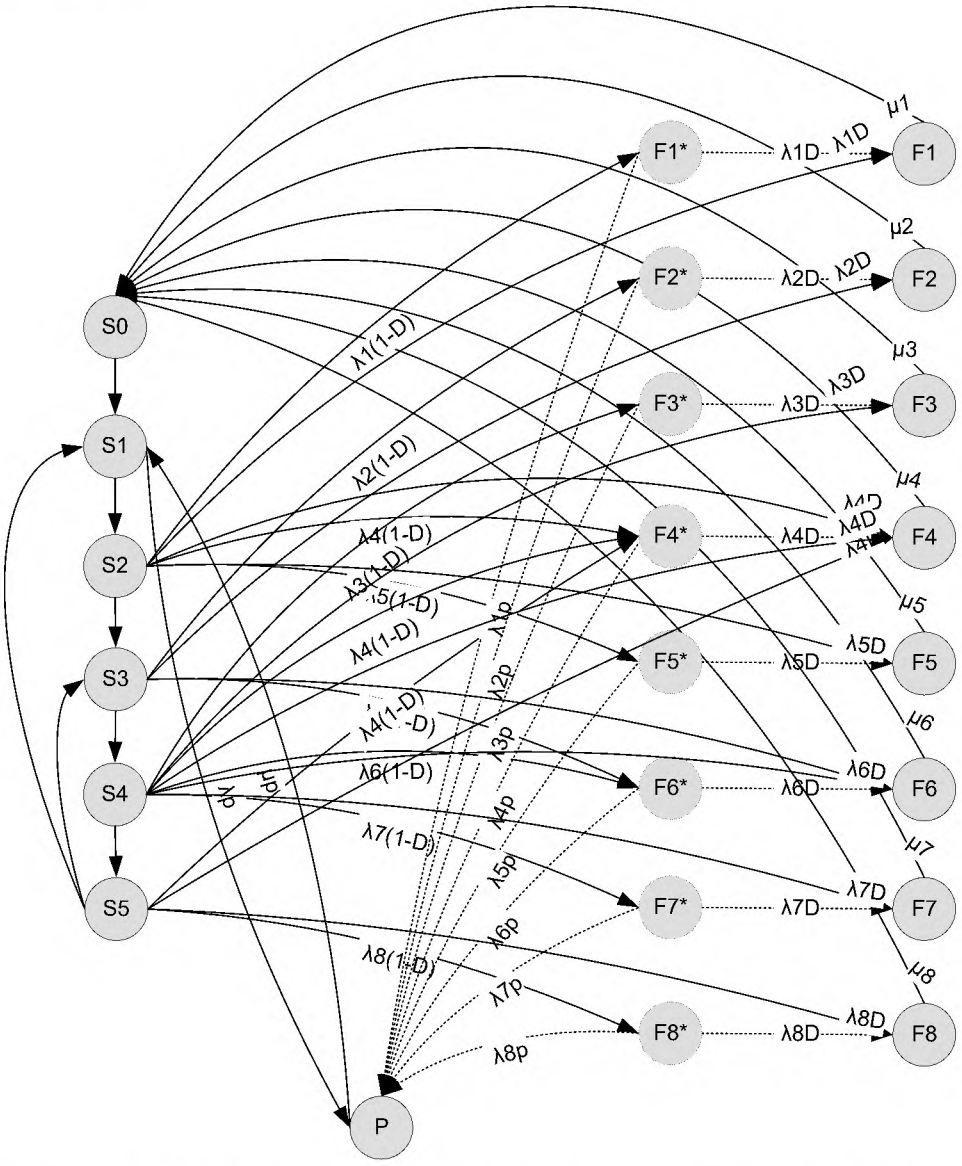


Рисунок 2.5 – Розмічений марковский граф моделі вебдодатку

Позначимо параметри інтенсивностей переходів між станами. Наприклад:

– перехід із працездатного стану ОК у стан профілактики P та назад позначимо як λp і μp відповідно;

– перехід із ОК у стан прихованої відмови F∗ позначимо як λ·(1−D) ;

– перехід із ОК у стан діагностованої відмови F позначимо як λ·D.

Нарешті, переходи між станами ОК і F будуть позначені як λ·D (у напрямку до відмови) та μ (у зворотному напрямку). Застосовуючи цей підхід, отримуємо розмічений граф (рис. 2.5). Використання інтенсивностей переходів між станами забезпечує можливість моделювання основних процесів, включаючи працездатність, профілактику та відмови. Врахування властивостей найпростішого потоку та припущень щодо незалежності заявок дозволяє створити математично коректну модель для аналізу та оптимізації роботи вебдодатку.

2.4 Визначення вхідних параметрів марковської моделі

Для визначення показників завантаження інформаційних систем [25] використовується інтенсивність запитів у межах від 1 до 100 запитів за секунду. Такі дані базуються на результатах досліджень лабораторії Hewlett Packard Labs, які були отримані при аналізі файлів журналу доступу до вебсайтів [26]. Середнє значення інтенсивності запитів можна оцінити як 10 запитів за секунду або 36 000 запитів на годину.

Додатковий аналіз статистичних даних ресурсу BASIS WS, метою якого було вивчення надійності сервіс-орієнтованих систем біомедичного спрямування, показав, що максимальна кількість запитів становила 15,74 запити за секунду (56 664 запити на годину), а мінімальна – 2,16 запити за секунду (7 776 запитів на годину). Таким чином, середнє значення інтенсивності запитів у реальних SOA-системах оцінюється як 8,95 запитів за секунду (32 220 запитів на годину).

Варто зазначити, що інтенсивність запитів може коливатися залежно від часу доби, сезонів або специфічних подій. Наприклад, для туристичних агентств

найбільше навантаження спостерігається у серпні-вересні та січні, а також під час рекламних кампаній («гарячі путівки» або промо-тури). У часовому розрізі найвища активність користувачів припадає на 15–16 годину дня, тоді як найменша на 4–5 годину ранку. Загальний діапазон інтенсивності запитів для таких систем може варіюватися від 100 до 100 000 запитів на годину.

Щодо інтенсивностей переходів системи у стани збоїв та атак, можна виділити наступне. Для кожного сервера в кластерній системі середню інтенсивність збоїв приймають як 0,001 (1/год), що еквівалентно 10^{-3} [24]. Інтенсивність відмов становить приблизно 10^{-5} (1/год). Програмні збої трапляються частіше, ніж апаратні: їх інтенсивність можна оцінити на рівні 10^{-2} (1/год), тоді як інтенсивність відмов через програмне забезпечення дорівнює 10^{-4} (1/год). Інтенсивність збоїв операційної системи зазвичай удвічі перевищує інтенсивність збоїв системного ПЗ (наприклад, СУБД або вебсерверів), а збої прикладного ПЗ можуть бути втричі частішими за збої системного ПЗ [22].

Для оцінки інтенсивностей атак можна скористатися такими припущеннями [25]: на початкових етапах інтенсивність атак становить 0,01–0,1% від загального потоку запитів. Якщо зловмисник виявляє вразливість, потік атак може збільшитися до 1% від потоку запитів.

Таблиця 2.1 – Постійні значення параметрів марковської моделі вебдодатку

λ	Значення (1/годин)	μ	Значення (1/годин)
λ_1	10^{-5}	μ_1	0,5
λ_2	10^{-4}	μ_2	1,429
λ_3	$5 \cdot 10^{-3}$	μ_3	8,571
λ_4	$5 \cdot 10^{-4}$	μ_4	2,4
λ_5	$9 \cdot 10^{-4}$	μ_5	15
λ_6	$3 \cdot 10^{-3}$	μ_6	6
λ_7	$2 \cdot 10^{-3}$	μ_7	20
λ_8	$9 \cdot 10^{-4}$	μ_8	0,6

Для дослідження поведінки системи при різних значеннях вхідних параметрів, в моделі були прийняті наступні постійні вихідні дані, представлені в таблиці 2.1. Далі, для ряду параметрів були прийняті інтервальні оцінки, що дозволило дослідити динаміку поведінки системи. Межі зміни цих параметрів і їх значення за замовчуванням вказані в таблиці 2.2.

Таблиця 2.2 – Змінні значення параметрів марковської моделі вебдодатку

Параметр	Інтервал зміни	Значення за замовчуванням
$\rho_1 - \rho_7$	0,1…100000	32000 (1/годин)
D	0…1	0,8
μ_p	0,3…0,6	0,5 (1/годин)
λ_p	0,00595…1	0,00595 (1/годин)

Постійні значення інтенсивностей переходів для марковської моделі вебдодатку (таблиця 2.1) охоплюють інтервали від 10^{-5} до $3 \cdot 10^{-3}$ (1/год), а коефіцієнти виходу зі станів (μ) варіюються від 0,5 до 20 (1/год).

Для параметрів із змінними значеннями (таблиця 2.2) було визначено діапазони:

– Інтенсивності обробки запитів (ρ1–ρ7) коливаються в межах від 0,1 до 100 000 (1/год), із значенням за замовчуванням 32 000 (1/год).

– Параметр D, який визначає ймовірність успішної обробки запиту, змінюється від 0 до 1, із типовим значенням 0,8.

– Інтенсивності профілактики (λp і μp) мають свої діапазони, залежно від частоти профілактичних заходів: наприклад, профілактика раз на тиждень визначає λp=0,00595 (1/год), а при щоденній профілактиці – λp=0,0417 (1/год).

Інтенсивності обробки запитів (ρ1–ρ7) у моделі вважаються рівними при відсутності затримок. Для профілактики значення μp обирається як найменше серед параметрів виходу зі станів (μ1–μ8), що дорівнює 0,5 (1/год).

Дані параметри дозволяють моделювати поведінку системи за різних умов і робити прогнози щодо її продуктивності, надійності та захищеності.

2.5 Дослідження функціонування інформаційної системи за допомогою марковської моделі

Модифікований граф, наведений на рисунку 2.6, створений шляхом внесення змін до вихідного графа, зображеного на рисунку 2.5. Основні зміни включають:

– впровадження наскрізної нумерації станів, де несправні стани поділяються на приховані відмови (S14–S21), явні відмови (S6–S13), а також стан профілактики (S22);

– додавання позначень переходів між справними станами як ρ1–ρ7;

– прямі переходи зі справних станів до станів явних відмов;

– позначення інтенсивностей переходів із станів прихованих відмов до стану профілактики як λp.

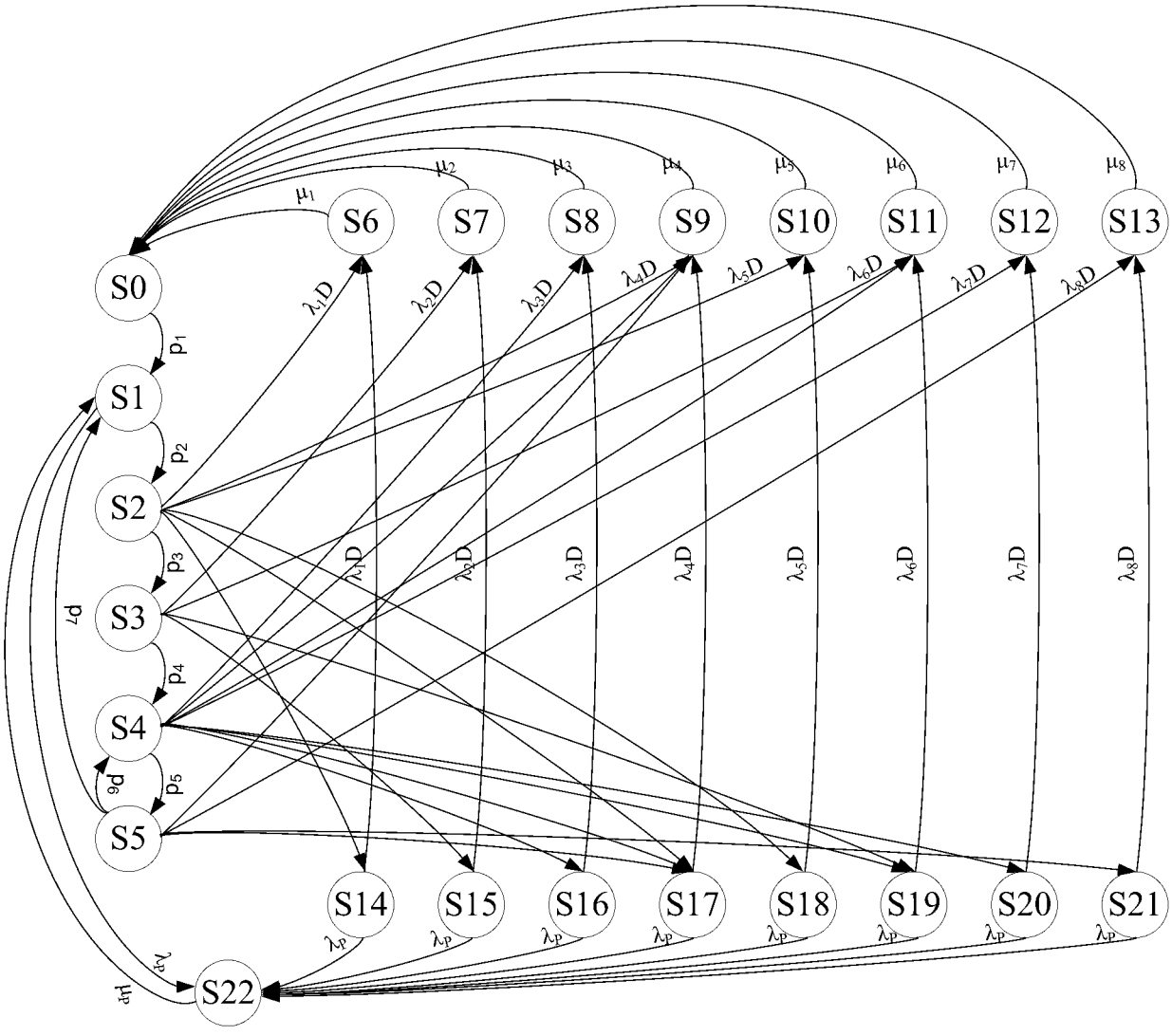


Рисунок 2.6 – Розмічений марковський граф моделі вебдодатку

Розв’язавши систему лінійних диференціальних рівнянь Колмогорова, побудовану на основі графа з рисунка 2.6, для параметрів, наведених у таблиці 2.1, було отримано результати, представлені на рисунку 2.7.

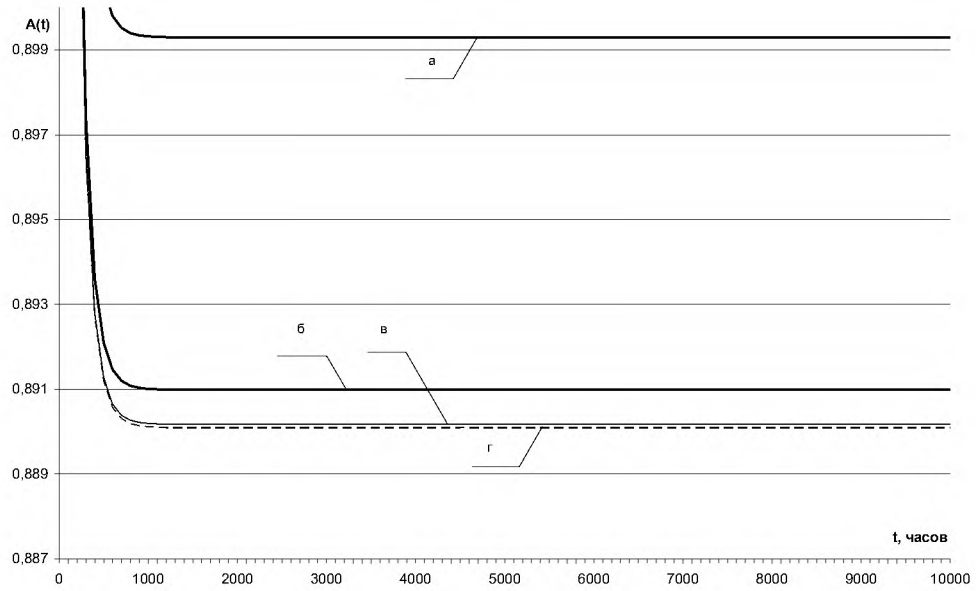


Рисунок 2.7 – Залежність коефіцієнта готовності системи А для різних значень параметрів ρ: а) ρ=0,1; б) ρ=1; в) ρ=5; г) ρ=8

Для побудови матриці системи рівнянь Колмогорова-Чепмена використовувалася функція matrixA [27]. Система рівнянь розв'язувалася в середовищі MATLAB за допомогою методу ode15s для часових інтервалів до 10 000 годин. Результати розв'язання представлені графічно на рисунку 2.7.

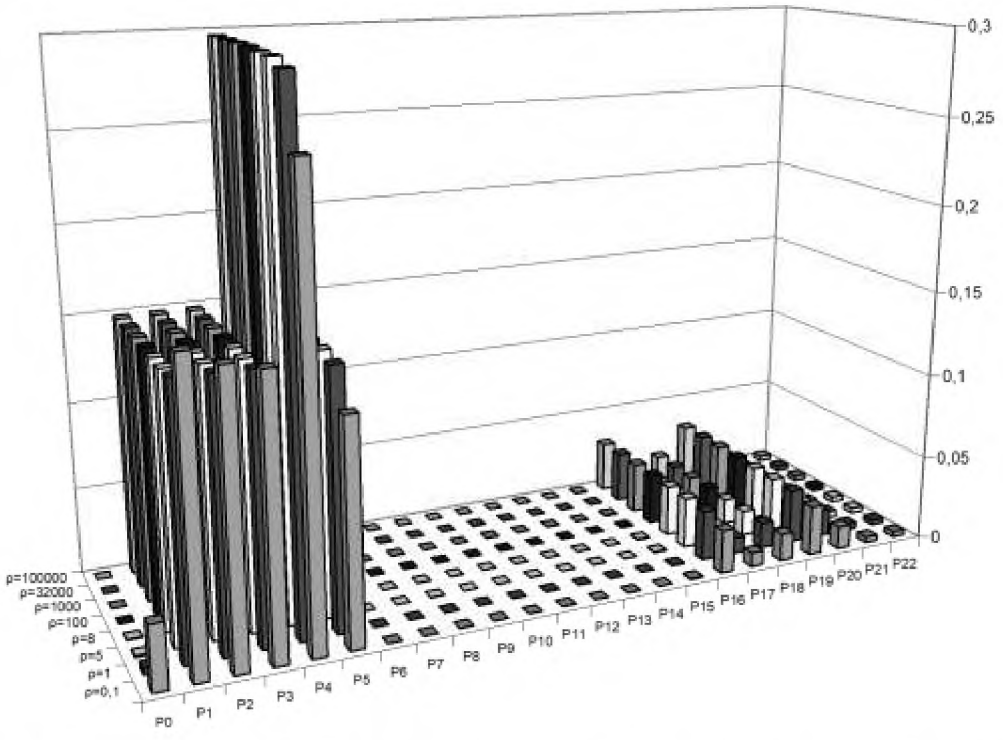


Рисунок 2.8 – Діаграма розподілу ймовірностей станів системи моделі ІС при різних значеннях параметрів ρ1–ρ7

Графіки показують, що для малих значень параметра ρ динаміка зміни коефіцієнта готовності залишається стабільною. Система досягає стаціонарних значень через 900 годин із похибкою 10^{-5} або через 1200 годин із похибкою 10^{-6}. Це дозволяє припустити, що для великих значень ρ характер динаміки залишиться подібним. У стаціонарному режимі значення коефіцієнта готовності можна розраховувати через систему лінійних рівнянь, що спрощує дослідження залежностей від інших параметрів.

Аналіз даних рисунків 2.7 і 2.8 вказує на те, що основними чинниками, які знижують готовність ІС, є приховані відмови програмного забезпечення (ймовірності P16–P21) та часте проведення профілактики прихованих відмов (ймовірність P22).

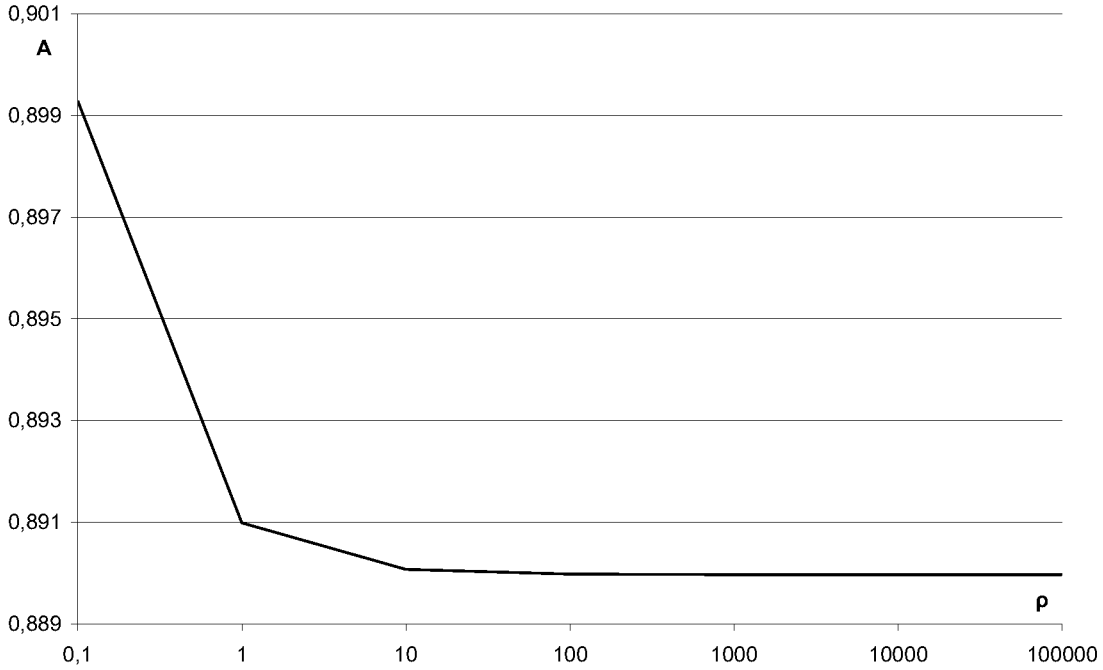


Рисунок 2.9 – Залежність коефіцієнта готовності системи A від значень параметрів ρ1–ρ7

Збільшення інтенсивності запитів (ρ1–ρ7) навіть на кілька порядків майже не впливає на готовність системи. Це викликано рівномірним перерозподілом ймовірностей початкових станів між P1, P2, P3 і P5. Ймовірність перебування системи в початковому стані (P0) при збільшенні ρ1–ρ7 прямує до нуля.

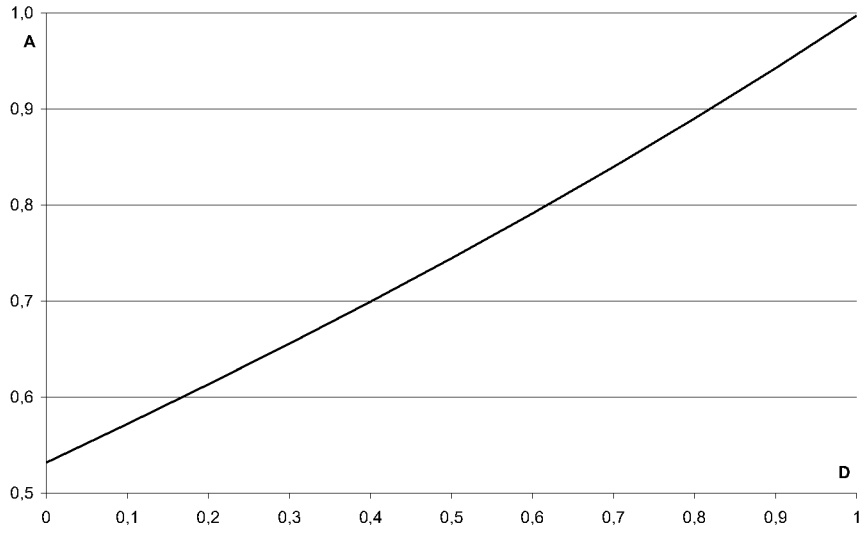


Рисунок 2.10 – Залежність коефіцієнта готовності системи A від значень параметра D

На рисунку 2.10 демонструється, що підвищення повноти контролю (D) зменшує ймовірність прихованих відмов, завдяки чому зростає загальна готовність системи. Залежність між повнотою контролю і коефіцієнтом готовності є майже лінійною.

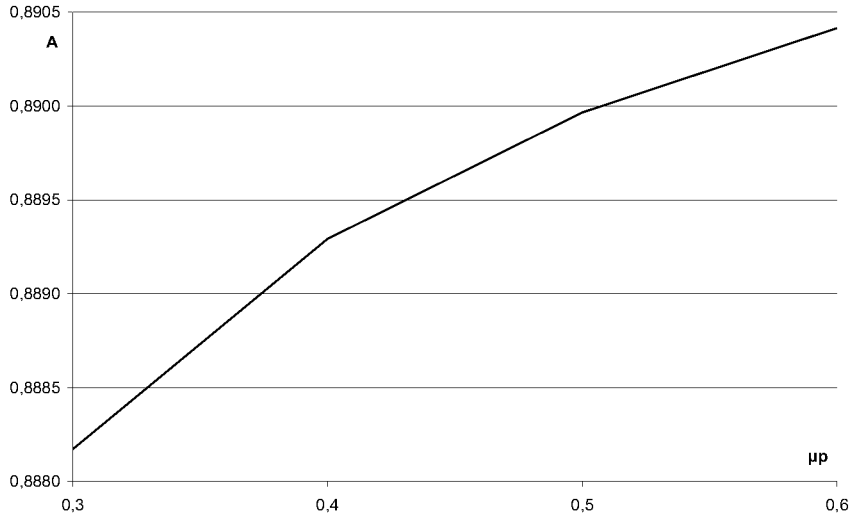


Рисунок 2.11 – Залежність коефіцієнта готовності системи A від значень параметра μp

Аналіз графіка на рисунку 2.11 свідчить, що залежність готовності від середньої тривалості профілактики (μp) має нелінійний характер. Зменшення

тривалості профілактики на дві години дозволяє підвищити готовність системи лише на 0,0025 у межах значення 0,888. Це вказує на обмежену ефективність скорочення часу профілактики за заданих параметрів.

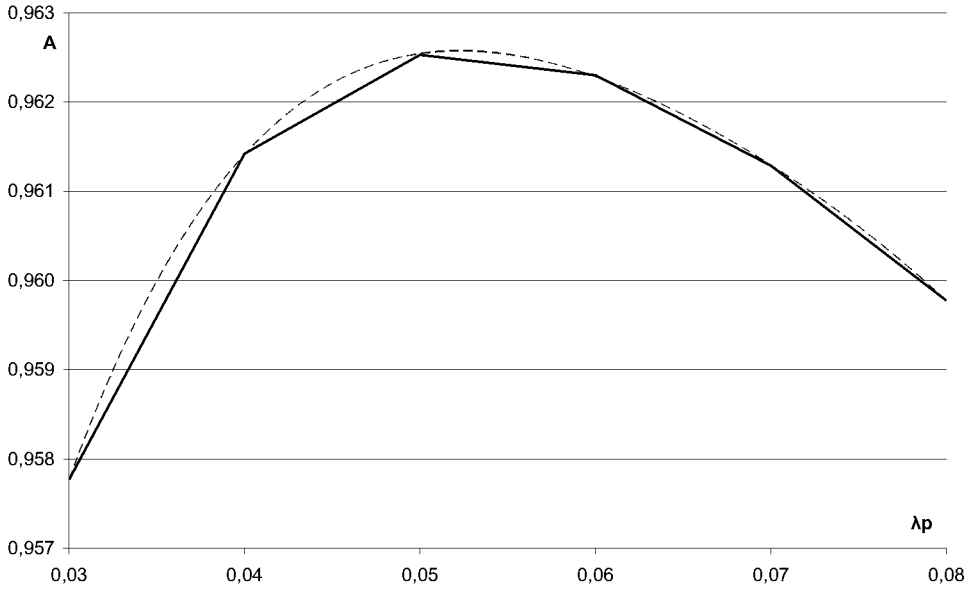


Рисунок 2.12 – Залежність коефіцієнта готовності системи A від значень параметра λp

Графік на рисунку 2.12 демонструє, що збільшення інтенсивності профілактики (λp) дозволяє ефективно усувати приховані відмови до певного рівня, після якого система частіше перебуває у стані профілактики (S22), що знижує її готовність. Оптимальне значення λp, при якому готовність є максимальною, дорівнює приблизно 0,05 (період профілактики Tp=20 годин). Подальше вдосконалення моделі може включати апроксимацію функції A(λp) для точнішого визначення оптимального значення.

Проведене дослідження показало, що на готовність системи найбільше впливають приховані відмови програмного забезпечення та частота профілактики. Оптимізація параметрів профілактики, зокрема λp, дозволяє підвищити надійність системи. Залежності готовності від інших параметрів вказують на можливість подальшої оптимізації системи через балансування параметрів профілактики та контролю.

Висновки до розділу 2

У другому розділі було розглянуто принципи та послідовність розробки марковських моделей для оцінки готовності інформаційних систем. Проаналізовано ключові аспекти створення моделей, включаючи визначення станів системи, розробку орієнтованого графа переходів та оцінку інтенсивностей переходів між станами. Визначено основні стани функціонування інформаційних систем, такі як готовність, відмова, профілактика та приховані відмови, а також їх вплив на надійність системи.

Виконано структурування процесів моделювання на основі декомпозиції системи, що включає вертикальний і функціональний аналіз. Створено розмічений граф марковської моделі, який дозволяє дослідити динаміку переходів між станами, включаючи випадкові відмови та відновлення системи. Проаналізовано вхідні параметри марковської моделі, серед яких інтенсивність запитів, ймовірність успішної обробки та параметри профілактичних заходів. Встановлено, що діапазон інтенсивності запитів може суттєво коливатися залежно від часу доби, сезону та специфічних подій, що створює додаткові виклики для моделювання.

Досліджено функціонування системи за допомогою графів станів і системи рівнянь Колмогорова, результати яких дозволили побудувати залежності коефіцієнта готовності від різних параметрів моделі. Визначено, що готовність системи значною мірою залежить від прихованих відмов програмного забезпечення та частоти профілактичних заходів.

Розглянуто вплив окремих параметрів, таких як інтенсивність обробки запитів ($\rho 1-\rho 7$), повнота контролю (D), тривалість профілактики (μp) та її інтенсивність (λp), на готовність системи. Зокрема, виявлено, що збільшення інтенсивності профілактики до певного оптимального рівня підвищує готовність, але її надмірне збільшення може мати протилежний ефект через перевантаження системи профілактичними заходами.

РОЗДІЛ 3
РОЗРОБКА І ДОСЛІДЖЕННЯ ІМІТАЦІЙНОЇ МОДЕЛІ ФУНКЦІОНУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ В УМОВАХ АТАК НА ЇЇ ГОТОВНІСТЬ

3.1 Обмеження алгоритму імітаційного моделювання

Необхідність імітаційного моделювання для дослідження інформаційних систем як відновлюваних систем, що функціонують в умовах прояву дефектів апаратного та програмного забезпечення, а також здійснення оновлень ПЗ, зумовлена такими факторами [22]:

– потреба перевірки достовірності результатів, отриманих за допомогою аналітичного моделювання [23];

– розробка підходу до побудови імітаційних моделей функціонування розглянутих систем для усунення обмежень, які накладаються при аналітичному моделюванні.

Метою імітаційного моделювання є визначення динаміки змін коефіцієнта готовності ІС в процесі його експлуатації з урахуванням оновлень програмного забезпечення [26].

Аналіз завдання виявив, що досліджуваний процес можна формалізувати у межах математичних схем систем масового обслуговування (Q-схем). Однак застосування принципів формалізації Q-схем для створення алгоритму та його реалізації із використанням спеціалізованих мов імітаційного моделювання (GPSS, SLAM, GASP, SIMSCRIPT) або модулів імітаційного моделювання математичних пакетів (наприклад, SIMULINK у складі MATLAB) є трудомістким через специфіку зміни інтенсивності прояву дефектів і потребу у специфічному поданні результатів.

З огляду на це, доцільно використовувати метод безпосереднього моделювання, реалізований мовами програмування високого рівня [25]. Це

дозволяє поєднувати значення коефіцієнта готовності для різних часових моментів з метою отримання функціональної залежності коефіцієнта готовності .

Параметри імітаційної моделі

У процесі статистичних випробувань до аналітичних параметрів (λ_{HW}, λ_{SW}, μ_{HW}, μ_{SW}, λ_{UP}, μ_{UP}) додаються:

– α – довірча ймовірність оцінки коефіцієнта готовності;

– ε – похибка визначення коефіцієнта готовності.

Коефіцієнт готовності є випадковою величиною , яка може набувати одного з можливих значень. Статистичною оцінкою є вибіркове середнє, яке визначається з точністю і довірчою ймовірністю . Для обчислення потрібної кількості реалізацій процесу використовується послідовний метод, що складається з наступних етапів:

1. Генерація першої реалізації випадкової величини.

2. Генерація і-ї реалізації , що є незалежною від попередніх.

3. Оцінка дисперсії реалізацій за формулою:

$$S_n^2 = \frac{1}{n-1}\left[\sum_{i=1}^{n} A_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} A_i\right)^2\right].$$

4. Обчислення точності оцінки :

$$\delta = \frac{t_{\alpha/2,n-1} \cdot S_n}{\sqrt{n}},$$

де $t_{\alpha/2,n-1}$ – квантиль розподілу Стьюдента зі ступенями свободи .

5. Перевірка умови $\delta \leq \varepsilon$: якщо умова не виконується, процес повторюється зі збільшенням n+1. У разі виконання приймається значення $n_{тр}=n$.

Моделюючий алгоритм функціонування ІС базується на принципі особливих станів. Події, які викликають зміну стану системи (відмови, відновлення, оновлення ПЗ), обробляються послідовно, при цьому системний час змінюється відповідно до моментів подій. Випадкові часові інтервали моделюються за допомогою закону розподілу:

$$t = \frac{1}{\pi}\ln(Random)$$

де t – час безвідмовної роботи каналу;

Random – випадкова величина на інтервалі (0, 1), згенерована програмним датчиком ПСЧ;

π – параметр розподілу (λ або μ).

Блок-схема алгоритму моделювання представлена на рис. 3.1.

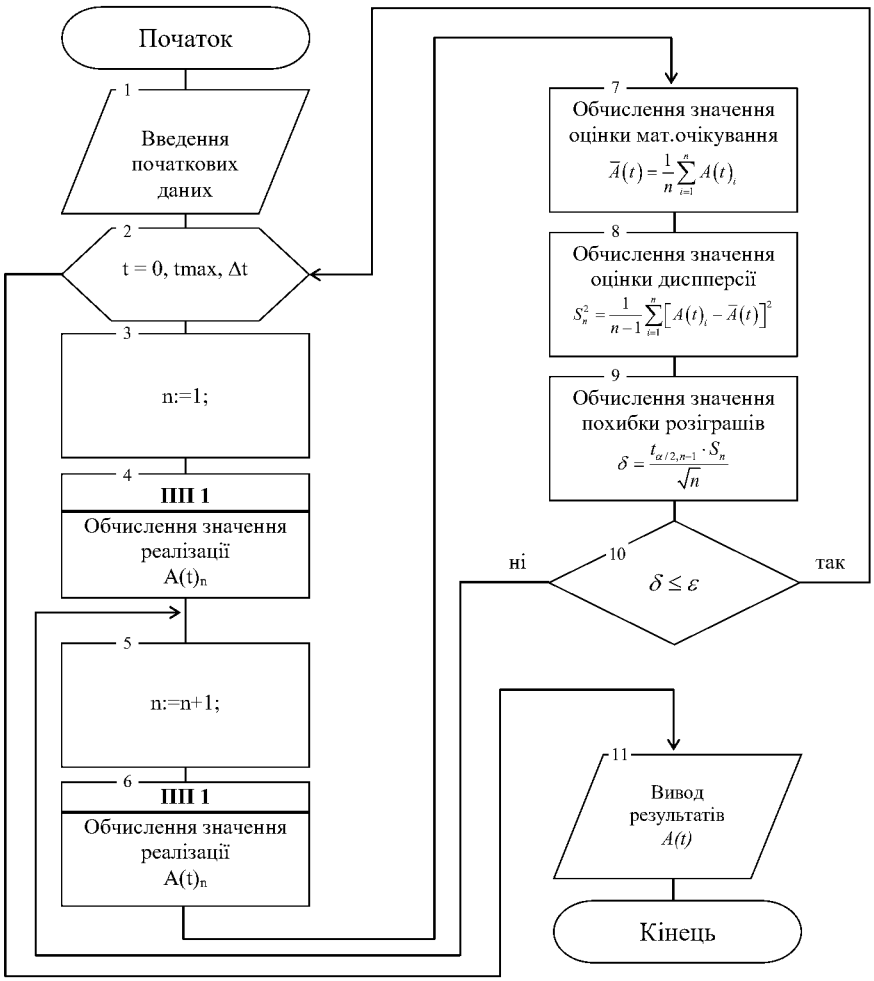


Рисунок 3.1 – Алгоритм імітаційного моделювання ІС за певною кількістю розіграшів $n_{тр}$

Розроблена модель усуває недоліки попередніх рішень [21, 22]:

1. Замість чотирьох динамічних масивів для кожного каналу використовується один масив, оскільки немає необхідності зберігати кожен часовий інтервал окремо.

2. Розмірність масиву значень квантиля розподілу Стьюдента збільшена з [3, 100] до [4, 500], що підвищує достовірність моделювання до 0,999.

3. Для оцінки справності системи застосовується метод множення логічних станів апаратних каналів із подальшим додаванням до логічного стану програмного каналу.

4. Кількість початкових реалізацій збільшена з 10 до 50, що знижує похибку розрахунку вибіркового середнього значення.

Розроблене імітаційне моделювання дозволяє аналізувати функціонування інформаційних систем як відновлюваних систем із високою точністю. Удосконалена модель спрощує обробку даних і підвищує ефективність розрахунків, забезпечуючи більш надійні результати. Використання методу послідовного моделювання знижує витрати ресурсів на початкових етапах дослідження. Це дає змогу адаптувати алгоритм для широкого спектра практичних завдань, що стосуються підтримки надійності вебсервісів.

3.2 Розробка імітаційної моделі функціонування веб-системи в умовах атак на її компоненти без визначення кількості розіграшів

Кількість запропонованих аналітичних моделей на даний момент є обмеженою (у порівнянні з більш розвиненими моделями [28, 29]), що дозволяє уникнути необхідності створення складних імітаційних моделей для веб-серверів. У рамках цієї роботи імітаційне моделювання має на меті визначити зміну показників функції готовності веб-сервера під час експлуатації системи, зокрема за умов атак на службу DNS [29]. Достовірність аналітичних моделей перевіряється шляхом статистичних випробувань.

Аналіз завдання імітаційного моделювання вказав, що описуваний процес належить до класу математичних схем систем масового обслуговування (так званих Q-схем) [22]. Для досягнення цієї мети було розроблено і реалізовано алгоритм моделювання, який базується на принципах формалізації Q-схем. Реалізацію здійснено за допомогою модуля імітаційного моделювання математичного пакета MATLAB. Оскільки розв'язувані завдання мають обмежений масштаб, візуально-

орієнтований інструмент SIMULINK не використовувався; замість нього застосовувалися функції командного вікна MATLAB.

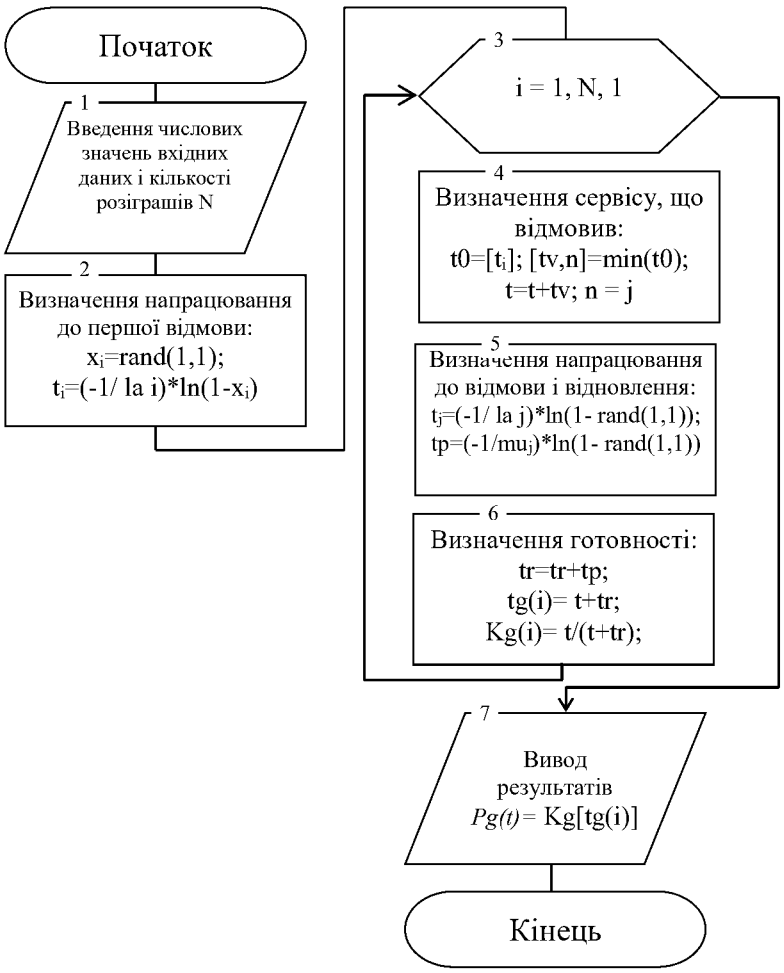


Рисунок 3.2 – Алгоритм імітаційного моделювання вебсистем

Детальний опис команд пакета MATLAB, які використовуються для імітаційного моделювання, наведено в [22]. На основі алгоритму імітаційного моделювання (рис. 3.2) було розроблено такі статистичні моделі ІС:

– модель із врахуванням відмов і відновлень трьох служб;

– модель, яка враховує атаки на службу DNS із подальшим її перезапуском;

– модель для атак на три служби із їх подальшим перезапуском;

– модель, що враховує атаки на службу DNS із подальшим усуненням несправностей конфігурації.

Програмні реалізації зазначених імітаційних моделей, розроблені в MATLAB, представлені в додатку А.

Імітаційне моделювання підтверджує ефективність аналітичних підходів до оцінки готовності веб-систем за умов атак. Розроблені моделі демонструють широкий спектр сценаріїв і забезпечують гнучкість аналізу систем. Використання MATLAB дозволяє швидко адаптувати методи моделювання до конкретних завдань.

3.3 Порівняльний аналіз результатів аналітичного і імітаційного моделювання

Результати порівняння імітаційних та аналітичних моделей представлені на рис. 3.3 і рис. 3.4. Щоб забезпечити належне відображення поведінки моделей, часовий інтервал для аналітичних розрахунків був штучно збільшений на порядок.

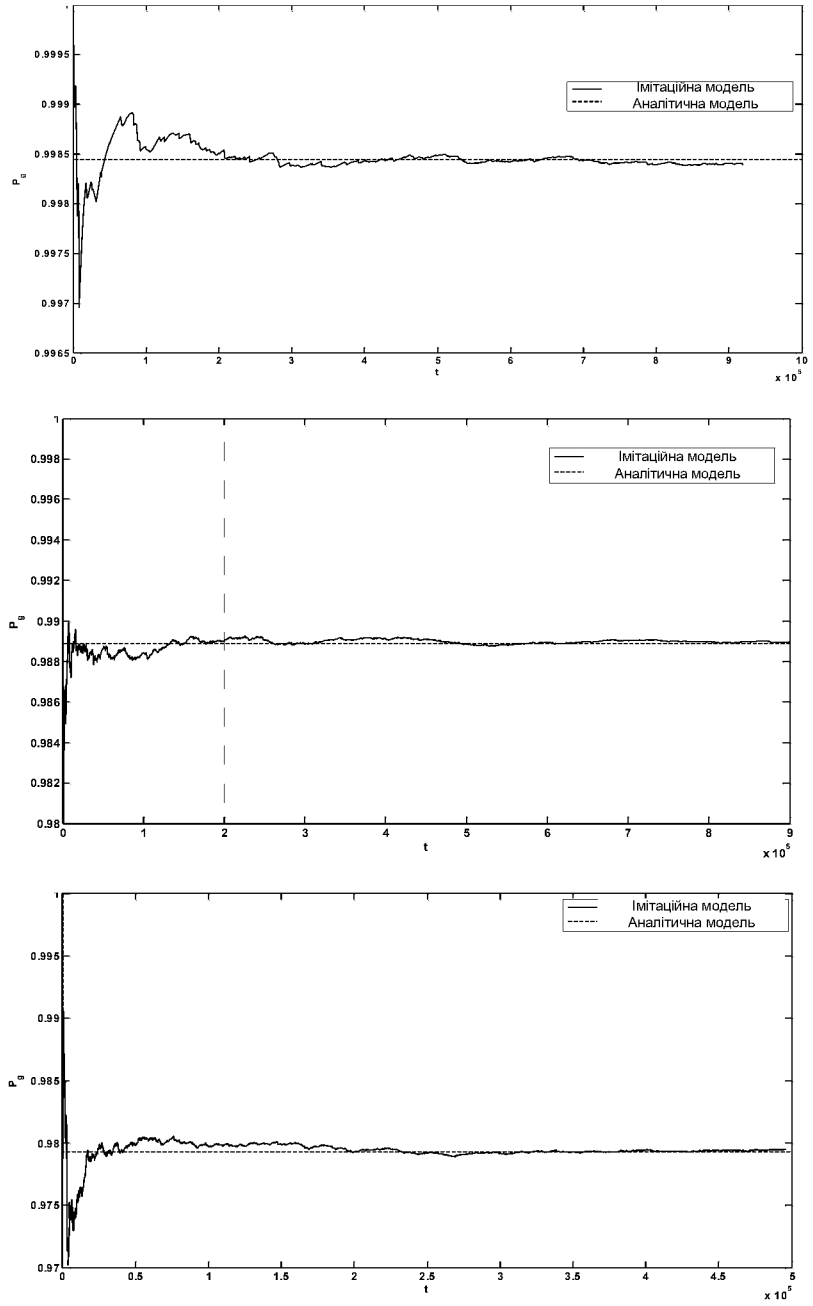
На графіках функцій готовності чітко простежуються дві характерні фази: перехідна, яка проявляється на початкових етапах роботи системи, і стаціонарна, що відображає усталений стан. Для імітаційних моделей, як видно на рис. 3.3, спостерігається специфічна поведінка функції готовності під час перехідної фази. Зокрема:

– на графіку (а) лінія готовності кілька разів перетинає горизонтальну асимптоту усталеного значення;

– на графіках (б) і (в) крива готовності підходить до асимптоти знизу.

Перехідний період у імітаційних моделях триває приблизно $3 \cdot 10^5$ годин, що значно більше (на чотири порядки), ніж у відповідних аналітичних моделей. Однак у стаціонарному стані значення коефіцієнта готовності в обох типах моделей практично збігаються, що підтверджує адекватність аналітичних підходів.

Порівняння моделей, що враховують усунення вразливостей конфігурації служби DNS, продемонструвало високу схожість коефіцієнтів готовності в усталеному режимі. Зокрема, на рис. 3.4 (б) імітаційна модель точно повторює динаміку аналітичної: після 5000 годин експлуатації вихід кривої готовності з точки мінімуму співпадає для обох моделей.



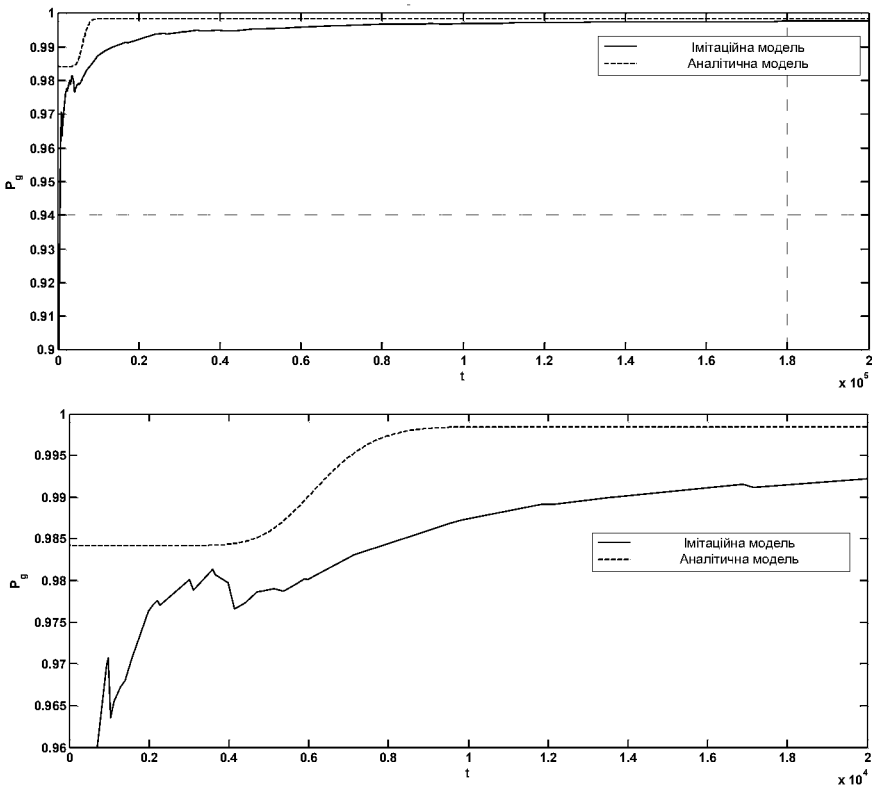
а) модель, що враховує відмови та відновлення трьох служб

б) модель із урахуванням атак на службу DNS та її подальшим перезапуском

в) модель із урахуванням атак на три служби з подальшим їх перезапуском

Рисунок 3.3 – Порівняння результатів імітаційного моделювання інформаційних систем і аналітичних моделей

Однак видно, що імітаційна модель досягає усталеного значення із затримкою порівняно з аналітичною. Це може бути пов’язано з навмисним підвищенням кількості вразливостей у вихідних даних до 30 (nv=30n_v = 30nv=30) для ілюстрації цього ефекту.



а) модель із часовим інтервалом [0...200000] годин

б) модель із часовим інтервалом [0...20000] годин

Рисунок 3.4 – Порівняння результатів імітаційного моделювання інформаційних систем і аналітичної моделі з урахуванням атак на службу DNS з подальшим усуненням несправностей конфігурації

Проведений аналіз демонструє високу відповідність між аналітичними та імітаційними моделями, особливо в усталеному стані. Імітаційні моделі дозволяють детальніше вивчати поведінку системи під час перехідного періоду, тоді як аналітичні підходи забезпечують швидший розрахунок сталих характеристик. Це підтверджує доцільність комбінованого використання обох підходів для оцінки надійності інформаційних систем.

3.4 Імітаційне моделювання функціонування інформаційної системи в умовах прояву дефектів та атак на її вразливості

Процес моделювання системи проводиться поетапно і включає кілька ключових кроків. На першому етапі після запуску скрипта здійснюється введення початкових значень вхідних параметрів, які визначають інтенсивності переходів і

розмірність моделі. Зокрема, задаються параметри, такі як Taim_interval – часовий інтервал дослідження поведінки системи, та Nr – кількість повних розіграшів імітаційної моделі. Важливою частиною є уточнення параметра Taim_interval, який визначається як подвоєне значення часу переходу до сталого стану функції готовності, отримане за допомогою аналітичної моделі ІС.

Під одним розіграшем розуміється повна побудова функції готовності на інтервалі [0..Taim_interval]. У ході розіграшу фіксуються два важливі показники: мінімальне значення функції готовності Amin та момент часу tconst, коли система переходить у поглинаючий стан (за умови, що це відбувається).

Розіграш здійснюється, починаючи з початкового стану S0. Цей стан відповідає першому елементу масиву V: V(1,:) = [0 0 0 0]. Відповідно, допоміжна змінна n приймає значення 1. Зі стану S0 можливі переходи в інші стани, наприклад, S9 і S15. Ці переходи описуються масивом E: E(1,:) = [1 10 0.003] та E(19,:) = [1 16 0.0005]. Для формування вибірки таких елементів використовується вираз: Q = E(find(E(:,1)==n),:).

Далі формується вектор t1, який містить часові інтервали, що визначають перехід із стану S0. Інтервали генеруються за допомогою функції t1 = exprnd(1/mu), де mu – вага відповідної дуги графа з масиву E. Мінімальне значення цього вектора визначає перехід до наступного стану. Відповідно, допоміжна змінна k отримує значення, яке визначається як k = Q(q+find(t1==min(t1))).

Після кожного переходу виконується перерахунок напрацювання між відмовами Tbf, якщо перехід відбувався з працездатного стану (наприклад, S0). Якщо ж перехід здійснювався з непрацездатного стану (n>(Nd+1)*(Nv+1)), тоді здійснюється перерахунок напрацювання відновлень Tbr.

Розіграш триває до досягнення системою поглинаючого стану S9 (за умови Nd = 2, Nv = 2) або поки сума всіх часових інтервалів не перевищить Taim_interval. Якщо система переходить у поглинаючий стан, готовність приймається рівною 1 для всіх подальших часових відліків. Повний розіграш функції готовності складається з сукупності часткових розіграшів, що дозволяють побудувати функцію готовності на всьому інтервалі [0..Taim_interval].

Після виконання Nr розіграшів формуються множини значень tconst та Amin, які обробляються за допомогою статистичних функцій MATLAB. Однак з множиною реалізацій функції готовності ситуація складніша, оскільки кожна реалізація має унікальні часові відліки. На рис. 3.5 наведено результати 10 000 повних розіграшів функції готовності у порівнянні з аналітичною моделлю ІС. Як видно, порівняння результатів аналітичного та імітаційного моделювання у такому форматі є складним і вимагає додаткової обробки даних.

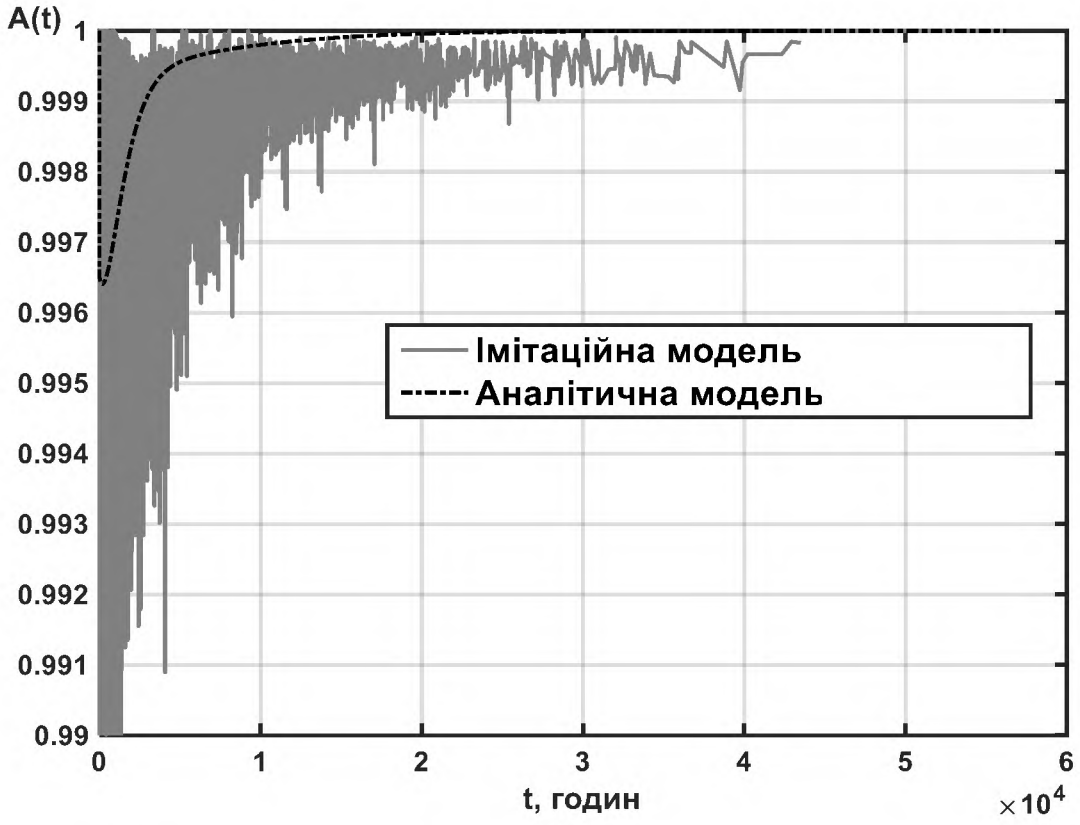
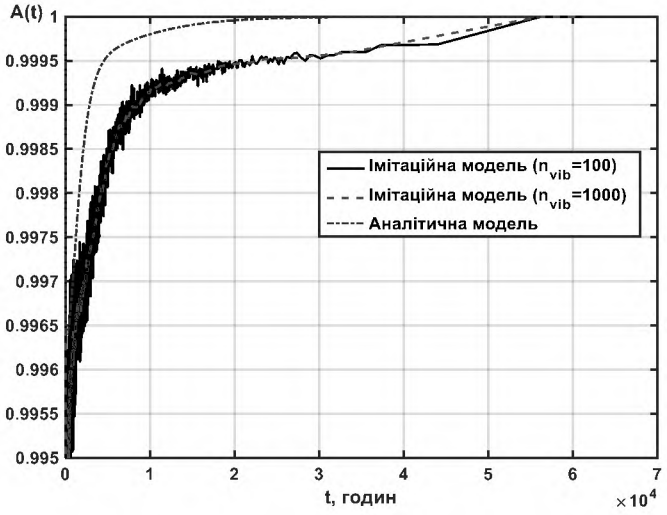


Рисунок 3.5 – Порівняння результатів аналітичного моделювання і всіх повних розіграшів Монте-Карло моделі ІС

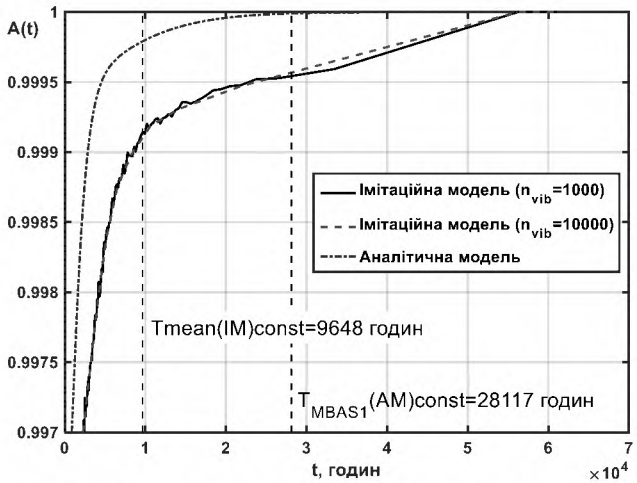
На основі отриманих часових відліків і значень функції готовності формується масив Ai. На початковому етапі необхідно відсікти детерміновані значення в точках t = 0 і t = Taim_interval, де A(t) = 1. Для цього виконується унікалізація елементів матриці (Ai = unique(Ai,'rows')) та видалення першого й останнього рядків: Ai(1,:) = []; Ai(end,:) = [].

У процесі статистичної обробки елементи матриці Ai групуються у вибірки, розмір яких визначається змінною n_vibor. Для кожної вибірки обчислюються середні значення часового відліку та функції готовності за допомогою функції mean(X). Результати обробки показані на рис. 3.6 для різних розмірів вибірок n_vibor. Збільшення розміру вибірки дозволяє отримати більш гладкі графіки, що зручно для порівняння результатів аналітичного та імітаційного моделювання.

На рис. 3.6, б у масштабі часової вісі також наведено результуючий показник TMBAS1const аналітичної моделі та середнє значення tconst для повних розіграшів Монте-Карло. Графік демонструє, що TMBAS1const знаходиться між значеннями mean(tconst) та max(tconst).



а)



б)

Рисунок 3.6 – Порівняння результатів обробки генеральної сукупності повних розіграшів Монте-Карло з різним обсягом вибірок (а – 100 і 1000; б – 1000 і 10000) і аналітичної моделі ІС

Для спрощення порівняння аналітичної та імітаційної моделей здійснюється інтерполяція функції готовності аналітичної моделі на часових відліках імітаційної моделі за допомогою функції interp1. Отримані значення tconst формують розподіл у діапазоні [367.7…5.1576e+04], який відрізняється від нормального та представлений на рис. 3.7. Аналіз розподілу показав, що найкращим чином йому відповідає гамма-розподіл.

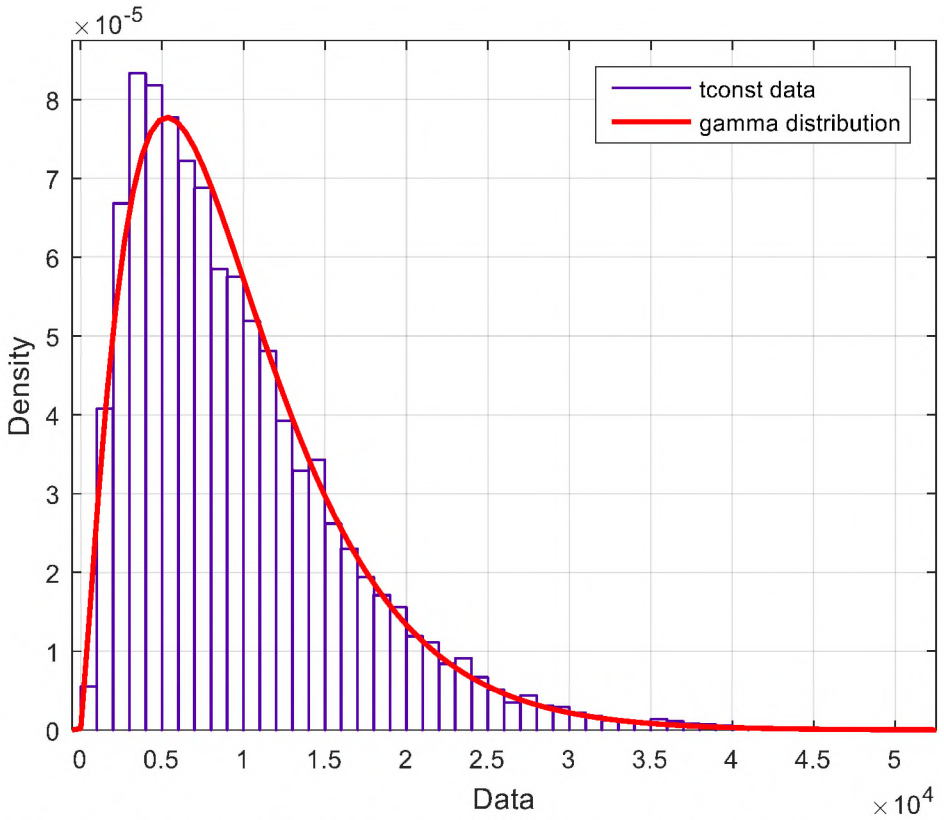


Рисунок 3.7 – Функція розподілу щільності ймовірностей результуючого показника tconst та її представлення гамма-розподілом

Параметри гамма-розподілу були визначені методом максимальної правдоподібності. Їх значення: a = 2.22245 (± 0.0293806) та b = 4341.19 (± 64.3568). Графік функції розподілу наведений на рис. 3.7. Для mean(tconst) та max(tconst) були також визначені значення функції готовності аналітичної моделі із зазначенням похибок.

Другий результуючий показник Amin, що моделюється на множині [0.0078…0.9996], має складний розподіл, який представлений на рис. 3.8. Середнє

значення mean(Amin) = 0.9808 демонструє відносну похибку 1.57% порівняно з мінімальним значенням функції готовності аналітичної моделі AM1min = 0.9964.

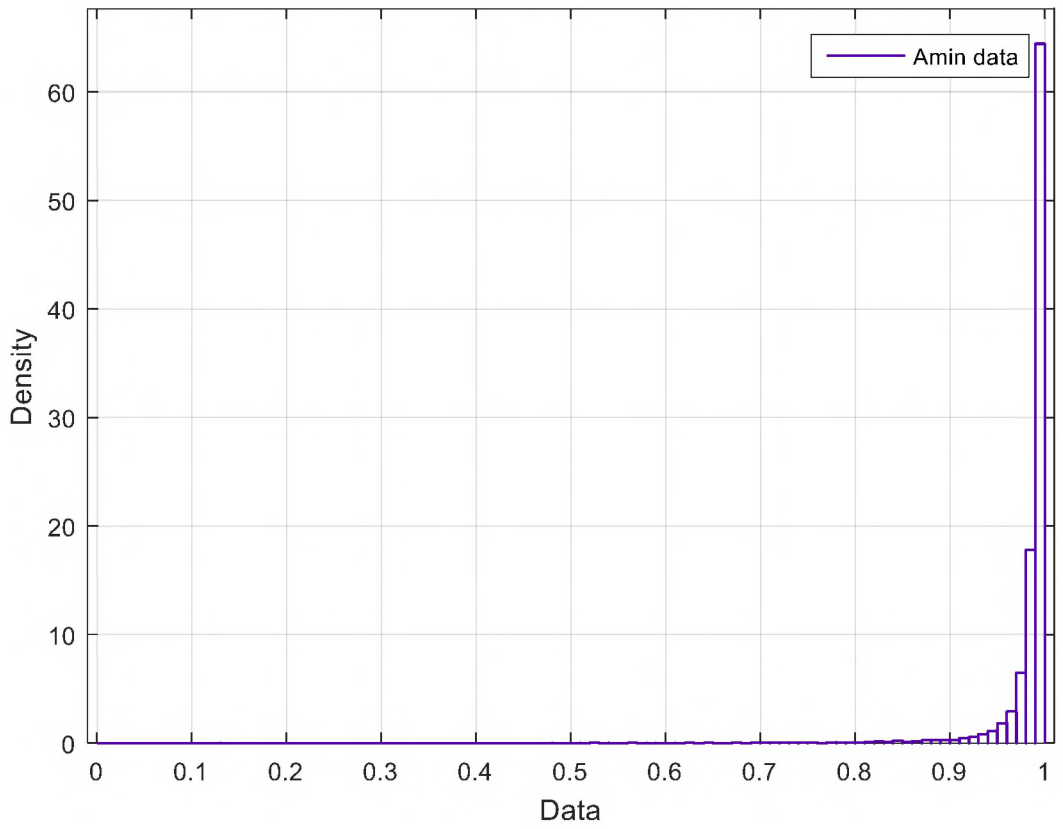


Рисунок 3.8 – Функція розподілу щільності ймовірностей розіграшів результуючого показника Amin моделі ІС

Досліджено також моделі інформаційної системи для графів без поглинаючих станів. У таких моделях необхідно враховувати час першого переходу системи у стан S9 (при Nd = 2, Nv = 2). Цей підхід реалізується через допоміжну змінну tc, яка фіксує момент входження в стан S9. Умови закінчення розіграшу також змінюються – розіграш завершується лише після перевищення суми часових інтервалів значення Taim_interval.

Результати моделювання демонструють, що імітаційна модель зближується з аналітичною при $t \to \infty$. Параметри логнормального розподілу для tconst також були визначені, їх значення: mu = 6.75208 (± 0.0208161) та sigma = 0.658262 (± 0.0147302). Функція розподілу представлена на рис. 3.9.

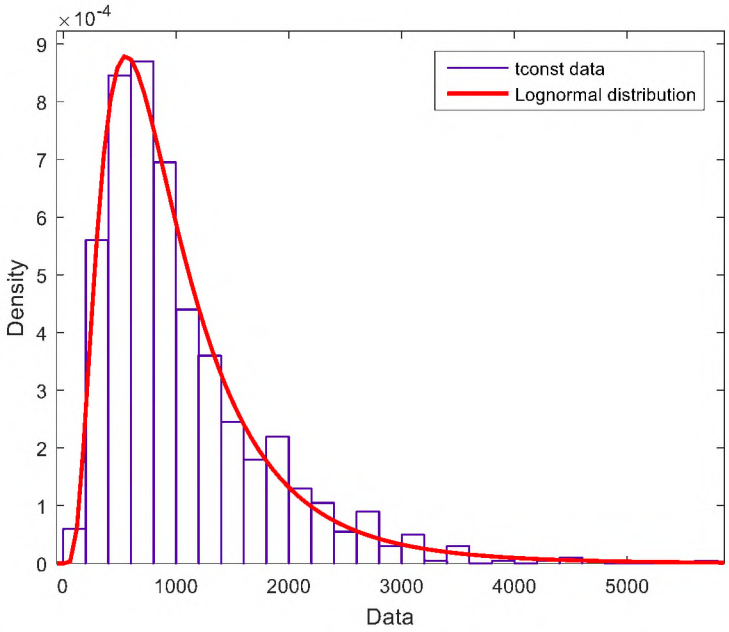


Рисунок 3.9 – Функція розподілу щільності ймовірностей результуючого показника tconst та її представлення логнормальним розподілом

Другий результуючий показник Amin моделювався у діапазоні [0.0362…0.9794], його розподіл показаний на рис. 3.10. Середнє значення mean(Amin) = 0.8761 демонструє відносну похибку 8.6% порівняно з мінімальним значенням функції готовності аналітичної моделі AM2min = 0.9619.

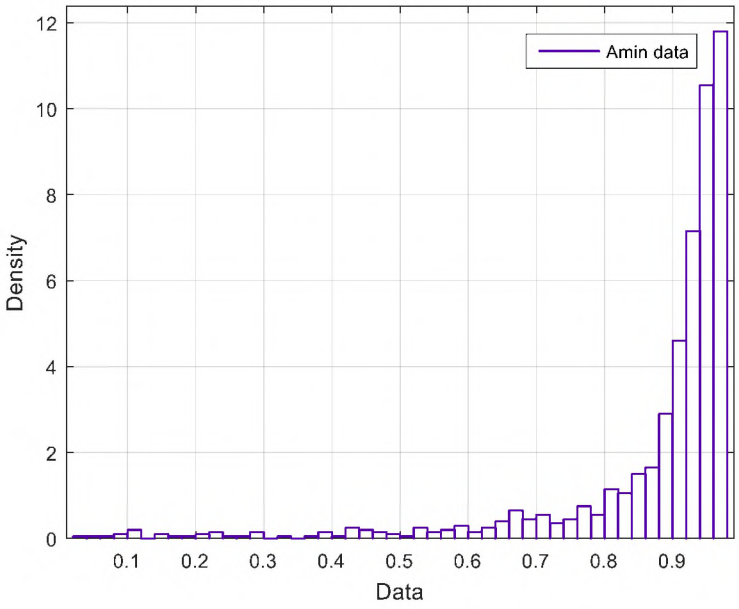


Рисунок 3.10 – Функція розподілу щільності ймовірностей розіграшів результуючого показника Amin моделі з урахуванням проведення обмеженої кількості загальних обслуговувань

Запропонований алгоритм імітаційного моделювання дозволив ефективно дослідити поведінку складних систем із поглинаючими та непоглинаючими станами. Проведений аналіз показав, що отримані значення функції готовності добре узгоджуються з аналітичними моделями. Застосування імітаційного підходу є доцільним для оцінки показників систем, коли аналітичні методи обмежені складністю моделі. Подальші дослідження можуть бути спрямовані на оптимізацію вибірок та покращення точності оцінки параметрів функцій розподілу.

3.5 Вибір та обґрунтування методики оцінки економічної ефективності

Аналіз основних показників порівняльної ефективності створення та функціонування інформаційних систем включає визначення наступних показників.

1) річна економія поточних витрат, отримана від функціонування системи;

2) додаткові капітальні вкладення (КВ), необхідні для створення системи;

3) термін окупності додаткових КВ;

4) розрахунковий коефіцієнт ефективності додаткових КВ;

5) річний економічний ефект;

6) річна економія трудових витрат на обробку даних у системі.

Оцінимо ці показники.

1. Річна економія поточних витрат складається з прямої економії та інших складових. Пряма економія виникає від автоматизації обробки інформації.

$$\Delta Ст = \Delta Сп + \Delta Сн, \tag{3.1}$$

де $\Delta Сп$ – пряма економія,

$\Delta Сн$ – непряма економія (у цій роботі не розраховується).

Пряма економія, яку отримують від автоматизації обробки інформації розраховується так:

$$\Delta Сп = \Delta Сб – \Delta Спор, \tag{3.2}$$

де ΔСб – базовий період, який береться до застосування системи,

ΔСпор – порівнюваний період, коли система працює у режимі автоматизації.

Розрахунок показника за порівнюваний період містить:

$$\Delta Спор = С1 + С2 + С3 + С4 + С5 + Спр, \quad (3.3)$$

де С1 – витрати на оплату праці персоналу;

С2 – нарахування на фонд оплати праці (податок 22%);

С3 – витрати сировину, матеріали (картриджі, папір, комплектуючі);

С4 – амортизація обладнання, зазвичай розглядається як лінійна з терміном служби від 3 до 8 років;

С5 – інші витрати (витрати на відрядження, інформаційні витрати, плата за кредит, податки, представницькі витрати);

Спр – передвиробничі витрати, які потрібно визначити додатково.

Передвиробничі витрати – витрати, які можуть бути направлені на розробку (купівлю) програмних засобів, на навчання фахівців і т.д.

2. Додаткові капітальні вкладення охоплюють будівництво, оренду приміщення, ремонт, придбання мережевого обладнання тощо.

3. Термін окупності капітальних вкладень розраховується за формулою:

$$Т = КД / \Delta Ст, \quad (3.4)$$

де ΔСт – річна економія поточних витрат,

КД – капітальні вкладення, наведені до 1 року

4. Розрахунковий коефіцієнт ефективності Ер є величина зворотна Т, Ер = 1/Т. Ен = 0,33 – нормативний коефіцієнт ефективності.

Якщо розрахунковий коефіцієнт більший чи дорівнює Ен, то проєкт приймається до впровадження, тобто створення АІС ефективно.

5. Річний економічний ефект розраховується за формулою:

$$Е = \Delta Ст – КД \cdot Ен , Ен = 0,15 \quad (3.5)$$

6.Річна економія трудових витрат розраховується як різниця між базовим та порівняльним періодами до та після впровадження системи в режимі автоматизації:

$$\Delta T = \Delta Tб – \Delta Tпор, \quad (3.6)$$

де ΔТб – період базовий до застосування системи,

ΔТпор – період, порівнюваний, тобто період роботи системи у режимі автоматизації.

Далі було виконано розрахунок основних економічних показників від впровадження інформаційної системи. Результати розрахунків представлені в таблиці 3.1.

Таблиця 3.1 – Результати розрахунку базових економічних показників

Затрати	Базовий період (ΔСб), грн.	Порівнюваний період (ΔСпор), грн.
С1	420000	300000
С2	144900	103500
С3	1250	2500
С4	0	1250
С5	500	500
Спр		1000
Всього	566650	408750

ΔСп = 566650- 408750 = 157900 грн.

ΔСт = 157900 грн.

2. Додаткові капітальні вкладення.

Оренда приміщення (на рік) = 5000 грн · 60 $м^2$ = 300000 грн.

Придбання комп’ютера (Комплектація: AMD Phenom II X6 1075, 8Гб, 2000Гб, GeForce GTX 560) = 12000 грн.

Придбання монітора (17» Acer V173Ab чорний 5ms) = 2500 грн.

Придбання принтера Canon LBP-6000 = 3400 грн.

Термін функціонування системи = 10 років.

Сума додаткових капітальних вкладень за рік

$$300000 + (12000 + 2500 + 3400) / 10 = 301790 \text{ грн.}$$

3. Термін окупності капітальних вкладень:

$$T = 301790 / 157900 = 1,91$$

4. Розрахунковий коефіцієнт ефективності Ер.

$$Ep = 1/1,91 = 0,524$$

Проєкт ефективний.

5. Річний економічний ефект

$$E = 157900 - 301790 \cdot 0,15 = 112631,5 \text{ грн.}$$

6. Річна економія трудових витрат:

$$\Delta T = 420000 - 300000 = 120000 \text{ грн.}$$

Розрахунок показав, що застосування системи принесе підприємству вигоду, ефект від використання системи становитиме 112631,5 грн на рік.

Висновки до розділу 3

У третьому розділі розроблено та досліджено імітаційну модель функціонування інформаційної системи в умовах атак на її готовність. Проаналізовано обмеження аналітичного моделювання та визначено необхідність використання імітаційних підходів для більш точного врахування динаміки змін коефіцієнта готовності інформаційних систем. Виявлено, що імітаційне моделювання дозволяє досліджувати системи з урахуванням відмов, відновлень, оновлень програмного забезпечення та атак.

Виконано розробку алгоритмів імітаційного моделювання, реалізованих мовами програмування високого рівня та у середовищі MATLAB. Розроблені алгоритми враховують випадкові часові інтервали між подіями, такими як відмови або відновлення компонентів системи, з використанням спеціальних розподілів ймовірностей. Застосовано метод послідовного моделювання для оцінки коефіцієнта готовності з урахуванням похибки та довірчої ймовірності.

Проаналізовано результати імітаційного моделювання для різних сценаріїв, включаючи атаки на службу DNS та інші компоненти системи. Порівняння з аналітичними моделями показало, що імітаційні моделі забезпечують високу відповідність у стаціонарних умовах, хоча можуть демонструвати специфічну поведінку у перехідній фазі.

Розглянуто сценарії моделювання для систем із поглинаючими та непоглинаючими станами. У таких моделях імітаційний підхід дозволив детально дослідити поведінку системи у перехідний період і під час впливу зовнішніх атак. Визначено параметри, що найбільше впливають на готовність системи, зокрема інтенсивність відмов та частота профілактичних заходів.

Досліджено методики оцінки економічної ефективності впровадження інформаційних систем. Виконано розрахунок основних економічних показників, таких як річна економія витрат, капітальні вкладення, термін окупності та коефіцієнт ефективності. Розрахунки показали, що застосування системи є економічно доцільним і приносить значний річний економічний ефект.

Таким чином, у розділі виконано комплексний аналіз імітаційного моделювання інформаційних систем, який продемонстрував переваги цього підходу для оцінки надійності та ефективності функціонування інформаційних систем. Отримані результати можуть бути використані для оптимізації параметрів роботи системи та підвищення її стійкості до зовнішніх впливів.

ВИСНОВКИ

У роботі була поставлена і вирішена актуальна задача дослідження надійності та готовності інформаційних систем у хмарних архітектурах на прикладі Microsoft Azure, із застосуванням аналітичних і імітаційних методів.

1. Виконано аналіз сучасного стану архітектур хмарних платформ, зокрема платформи Windows Azure. Проаналізовано її компоненти, підтримувані стандарти та протоколи, такі як SOAP, HTTP, XML. Розглянуто історію розвитку технологій для вебсервісів, а також особливості хмарної інфраструктури для забезпечення надійності інформаційних систем.

2. Вивчено методи оцінки готовності інформаційних систем на основі марковських моделей. Побудовано модель, яка враховує різні стани системи (готовність, відмова, профілактика). Розглянуто сценарії поведінки інформаційних систем при відмовах та визначено інтенсивності переходів між станами. Виконано декомпозицію системи для детального аналізу її функціонування.

3. Розроблено імітаційну модель для аналізу готовності інформаційних систем у випадках відмов та атак. Модель дозволяє досліджувати динамічні зміни у системі, оцінюючи ефективність профілактичних заходів та вплив атак на компоненти. Для дослідження використано метод Монте-Карло та систему рівнянь Колмогорова.

4. Проведено порівняння аналітичних та імітаційних підходів до моделювання надійності систем. Виявлено, що імітаційні методи дозволяють враховувати складність архітектури та динамічні фактори, які аналітичними методами оцінити складніше.

5. Розглянуто основні методи захисту інформаційних систем від відмов, зокрема резервні архітектури, реплікація даних та підвищення відмовостійкості баз даних. Проведено аналіз економічної ефективності застосування системи та її окупності.

6. Проведений аналіз дозволив зробити висновок, що комбінація аналітичного та імітаційного підходів є найбільш ефективною для дослідження

надійності складних систем. Імітаційне моделювання показало високу адекватність результатів і дало змогу оцінити поведінку системи в динаміці.

На основі отриманих висновків запропоновано наступні рекомендації щодо покращення надійності інформаційних систем: збільшення частоти профілактичних заходів, удосконалення механізмів реплікації даних, оптимізація інтенсивності обробки запитів. Таким чином, поставлені задачі розв'язано у повному обсязі. Напрямком подальших досліджень є оптимізація параметрів імітаційних моделей для різних архітектур інформаційних систем.