МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ВІННИЦЬКИЙ ВІКТОР ВІКТОРОВИЧ

Допускається до захисту:
Завідувач кафедри
інформаційних технологій,
к.т.н., доцент
_____Нескородєва Т. В.
«__»_____20__р.

«ЗАХИСТ ІНФОРМАЦІЇ У БАГАТОПРОЦЕСОРНИХ СИСТЕМАХ»

Спеціальність 125 Кібербезпека

Кваліфікаційна (бакалаврська) робота

Науковий керівник:

Крижановський В. Г.,
професор кафедри
інформаційних технологій
д-р.т.н., професор
_____
(підпис)

Оцінка : _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця 2021

АНОТАЦІЯ

Вінницький В. В. Захист інформації у багатопроцесорних системах Спеціальність 125 Кібербезпека. Донецький національний університет імені Василя Стуса, Вінниця, 2021.

У кваліфікованій (бакалаврській) роботі розглянуто методи захисту інформації у багатопроцесорних системах за допомогою ізоляції. Як засіб ізоляції обрана програмна система Docker. Для програмної системи Docker розроблено дві надбудови для захисту від несанкціонованого переходу пристроїв вводу та захисту програмної системи Docker від несанкціонованого фізичного доступу до обладнання інформаційної системи. Ключові слова: Контейнер, ізоляція, Docker, несанкціонований.

66 с., 3 рис., 46 джерел.

ABSTRACT

Vinnytsky VV Information protection in multiprocessor systems Specialty 125 Cybersecurity. Vasyl Stus Donetsk National University, Vinnytsia, 2021.

In qualified (bachelor's) work methods of information protection in multiprocessor systems by means of isolation are considered. The Docker software system was chosen as a means of isolation. Two add-ons have been developed for the Docker software system to protect against unauthorized transitions of input devices and to protect the Docker software system from unauthorized physical access to information system equipment. Keywords: Container, insulation, Docker, unauthorized.

66, pg., 3 figs., 46 origin.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

МВС	Міністерство внутрішніх справ
IT	Інформаційні технології
SMP	Symmetric Multi-Processing
NUMA	Non-Uniform Memory Architecture
MPP	Massively Parallel Processing
Clusters	Кластери
Constellations	Сузір'я
ІС	Інформаційна система
ОС	Операційна система
СУБД	Система управління базами даних
IBM	International Business Machines
cc-NUMA	cache coherent NUMA

ВСТУП

Актуальність теми: На сучасному етапі все більшу роль в подальшому розвитку інформаційних ресурсів грають паралельні обчислювальні системи і обчислення. Подібні системи знаходять застосування в сфері економічних, технологічних та інших процесів.

У зв'язку з їх розвитком, впровадженням та вдосконаленням широкого поширення набули методи завдання шкоди таким ресурсам. Найбільший інтерес викликають проблеми дослідження методів і засобів захисту інформації в паралельних обчислювальних процесах.

Вивчення і розробка подібної проблематики надасть можливості для подальшого розвитку нових і вже існуючих методів захисту інформації. Використання існуючих алгоритмів та їх доопрацювання дозволяють провести оптимізацію, а також подальшу програмну і апаратну реалізацію.

Ефективність засобів захисту від зовнішніх і внутрішніх впливів, що викликають порушення цілісності інформації та відмови резервованих обчислювальних комплексів, багато в чому визначається вартістю заходів, які повинні забезпечити високошвидкісний доступ до пам'яті, що зберігає важливі для обчислювального процесу дані і результати обчислень.

Доступ до ресурсів обчислювальних комплексів, дозволяє прискорити контроль і виявлення небезпечних станів, мінімізувати їх ймовірності і збільшити ефективність процесу зниження ризику при компенсації помилкового функціонування обчислювального процесу, втрати доступності та цілісності даних.

Висока надійність, стійкість, функціональна та інформаційна безпека систем критичного застосування досягається при резервуванні основних підсистем і комплектації системи засобами захисту від зловмисних і випадкових зовнішніх і внутрішніх впливів, які потенційно викликають відмови, зниження стійкості функціонування, порушення цілісності та доступності даних.

Підвищення функціональної і інформаційної безпеки систем досягається при мінімізації ймовірності і наслідків виникнення небезпечних станів, що виникають в результаті відмов, збоїв або зовнішніх зловмисних або випадкових (ненавмисних) дестабілізуючих впливів.

В даний час сфера застосування багатопроцесорних систем безперервно розширюється, охоплюючи все нові області в різних галузях науки, бізнесу і виробництва. Стрімкий розвиток кластерних систем створює умови для використання багатопроцесорної обчислювальної техніки в реальному секторі економіки.

Потреба вирішення складних прикладних задач з великим об'ємом обчислень і принципова обмеженість максимального швидкодії "класичних" - за схемою фон Неймана - ЕОМ привели до появи багатопроцесорних обчислювальних систем (МВС). Використання таких засобів обчислювальної техніки дозволяє істотно збільшувати продуктивність ЕОМ при будь-якому існуючому рівні розвитку комп'ютерного обладнання.

Мета роботи: метою даної дипломної роботи є дослідження способів захисту інформації багатопроцесорних систем від кібератак та розробка програмної системи для захисту та ізоляції застосунків в багатопроцесорних системах. Виходячи з поставленої мети в даній дипломній роботі, до виконання випливають наступні **завдання:**

- визначити поняття багатопроцесорних систем;
- охарактеризувати історію розвитку багатопроцесорних систем;
- висвітлити види і класифікацію багатопроцесорних систем;
- описати Архітектуру NUMA-Q, CMP, NUMAflex та FAME;
- проаналізувати основні види кіберзагроз;
- дослідити найчастіші загрози;
- висвітлити методи боротьби з кіберзагрозами;
- визначити класифікацію стандартів кібербезпеки;
- висвітлити рекомендації щодо використання;
- подати приклади використання;

- розглянути програмні системи ізоляції застосунків у багатопроцесорних системах;
- розробити програмну надбудову над системою ізоляції застосунків для підсилення безпеки останньої

Предмет роботи: предметом даної дипломної роботи є характеристика процесів та особливостей захисту інформації багатопроцесорних систем від кібератак.

Об’єкт роботи: об’єктом даної дипломної роботи є захист інформації багатопроцесорних систем за допомогою систем ізоляції застосунків.

Структура роботи: дана наукова праця складається з вступу, чотирьох розділів, висновків та списку використаних джерел.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Поняття багатопроцесорних систем

З ростом числа процесорів просто неможливо обійти необхідність реалізації моделі розподіленої пам'яті з високошвидкісною мережею для зв'язку процесорів. З швидким зростанням продуктивності процесорів і пов'язаним з цим посиленням вимоги збільшення пропускної здатності шини пам'яті, масштаб систем (тобто число процесорів в системі, для яких потрібна організація розподіленої пам'яті), зменшується, так само як і зменшується число процесорів, які вдається підтримувати на одній спільній шині загальної пам'яті [21, с. 103].

Головною перевагою систем з роздільною пам'яттю є хороша масштабованість: на відміну від SMP-систем в машинах з роздільною пам'яттю кожен процесор має доступ тільки до своєї локальної пам'яті, у зв'язку з чим не виникає необхідності в початковій синхронізації процесорів.

недоліки:

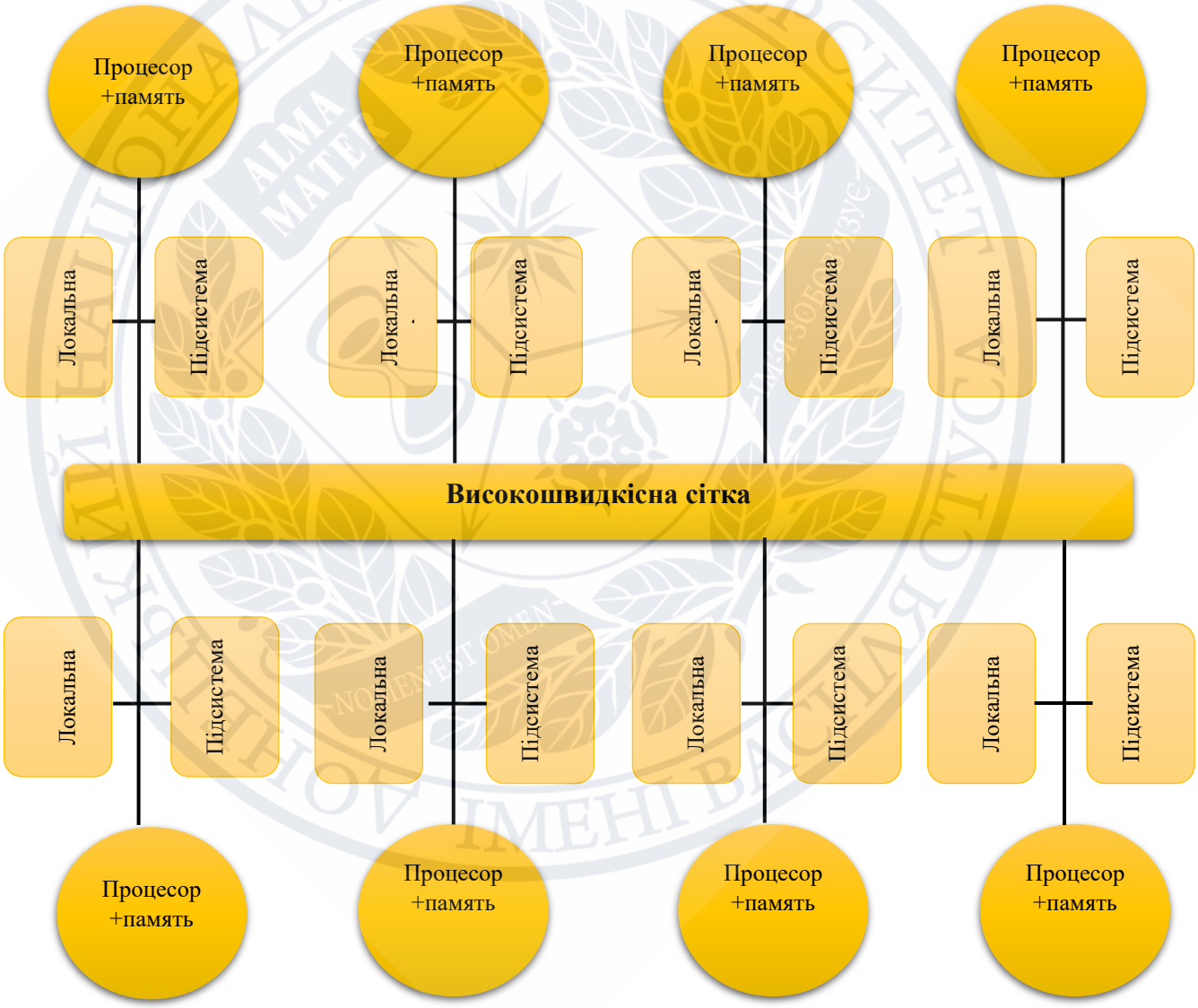
• відсутність спільної пам'яті помітно знижує швидкість між процесорного обміну, оскільки немає загального середовища для зберігання даних, призначених для обміну між процесорами. Потрібна спеціальна техніка програмування для реалізації обміну повідомленнями між процесорами;

• кожен процесор може використовувати тільки обмежений обсяг локального банку пам'яті;

• внаслідок зазначених архітектурних недоліків потрібні значні зусилля для того, щоб максимально використовувати системні ресурси. Саме цим визначається висока ціна програмного забезпечення для масивно-паралельних систем з роздільною пам'яттю [23, с. 139].

Системами з роздільною пам'яттю є суперкомп'ютери МВС-1000, IBM RS / 6000 SP, SGI / CRAY T3E, системи ASCI, Hitachi SR8000, системи Parsytec. Машини останньої серії CRAY T3E від SGI, засновані на базі процесорів Dec Alpha 21164 з піковою продуктивністю 1200 Мфлопс / с (CRAY T3E-1200), здатні масштабуватись до 2048 процесорів. При роботі з MPP системами використовують так звану Massive Passing Programming Paradigm - парадигму програмування з передачею даних (MPI, PVM, BSPlib) [21, с. 104].

Рисунок 1.1. Багатопроцесорна система з розподіленою пам'яттю



В процесі розвитку супер-комп'ютерних технологій ідею підвищення продуктивності обчислювальної системи за рахунок збільшення числа процесорів використовували неодноразово. Якщо не вдаватися в історичний

екскурс і обговорення всіх таких спроб, то можна в такий спосіб коротко описати розвиток подій.

У системах, створених на базі архітектури симетричної мультипроцесорності, у кожного процесора з'являється можливість виконувати будь-яке завдання незалежно від місцезнаходження даних обробки для цього завдання. У таких системах завдання переміщаються між процесорами, забезпечуючи ефективний розподіл навантаження [23, с. 140-141].

Така архітектура - один з найпростіших способів для масштабування системи, підвищення переносимості вихідного коду і швидкості його розробки. Недоліком таких систем є обмеження числа процесорів в системі, при їх збільшенні зростає навантаження на загальну шину, і внаслідок цього загальна продуктивність падає. Крім того, недоліком можна вважати певну складність інструментів розробки і засобів (Компілятор, операційна система, засоби налагодження), що надають необхідний для реалізації такої системи функціонал.

У асиметричній багатопроцесорній архітектурі кожен процесор може використовуватися для вирішення відмінною від інших процесорів завдання. явним недоліком такої концепції є низька масштабованість системи, складність забезпечення рівномірного розподілу загального навантаження. Однак такий підхід накладає менше вимог до засобів розробки і налагодження та, в загальному, є більш простий альтернативою симетричною мультипроцесорності [21, с. 105-106].

Ефективність використання комп'ютерів у вирішальній мірі залежить від складу і якості програмного забезпечення, встановленого на них. В першу чергу це стосується програмного забезпечення, призначеного для розробки прикладних програм. Так наприклад, недостатня розвиненість таких коштів для систем МРР була стримуючим фактором для їх широкого використання.

В даний час ситуація змінилася, і завдяки кластерним технологіям MPP системи стали найпоширенішим і доступним різновидом високопродуктивних обчислювальних систем.

У багатопроцесорних системах цього типу на кожному обчислювальному вузли функціонує власні копії операційної системи, під управлінням яких виконуються незалежні програми. Це можуть бути як дійсно незалежні програми, так і паралельні гілки однієї програми. В цьому випадку єдино можливим механізмом взаємодії між ними є механізм передачі повідомлень [23, с. 142].

Прагнення домогтися максимальної продуктивності змушує розробників при реалізації механізму передачі повідомлень враховувати особливості архітектури багатопроцесорної системи. Це сприяє написання більш ефективних, але орієнтованих на конкретний комп'ютер програм. Разом з тим незалежними розробниками програмного забезпечення було запропоновано безліч реалізацій механізму передачі повідомлень, незалежних від конкретної платформи.

Якщо традиційно МВС застосовувалися в основному в науковій сфері для вирішення обчислювальних завдань, що вимагають потужних обчислювальних ресурсів, то зараз через бурхливий розвитку бізнесу різко зросла кількість компаній, що відводять використанню комп'ютерних технологій та електронному документообігу головну роль [21, с. 107].

У зв'язку з цим невпинно зростає потреба в побудові централізованих обчислювальних систем для критично важливих додатків, пов'язаних з обробкою транзакцій, управлінням базами даних і обслуговуванням телекомунікацій. Можна виділити дві основні сфери застосування описуваних систем: обробка транзакцій в режимі реального часу (OLTP, on-line transaction processing) і створення сховищ даних для організації систем підтримки прийняття рішень (Data Mining, Data Warehousing, Decision Support System).

Система для глобальних корпоративних обчислень - це, перш за все, централізована система, з якою працюють практично всі користувачі в

корпорації, і, відповідно, вона повинна весь час перебувати в робочому стані. Як правило, рішення подібного рівня встановлюють в компаніях і корпораціях, де навіть короткочасні простої мережі можуть привести до величезних збитків.

Тому, для організації такої системи не підходить звичайний сервер зі стандартною архітектурою, цілком придатний там, де немає жорстких вимог до продуктивності і часу простою. Високопродуктивні системи для глобальних корпоративних обчислень повинні відрізнятися такими характеристиками, як підвищена продуктивність, масштабованість, мінімально допустимий час простою.

Поряд з розширенням сфери застосування у міру вдосконалення МВС відбувається ускладнення і збільшення кількості завдань в областях, традиційно використовують високопродуктивну обчислювальну техніку. В даний час виділено коло фундаментальних і прикладних проблем, ефективне вирішення яких можливе тільки з використанням надпотужних обчислювальних ресурсів [23, с. 143].

Головною відмінною рисою багатопроцесорної обчислювальної системи є її продуктивність, тобто кількість операцій, вироблених системою за одиницю часу. Розрізняють пікову і реальну продуктивність. Під піковою розуміють величину, рівну твору пікової продуктивності одного процесора на число таких процесорів в даній машині. При цьому передбачається, що всі пристрої комп'ютера працюють в максимально продуктивному режимі [21, с. 108-109].

Пікова продуктивність комп'ютера обчислюється однозначно, і ця характеристика є базовою, по якій роблять порівняння високопродуктивних обчислювальних систем. Чим більше пікова продуктивність, тим (теоретично) швидше користувач зможе вирішити свою задачу.

Пікова продуктивність є величина теоретична і, взагалі кажучи, недосяжна при запуску конкретного додатка. Реальна ж продуктивність, що досягається на даному додатку, залежить від взаємодії програмної моделі, в

якій реалізовано додаток, з архітектурними особливостями машини, на якій додаток запускається.

Існує два способи оцінки пікової продуктивності комп'ютера. Один з них спирається на число команд, виконуваних комп'ютером за одиницю часу. Одиницею виміру, як правило, є MIPS (Million Instructions Per Second). Продуктивність, виражена в MIPS, говорить про швидкість виконання комп'ютером своїх же інструкцій.

Але, по-перше, заздалегідь не ясно, в скільки інструкцій відобразиться конкретна програма, а по-друге, кожна програма має свою специфіку, і число команд від програми до програми може змінюватися дуже сильно. У зв'язку з цим, дана характеристика дає лише саме загальне уявлення про продуктивність комп'ютера [21, с. 110-111].

Інший спосіб вимірювання продуктивності полягає у визначенні числа речових операцій, які виконуються комп'ютером за одиницю часу. Одиницею вимірювання є Flops (Floating point operations per second) - число операцій з плаваючою точкою, вироблених комп'ютером за одну секунду. Такий спосіб є більш прийнятним для користувача, оскільки йому відома обчислювальна складність програми і, користуючись цією характеристикою, користувач може отримати нижню оцінку часу її виконання.

Однак пікова продуктивність виходить тільки в ідеальних умовах, тобто при відсутності конфліктів при зверненні до пам'яті при рівномірного завантаження всіх пристроїв. У реальних умовах на виконання конкретної програми впливають такі апаратно-програмні особливості даного комп'ютера, як: особливості структури процесора, системи команд, склад функціональних пристроїв, реалізація введення / виведення, ефективність роботи компіляторів [23, с. 144].

Одним з визначальних чинників є час взаємодії з пам'яттю, яке визначається її будовою, об'ємом і архітектурою підсистем доступу в пам'ять. У більшості сучасних комп'ютерів в якості організації найбільш ефективного

доступу до пам'яті використовується так звана багаторівнева ієрархічна пам'ять [23, с. 145].

Як рівнів використовуються регістри і реєстрова пам'ять, основна оперативна пам'ять, кеш-пам'ять, віртуальні і жорсткі диски, стрічкові роботи. При цьому витримується наступний принцип формування ієрархії: при підвищенні рівня пам'яті швидкість обробки даних повинна збільшуватися, а обсяг рівня пам'яті - зменшуватися.

Ефективність використання такого роду ієрархії досягається за рахунок зберігання часто використовуваних даних в пам'яті верхнього рівня, час доступу до якої мінімально. А оскільки така пам'ять обходиться досить дорого, її обсяг не може бути більшим. Ієрархія пам'яті відноситься до тих особливостей архітектури комп'ютерів, які мають величезне значення для підвищення їх продуктивності [21, с. 112].

1.2. Історія розвитку

В процесі розвитку супер-комп'ютерних технологій ідею підвищення продуктивності обчислювальної системи за рахунок збільшення числа процесорів використовували неодноразово. Якщо не вдаватися в історичний екскурс і обговорення всіх таких спроб, то можна таким чином коротко описати розвиток подій [23, с. 19].

Експериментальні розробки зі створення багатопроцесорних обчислювальних систем почалися в 70-х роках 20 століття. Однією з перших таких систем стала розроблена в Іллінойському університеті МВС ILLIAC IV, яка включала 64 (у проекті до 256) процесорних елемента (ПЕ), які працюють за єдиною програмою, яка застосовується до вмісту власної оперативної пам'яті кожного ПЕ. Обмін даними між процесорами здійснювався через спеціальну матрицю комунікаційних каналів. Зазначена особливість комунікаційної системи дала назву "матричні суперкомп'ютери" відповідного класу МВС. Відзначимо, що більш широкий клас МВС з розподіленою

пам'яттю і з довільною комунікаційною системою отримав згодом назву "багатопроцесорні системи з масовим паралелізмом", або МВС з МРР-архітектурою (МРР - Massively Parallel Processing). При цьому, як правило, кожен з ПЕ МРР системи є універсальним процесором, що діє за своєю власною програмою (на відміну від загальної програми для всіх ПЕ матричної МВС) [23, с. 20-21].

Перші матричні МВС випускалися буквально поштучно, тому їх вартість була фантастично високою. Серійні ж зразки подібних систем, такі як ICL DAP, що включали до 8192 ПЕ, з'явилися значно пізніше, проте не набули широкого поширення через складність програмування МВС з одним потоком управління (з однією програмою, загальною для всіх ПЕ).

Перші промислові зразки мультипроцесорних систем з'явилися на базі векторно-конвеєрних комп'ютерів в середині 80-х років. Найбільш поширеними МВС такого типу були суперкомп'ютери фірми Cray. Однак такі системи були надзвичайно дорогими і вироблялися невеликими серіями. Як правило, в подібних комп'ютерах об'єднувалося від 2 до 16 процесорів, які мали рівноправний (симетричний) доступ до спільної оперативної пам'яті. У зв'язку з цим вони отримали назву симетричні мультипроцесорні системи (Symmetric Multi-Processing - SMP).

Як альтернатива таким дорогим мультипроцесорним системам на базі векторно-конвеєрних процесорів була запропонована ідея будувати еквівалентні за потужністю багатопроцесорні системи з великого числа дешевих мікропроцесорів, що серійно випускаються. Однак дуже скоро виявилося, що SMP архітектура має дуже обмежені можливості з нарощування числа процесорів в системі через різке збільшення числа конфліктів при зверненні до спільної шині пам'яті. У зв'язку з цим виправданою представлялася ідея забезпечити кожен процесор власною оперативною пам'яттю, перетворюючи комп'ютер у об'єднання незалежних обчислювальних вузлів.

Такий підхід значно збільшив ступінь масштабованості багатопроцесорних систем, але в свою чергу зажадав розробки спеціального способу обміну даними між обчислювальними вузлами, реалізованого зазвичай у вигляді механізму передачі повідомлень (Message Passing). Комп'ютери з такою архітектурою є найбільш яскравими представниками MPP систем. В даний час ці два напрямки (або якісь їхні комбінації) є домінуючими в розвитку супер-комп'ютерних технологій [23, с. 22-23].

1.3. Види і класифікація

Основною ознакою векторно-конвеєрних систем є наявність спеціальних векторно-конвеєрних процесорів, в яких передбачені команди однотипної обробки векторів незалежних даних, ефективно виконуються на конвеєрних функціональних пристроях [24].

• SMP (Symmetric MultiProcessor) – симетричні мультипроцесорні системи;

• NUMA (Non-Uniform Memory Architecture) мультипроцесорні системи з неоднорідним доступом до пам'яті;

• MPP (Massively Parallel Processing) – масово паралельні системи

• Clusters (кластери) - набори повноцінних серверів, об'єднаних в єдину обчислювальну систему

• Constellations (сузір'я) – асоціації мультипроцесорів.

Велика розмаїтість багатопроцесорних обчислювальних систем породило природне бажання ввести для них якусь класифікацію. Ця класифікація має однозначно відносити ту чи іншу обчислювальну систему до певного класу. З іншого боку, приналежність обчислювальної системи до деякого класу повинна досить виразно і повно характеризувати її. Одна з перших класифікацій була запропонована Флінном ще в кінці 60-х років. На неї до сих пір найбільш часто посилаються в літературі. Вона базується на понятті двох потоків: потоку команд і потоку даних.

У будь-якій обчислювальній системі можна виділити два найважливіших компонента - центральний процесор, що виконує обчислення, і оперативну пам'ять, в якій зберігається виконувана програма, розроблена спільно з робочою даними. У класичній однопроцесорній обчислювальній системі єдиний потік інструкцій, що генерується програмою, обробляє єдиний потік даних.

За класифікацією Флінна, обчислювальні системи цього типу отримали назву SISD (Single Instruction stream - Single Data stream). На багатопроцесорних системах можлива різна організація виконання паралельної програми. SIMD системи (Single Instruction stream - Multiple Data stream) дозволяють організувати виконання на всіх процесорах однієї і тієї ж команди над різними даними. При цьому в класичних SIMD архітектурах виконання кожної команди програми в усіх процесорах здійснюється синхронно під управлінням єдиного блоку інтерпретації команд єдиного примірника програми, що позбавляє від необхідності використання спеціальних засобів синхронізації програм для забезпечення одночасності виконання команд міжпроцесорних комунікацій [24].

Тому SIMD системи часто називають також системами з синхронним паралелізмом. Більш загальним випадком, є MIMD системи (Multiple Instruction stream - Multiple Data stream), в яких кожен процесор може виконувати свій потік інструкцій над окремим потоком даних. Відзначимо, що для програмування багатопроцесорних систем з архітектурою типу MIMD може використовуватися принцип програмування SPMD (Single Program - Multiple Data), реалізований шляхом клонування екземплярів однієї програми в усі процесори системи з архітектурою типу MIMD. При цьому, однак, виконання різних примірників такої програми в різних процесорах здійснюється асинхронно і, тому, виконання міжпроцесорних комунікацій має синхронізуватися з використанням відповідних засобів.

Як правило, багатопроцесорні системи з архітектурою типу MIMD і розвиненим системним програмним забезпеченням крім режиму MIMD (який

без всякого зміни сенсу може бути названий MPMD) підтримують також режим SPMD виконання паралельної програми. Наприклад, обидва режими роботи підтримувала багатопроцесорна обчислювальна система nCUBE 2S. виконання різних примірників такої програми в різних процесорах здійснюється асинхронно і, тому, виконання міжпроцесорних комунікацій має синхронізуватися з використанням відповідних засобів. Як правило, багатопроцесорні системи з архітектурою типу MIMD і розвиненим системним програмним забезпеченням крім режиму MIMD (який без всякого зміни сенсу може бути названий MPMD) підтримують також режим SPMD виконання паралельної програми [24].

Хоча класифікація Флінна аж до теперішнього часу є найбільш згадуваною при характеристиці того чи іншого комп'ютера, однак вона вже погано відображає стан розвитку суперкомп'ютерних технологій. На сьогоднішній день класичні системи SIMD архітектури (ILLIAC IV, ICL DAP і ін.).

Залишилися в минулому, а клас MIMD виявився перевантаженим, оскільки до нього можна віднести будь-яку сучасну багатопроцесорну систему. Теоретично можлива архітектура MISD (Multiple Instruction stream - Single Data stream) так і не отримала жодного реального втілення. Проте, оскільки в літературі по багатопроцесорним системам і паралельного програмування ці поняття до цих пір застосовуються, то в ознайомлювальних цілях вони згадані і в нашому посібнику [24].

РОЗДІЛ 2
АРХІТЕКТУРНІ ОСОБЛИВОСТІ БАГАТОПРОЦЕСОРНИХ СИСТЕМ

2.1. Архітектура NUMA-Q

Конкуренція на ринку високопродуктивних комп'ютерних систем не слабшає. Серед найбільших подій, що сталися за останні роки минулого століття на ринку, слід зазначити, наприклад, придбання в 1996 р корпорацією Silicon Graphics (www.sgi.com) компанії Cray Research, виробника знаменитих векторно-конвеєрних суперкомп'ютерів Cray 1/2 / X- MP, а також серій C90 і T3D, попередників нинішніх Cray T90 / J90 і T3E. (Правда, в минулому році SGI продала торговельну марку Cray і частина пов'язаного з нею бізнесу.) [10].

Іншим прикладом може служити злиття в 1997 р корпорації Hewlett-Packard (www.hp.com) і компанії Convex Computer, в результаті якого в сімейство серверів HP 9000 були включені SMP- і NUMA-системи Convex SPP на базі процесорів PA-RISC, попередники нинішніх машин V-класу. Не менш значущою була угода, укладена в 1998 р Compaq Computer (www.compaq.com) з купівлі корпорації Digital Equipment Corporation (DEC) разом з відомою лінією серверів AlphaServer [16].

У липні 1999 р про своє злиття оголосили найбільші комп'ютерні корпорації IBM (www.ibm.com) і Sequent Computer Systems (www.sequent.com). Загальна сума угоди становила приблизно 810 млн. $ Після вирішення всіх фінансових питань Sequent фактично стала підрозділом IBM. На той час ця корпорація була постачальником масштабованих NUMA-серверів, що включають до 64 процесорів Intel і працюють під управлінням ОС DYNIX / ptx. Дані комп'ютери в основному були призначені для комерційних систем онлайнової обробки транзакцій і підтримки СУБД. По завершенні злиття маркетинг та продажу цих серверів стали здійснюватися через відповідну всесвітню інфраструктуру IBM [10].

Спочатку аналітики критикували IBM за те, що вона додає ще одну версію Unix до своїх продуктів, серед яких вже є ОС AIX. У момент покупки корпорація особливо не поширювалася про свої плани щодо NUMA-Q, повідомивши лише про намір інтегрувати архітектуру, створену в Sequent, в інші свої сервери. Як заявив Роберт Стефенсон (Robert Stephenson), віце-президент IBM і керівник підрозділу IBM Server Group: "NUMA стане визначальною технологією для Unix- і NT-серверів вже на початку XXI століття [16].

Це елегантне рішення поєднує в собі високу масштабованість і легкість управління ". Дійсно, сервери Sequent дозволяють NT і Unix працювати на одній системі: частина процесорів може бути виділена під завдання Windows NT, частина - під процеси Unix, при цьому можливий поділ даних між завданнями різних ОС. Вважається, що з компанією Sequent пов'язаний один з проривів в комп'ютерних технологіях [10].

У 1983 р в Портленді (шт. Орегон) вісімнадцятьма колишніми співробітниками компанії Intel під керівництвом Кейсі Пауелла була організована компанія Sequel. Вона орієнтувалася на створення обчислювальних архітектур для обслуговування інформаційних технологій в найбільш трудомістких сферах роботи з діловою інформацією - оперативної обробці транзакцій, системах підтримки прийняття рішень і ділових комунікаціях. На початку 80-х найбільш важкі завдання вирішувалися на мейнфреймах, а ця компанія поставила собі за мету побудову такої обчислювальної платформи, яка істотно перевершить можливості однопроцесорних систем, але при цьому буде базуватися на найпоширеніших і недорогих процесорах від Intel [10].

Sequel, перейменована в 1987 р в Sequent Computer Systems, реалізувала першу SMP-версію ОС Unix, що володіє незалежною від процесора архітектурою, а в травні того ж року анонсувала першу комп'ютерну систему Symmetry (Unix SMP), масштабовану до 30 процесорів Intel. Перша ОС SMP Unix компанії Sequent називалася DYNIX і була випущена в розвиток BSD 4.2.

У 1990 р Sequent випустила для своїх машин на основі Unix System V нову версію ОС - DYNIX / ptx. На вимогу підвищити надійність систем компанія Sequent відповіла створенням в 1992 р кластерів для Unix. Крім того, вона не без успіху вирішувала одну з головних проблем архітектури SMP - подолання бар'єру масштабованості в 30 процесорів, піднявши за допомогою архітектури NUMA планку майже на порядок.

Як відомо, NUMA (Non-Uniform Memory Access) - це архітектура, яка використовується в багатопроцесорних системах, де час доступу залежить від географічного розташування пам'яті. Процесор може працювати з власної локальної пам'яттю набагато швидше, ніж з нелокальної, яка в свою чергу є локальною для іншого процесора або розділяється між декількома пристроями.

Ключовим поняттям багатопроцесорних архітектур є вузол - обчислювальна система, що складається з одного або декількох процесорів, що має оперативну пам'ять і систему введення-виведення. Вузол характеризується тим, що на ньому працює єдина копія ОС. Зазвичай SMP-вузол містить не менше двох однакових пристроїв, які використовуються рівноправно. Процесори SMP взаємодіють один з одним за допомогою так званої шини з’єднання і використовують загальний пул пам'яті. При збільшенні в сервері числа процесорів зростає і трафік на даній шині. При значному зростанні цього числа ефективна пропускна здатність шини істотно знижується [16].

NUMA, як і SMP, дозволяє отримати об'єднану обчислювальну потужність великої кількості процесорів, кожен з яких звертається до загального пулу пам'яті. Однак в цьому випадку процесори організовані в невеликі групи, вузли, за допомогою яких вони можуть зв'язуватися між собою. Наприклад, 16-процесорний сервер може містити чотири вузли по чотири процесори. Кожен вузол має власний пул пам'яті.

У порівнянні з SMP архітектура NUMA зменшує навантаження на шину, оскільки процесори в вузлах взаємодіють один з одним і зі своєю локальною

оперативною пам'яттю через окремі шини. Крім того, вони можуть звертатися до пулів пам'яті інших вузлів, хоча час доступу залежить від того, наскільки ці вузли віддалені один від одного. Тому таку архітектуру часто називають архітектурою з розподіленою пам'яттю. У NUMA може бути задіяно 256 і навіть 512 процесорів [10].

Кілька процесорів можуть розділяти доступ до однієї і тієї ж пам'яті, забезпечуючи широку смугу пропускання за рахунок використання великої багаторівневої кеш-пам'яті. Оскільки вони працюють з даними, що зберігаються в єдиній пам'яті вузла, в подібній архітектурі обов'язково повинен бути механізм підтримки когерентності даних. Це поняття означає, що в будь-який момент часу для кожного елемента даних у всій пам'яті вузла існує тільки одне його значення, хоча одночасно можуть існувати кілька копій цього елемента, розташовані в різних видах пам'яті і оброблювані різними процесорами. Механізм когерентності повинен стежити за тим, щоб операції з одним і тим же елементом даних виконувалися на різних процесорах послідовно, а застарілі копії віддалялися. В сучасних SMP-архітектурі когерентність реалізується апаратними засобами [10].

Існує два класи протоколів когерентності кеш-пам'яті:

- на основі довідника (directory based). Інформація про стан блоку фізичної пам'яті міститься тільки в одному місці, званому довідником (фізично довідник може бути розподілений по вузлах системи);

- протоколи спостереження (snooping). Кожна кеш-пам'ять, яка містить копію даних деякого блоку фізичної пам'яті, має також відповідну копію службової інформації про її стан. Централізована система записів відсутня. Зазвичай кеш-пам'ять розташована на загальною (що розділяється) шині і все її контролери спостерігають за шиною (переглядають її) і відстежують, чи не містять вони копію відповідного блоку [16].

У багатопроцесорних системах з централізованою спільною пам'яттю протоколи спостереження набули популярності, оскільки для опитування стану кеш-пам'яті вони можуть використовувати заздалегідь існуюче фізичне

з'єднання (шину пам'яті). Кешування на основі каталогу подвоює пропускну здатність шини, але вимагає більш складних апаратних засобів і викликає додаткові затримки при пересилці даних між пам'яттю і обома шинами.

Різновидом архітектури NUMA є кеш-когерентний доступ до неоднорідною пам'яті cc-NUMA (cache coherent NUMA). В цьому випадку всі процесори об'єднані в один вузол, причому перший рівень ієрархії пам'яті утворює їх кеш-пам'ять, а cc-NUMA підтримує когерентність всередині вузла апаратно. У системах cc-NUMA розподілена пам'ять представляється єдиним адресним простором [10].

Чи не відбувається ніякого копіювання сторінок або даних. Немає програмної передачі повідомлень для синхронізації доступу. Є просто єдиний масив пам'яті (хоча фізично складається з окремих частин). Апаратна когерентність кеш-пам'яті означає, що не потрібно ніякого програмного забезпечення для підтримки актуальності безлічі копій даних. Все це виконується на апаратному рівні так само, як в будь-якому SMP-вузлі, з одним екземпляром ОС і безліччю процесорів. У свою чергу NUMA-Q - це по суті реалізація архітектури cc-NUMA компанією Sequent [16].

Елементарним блоком платформи NUMA-Q служить Квод (quad), в якому об'єднані чотири процесори, блок пам'яті, що і шина PCI з сімома слотами. Кілька Квод можуть бути з'єднані для формування більшого одиночного SMP-вузла з апаратно реалізованої кеш-когерентністю. Кеш-когерентне з'єднання, яке встановлюється між шинами, називається IQ-Link. Воно практично прозоро для програм подібно звичайному кеш-пам'яті. У традиційному сенсі пам'ять в кожному Квод не є локальною [10].

Швидше це одна третина адресного простору фізичної пам'яті, що має власний адресний діапазон. Адресна карта розподіляється по пам'яті рівномірно, при цьому кожен Квод містить суміжну частина адресного простору.

Наприклад, якщо потрібну адресу знаходиться за межами діапазону локальної пам'яті Квод, пошук буде поширений на кеш-пам'ять IQ-Link, яка

називається віддаленої. Доступ до неї здійснюється з тією ж швидкістю, що і до локальної пам'яті Квод. Якщо і в кеш-пам'яті IQ-Link дані не знайдені, то для їх отримання відсилається запит на шину IQ-Link. Після того як необхідне значення завантажено з іншого Квод, воно зберігається в віддаленій кеш-пам'яті IQ-Link запитувача Квод [16].

Новий NUMA-Q від Sequent нерівномірний Доступ до пам'яті забезпечує нові рівні продуктивності, доступності та керованості в системах корпоративних класів.

2.2. Архітектура CMP

Зростання залежності від комп'ютерних систем вимагає від процесорів забезпечують вищі та вищі показники. Особливо це стосується полів мікропроцесорних систем, які зазвичай використовують паралельність на всіх рівнях деталізація для поліпшення обчислювальних характеристик. В даний час VLIW, загальноприйняті в мікропроцесорах, - це дві широко розповсюджені архітектури паралелізму на рівні інструкцій [31].

Однак вони досягають і кращого продуктивність ціною збільшення складності логіки управління. Статичні дані, однак, показали, що гранична віддача додаткового обладнання різко зменшувався зі збільшенням складності [30].

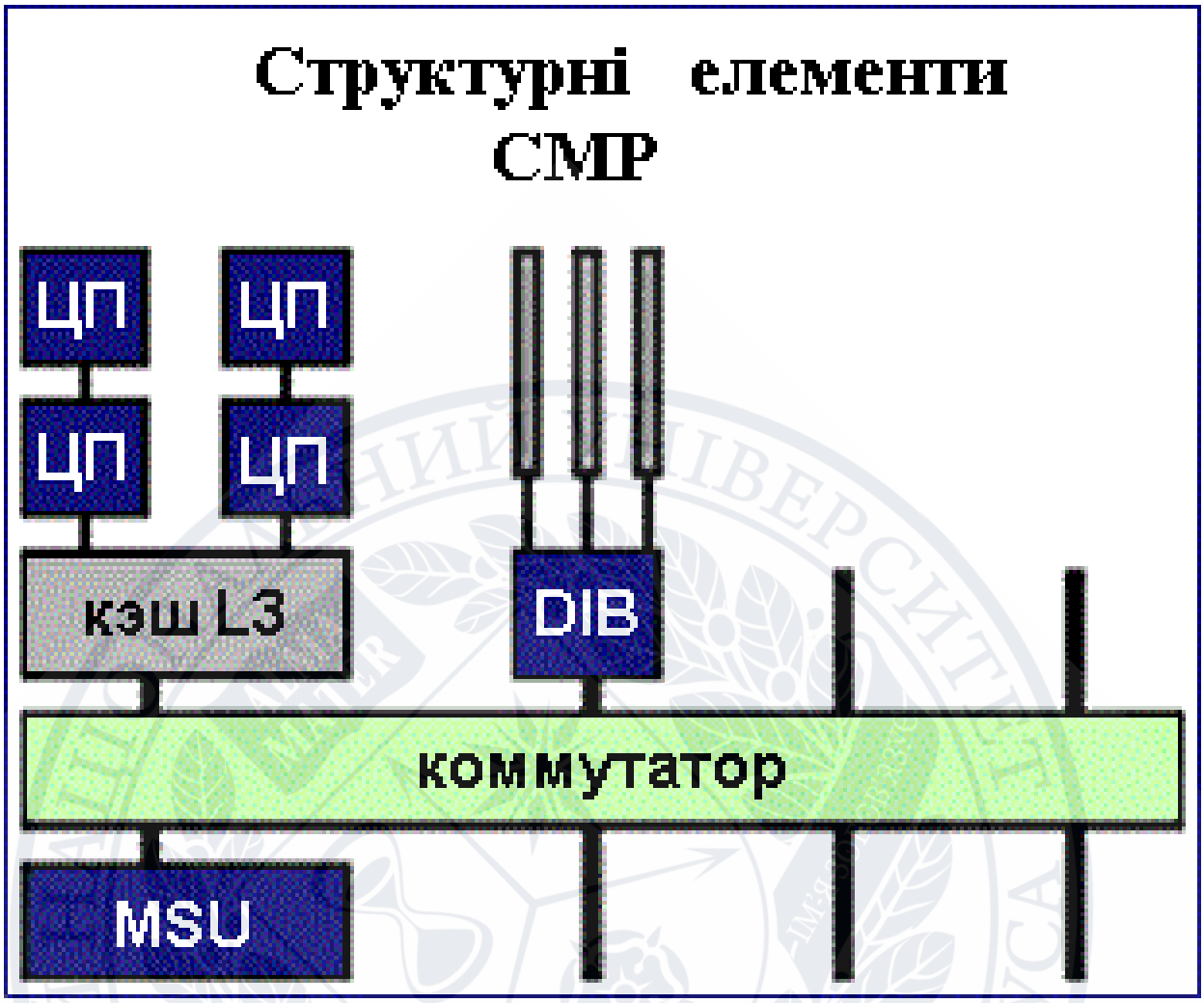
CMP інтегрують декілька процесорів в єдиний чіп і, таким чином, поєднують переваги як мікропроцесорної, так і багатопроцесорної архітектури. З точки зору будівництва, стандартним CMP є простіший у реалізації, ніж інші мікропроцесори [HNO97].

Загальні архітектури, такі як супер скалярна система та VLIW, як правило, набагато важкі для проектування та таким чином, призводить до збільшення часу розробки та перевірки. Навпаки, CMP зазвичай використовують дуже просту логіку управління і навіть повторно

використовують існуючу чітко визначені або комерційно успішні ядра процесора для побудови систем таким чином як зменшити ризик дизайну [31].

Крім того, у міру затримки між мереж між воротами стає все більш значним, мінімізуючи довжину проводів і спрощення проектування критичних шляхів - дві важливі особливості. CMP можуть також значно пом'якшити цей проектний тиск шляхом створення кожного процесора займають відносно невелику площу на великому процесорному чіпі.

З точки зору продуктивності обчислень, CMP також пропонують більше переваг. Оскільки затримки між процесорного зв'язку менші, а пропускна здатність вища, CMP набагато менш чутливі до компонування даних та управління комунікаціями. Як результат, вони можуть прискорити передачу даних транзакцій між процесорами, що робить їх швидшими, ніж звичайні мультипроцесори для запуску паралельних програм, особливо коли потоки між процесори часто спілкуються [30].



Всього викладеного, мабуть, було б недостатньо, щоб можна було говорити про СМР як про нову архітектурну парадигмі. Однак розробники Unisys заклали в СМР і інші унікальні особливості, що забезпечують можливості статичного і динамічного розбиття SMP-сервера, що призводять до перетворення в кластер всій SMP-системи, в свою чергу побудований з SMP-серверів з числом процесорів, кратним 4 . Мало того, і сам такий кластер є унікальним явищем в комп'ютерній індустрії: взаємодія його вузлів здійснюється ... через загальне поле оперативної пам'яті! [30].

Зображений на рис. 2 SMP-сервер в архітектурі СМР може бути сконфігурований в вісім 4-процесорних SMP-систем, або в чотири 8-процесорні SMP-системи, або в іншу комбінацію SMP-систем з числом процесорів, кратним 4. Подібна реконфігурація комп'ютера з виділенням

окремих розділів давно відомо в світі мейнфреймів. Серед Unix-серверів така доменна організація доступна, наприклад, в серверах Sun Ultra Enterprise 10000. Найбільш «слабким» аналогом цього в SMP-серверах є виділення «процесорних наборів», доступне, зокрема, в версіях Unix для HP / Convex SPP і SGI Challenge / Power Challenge [31].

На відміну від цих коштів виділення розділів-доменів в CMP передбачає можливість роботи в кожному з них своєї операційної системи. У CMP-серверах це можуть бути взагалі різні операційні системи - скажімо, NT або SCO UnixWare чи інша версія Unix. Особливістю CMP є те, що фізично вся оперативна пам'ять цих розділів є спільною. При цьому в CMP можливо три типу поділу поля оперативної пам'яті між розділами: а) кожна операційна система використовує виключно власну пам'ять; б) кожна ОС має свою пам'ять і утворюється ще одна загальна для різних ОС область пам'яті; в) кожна ОС має свою пам'ять і утворюється кілька областей пам'яті, що розділяються деякими ОС [30].

2.3. Архітектура NUMAflex

Так, ще в 1988 р почалася робота над першої NUMA-машиною - частиною проекту DASH спільно зі Стенфордським університетом. Взагалі кажучи, однією з найбільш досконалих і гнучких при побудові серверів по праву вважається архітектура ccNUMA (cache coherence Non-Uniform Memory Access) з неоднорідним доступом до пам'яті і підтримкою когерентності кеш-пам'яті всіх процесорів, в тому числі що належать різним вузлам. Система ccNUMA зазвичай складається з набору вузлів, кожен з яких має власні процесори, локальну оперативну пам'ять і засоби введення-виведення [9].

Архітектура ccNUMA зовсім не обмежується масивно-паралельними обчислювальними системами (MPP, Massive Parallel Processing). Вона активно застосовується і при побудові систем з невеликим числом процесорів в якості

альтернативи симетричною багатопроцесорної архітектурі (SMP, Symmetrical Multi Processing) [8].

На відміну від класичної архітектури NUMA, при використанні кеш-когерентного доступу до неоднорідною пам'яті ccNUMA все процесори об'єднані в один вузол, причому перший рівень ієрархії пам'яті утворює їх кеш-пам'ять, а ccNUMA підтримує когерентність всередині вузла апаратно. У системах ccNUMA розподілена пам'ять виглядає як єдиний адресний простір. Чи не відбувається ніякого копіювання сторінок або даних, немає програмної передачі повідомлень для синхронізації доступу. Є просто єдиний масив пам'яті (фізично, втім, що складається з окремих частин). Апаратна когерентність кеш-пам'яті означає, що не потрібно ніякого ПО для підтримки актуальності безлічі копій даних. Все це виконується на апаратному рівні так само, як в будь-якому SMP-вузлі, - з одним екземпляром ОС і безліччю процесорів [9].

За останні кілька років корпорація SGI кардинальним чином доповнила і модернізувала випускаються сімейства комп'ютерів. Зокрема, її зусилля з розробки архітектури ccNUMA втілилися в сервери сімейства Origin 3x00. Ці комп'ютери на базі технології NUMAflex мають архітектуру пам'яті NUMA 3 (це вже третє покоління подібної архітектури). Завдяки модульній конструкції нове сімейство серверів SGI може легко масштабуватися в розрахунку на конкретну задачу або додаток з використанням набору базових модулів. При цьому обчислювальну потужність сервера можна нарощувати від 2 до 512 процесорів, а продуктивність підсистеми вводу-виводу - від 11,2 до 716 Гбайт / с. [22]

Нагадує за своїм принципом дитячий конструктор, модульна архітектура NUMAflex дозволяє відмовитися від звичної практики - купувати високопродуктивні сервери з явним запасом потужності. Тепер користувачі мають можливість розширювати і модернізувати в своїх системах тільки необхідні елементи або додавати нові технології в міру їх появи, в той час як раніше їм доводилося купувати дорогі системи, можливості яких або набагато

перевершували вимоги власників, або швидко старіли, так що їх потрібно повністю замінювати. [9].

Завдяки NUMAflex кожен модуль системи наділяється конкретною функцією і може бути пов'язаний (за допомогою високошвидкісного з'єднання) з багатьма іншими модулями різних типів, створюючи єдину конфігурацію. Залежно від конфігурації одні і ті ж модулі можна використовувати для подальшої модернізації і підвищення продуктивності.

Не дарма в назві архітектури фігурує слово flex (скорочення від англійського flexibility - гнучкість). [22].

NUMAflex дійсно відрізняється особливою гнучкістю в побудові різних конфігурацій системи і її зміні "на льоту", в процесі функціонування. У NUMAflex реалізована можливість розбивати всю ccNUMA-систему на розділи (партіції, або домени), які теж являють собою ccNUMA- або SMP-комп'ютери. Розподіл на розділи дозволяє перетворювати ccNUMA-систему в кластерну структуру. Вузлами цього кластера можуть бути знову-таки ccNUMA-сервер. Нова технологія використовується не тільки в високопродуктивних серверах серії Origin 3x00, але і в системах візуалізації Onyx 3000 [8].

Сімейство Origin 3000 складається з моделей Origin 3200, 3400 і 3800, причому останні можуть містити від 2 до 512 процесорів і до 1 Тбайт оперативної пам'яті. Ці багатопроцесорні системи працюють під управлінням ОС IRIX 6.5, першої в світі 64-розрядної ОС UNIX, що забезпечує високу продуктивність і суперкомп'ютерні можливості для вирішення практичних завдань. Крім того, нові моделі повністю сумісні з додатками для систем попереднього покоління Origin 2000 і Onyx2. [9].

Всі системи з архітектурою NUMAflex в даний час будуються на RISC-процесорах MIPS R12000A 400 МГц (R14000 500 МГц). Як було заявлено, найближчим часом з'являться обчислювальні модулі на базі 64-розрядних процесорів Intel Itanium. Таким чином, користувач може вибирати тип

використовуваних процесорів або модернізувати всю систему, замінюючи тільки обчислювальні модулі [22].

Архітектура MIPS (Microprocessor without Interlocked Pipeline Stages), розроблена фахівцями SGI / MIPS Technologies, була однією з перших RISC-архітектур, які отримали визнання галузі. В даний час вона ліцензована багатьма найбільшими виробниками напівпровідникових пристроїв, серед яких можна відзначити Broadcom, IDT, LSI Logic, NEC, NKK, Philips і Toshiba. Розроблено спеціальний стандарт, що забезпечує переносимість бінарних додатків між різними MIPS-платформами, що працюють під управлінням ОС UNIX. У MIPS Technologies, мабуть, самий тривалий досвід роботи з 64-розрядними архітектурою, а що випускаються нею мікропроцесори підтримують створення SMP-архітектур з пам'яттю, що з десятків мікропроцесорів [9].

Користувачів привертає в цій архітектурі орієнтація на використання мультимедійних даних і високоякісних засобів візуалізації зображень. Досить часто мікропроцесори цього сімейства застосовуються для створення вбудованих контролерів. Власне абревіатура MIPS означає "мікропроцесор без затримок очікування конвеєра". У самій назві підкреслюється важлива властивість цієї RISC-архітектури - збалансованість тракту вибірки команд з функціональними вузлами процесор [8].

Сімейство кристалів R1x000 можна охарактеризувати як суперскалярні мікропроцесори з позачерговим спекулятивним виконанням команд, в яких використовується техніка перейменування регістрів і динамічний пророцтво переходів. Зокрема, R10000 може виконувати чотири команди за такт, причому вибірка і декодування чотирьох команд також відбувається за один такт. Ємність кеш-пам'яті першого рівня в процесорі становить 32 Кбайт для команд і стільки ж для даних. Обидва пристрої відносяться до двоканальним набірний-асоціативним. Ємність кеш-пам'яті другого рівня може варіюватися в діапазоні від 0,5 до 16 Мбайт. У процесорі передбачений повністю

асоціативний буфер швидкої переадресації TLB (ємністю 64 рядка) і додатковий буфер TLB для команд (ємністю 8 рядків) [9].

При розробці процесора R10000 велику увагу було приділено ефективної ієрархії пам'яті. Зокрема, в цьому кристалі забезпечується раннє виявлення промахів кеш-пам'яті з виконанням іншої корисної роботи. Так, кеш-пам'ять підтримує одночасну вибірку і виконання команд завантаження і запису даних в пам'ять, а також операції перезавантаження рядків кеш-пам'яті [22].

Робота конвеєрів кеш-пам'яті даних тісно координована. При виявленні промаху при зверненні до кеш-пам'яті даних її робота не блокується, отже, вона може продовжувати обслуговувати такі запити.

Архітектура R1x000 включає 64 фізичних регістра. У процесорі є п'ять повністю незалежних виконавчих пристроїв: два цілочисельних, два речової арифметики, а також пристрій завантаження / запису [8].

Час виконання всіх цілочисельних операцій, за винятком операцій множення і ділення, становить один такт. Цілочисельні операції множення виробляють твір з подвійною точністю. Для операцій з одинарної точністю поширення знака результату до 64 розрядів відбувається перш, ніж він буде поміщений в регістри. Час виконання операцій з подвійною точністю приблизно в два рази більше, ніж операцій з одинарної точністю [9].

Крім основних пристроїв речової арифметики існують два додаткових, які обробляють довгі операції ділення і обчислення квадратного кореня. Пристрій завантаження / запису виконує команди завантаження, записи, попередньої вибірки, а також інструкції для роботи з кеш-пам'яттю. Зовнішня кеш-пам'ять другого рівня управляється за допомогою внутрішнього контролера, який має для цієї мети спеціальний порт. За магістралі шириною в 128 розрядів пересилаються дані на внутрішній тактовій частоті 200 МГц. Системний 64-розрядний інтерфейс R10000 використовується в якості шлюзу між самим процесором, пов'язаної з ним кеш-пам'яті другого рівня і рештою

системи. Мікросхема R10000 допускає два способи організації багатопроцесорної системи [9].

Архітектура наступного за R10000 кристала, R12000, відрізняється дуже незначно. Так, в чотири рази зросла ємність таблиці передбачення переходів, з'явилася кеш-пам'ять адрес переходів, в адресному черзі були введені окремі конвеєри для перевірки тегів і для розрахунку адрес, ємність активного списку черги команд збільшилася з 32 до 48 рядків, вдвічі зросла ємність таблиці використання даних в контролері кеш-пам'яті другого рівня.

Крім того, поліпшена розщеплена обробка транзакцій на системній шині для зменшення часу зайнятості кеш-пам'яті другого рівня. Варто відзначити, що в R12000 все команди завантаження регістрів, записи в пам'ять і роботи з кеш-пам'яттю надсилаються спочатку в чергу цілочисельних команд, а звідти видаються в виконавчий пристрій. Після цього вони видаляються з черги цілочисельних команд і поміщаються в чергу команд завантаження регістрів і записи в пам'ять. Така схема, за твердженням розробників, дозволила навіть спростити конструкцію R12000 в порівнянні з R10000 [8].

У R12000 була також кілька доопрацьована архітектура кеш-пам'яті. Виконаний з дотриманням проектних норм 0,25 мкм кристал R12000 містить 6,9 млн. Транзисторів. На тактовій частоті 300 МГц він показує продуктивність 27,5 SPECfp95 [22].

Версія процесора R12000A з'явилася в липні минулого року і повністю заснована на архітектурі R12000. Змінилася тільки технологія виготовлення. Завдяки посиленню проектних норм до 0,18 мкм тактові частоти вдалося збільшити до 360 і навіть 400 МГц. Варто відзначити, що короткі конвеєри, закладені в архітектурі, ускладнюють нарощування тактових частот. Так, при ємності кеш-пам'яті другого рівня 8 Мбайт продуктивність кристала на частоті 400 МГц збільшилася до 43,5 SPECfp95.

За наявними даними (The IRIX 6.5.X / MIPS Roadmap, Release 6.5.10), в SGI збираються продовжувати розробки цієї лінії процесорів, принаймні до

2005 р Слідом за R12000A в цьому році має з'явитися кристал R14000 з тактовою частотою 500 МГц . У ньому буде використовуватися кеш-пам'яті другого рівня типу DDR SDRAM, що працює на основній частоті процесора. Кристал буде виготовлятися з дотриманням проектних норм 0,13 мкм і з використанням мідних з'єднань [22].

Рівень напруги живлення буде знижений до 1,5 В. Ємність кеш-пам'яті другого рівня в системах, побудованих на базі R14000, також може становити 8 Мбайт. Модель R14000A буде відрізнятися від базової тільки новим технологічним процесом і подальшим посиленням проектних норм. На 2002 р намічений випуск R16000 з тактовою частотою 600 МГц. У цьому процесорі кеш-пам'яті другого рівня буде розміщена на самому кристалі, де для кеш-пам'яті третього рівня будуть передбачені теги [9].

Кардинальні зміни в архітектурі MIPS очікуються в 2003 році з появою моделі R18000. У цьому процесорі, вперше після R8000, стануть видаватися чотири 64-розрядних результату з плаваючою комою за такт. В архітектуру будуть додані додаткові пристрої завантаження / запису і речової арифметики. На тактовій частоті 800 МГц R18000 повинен мати пікову продуктивність 3,2 GFLOPS. Тактова частота для процесорів MIPS досягне заповітної риси 1 ГГц не раніше 2005 - тоді, коли має з'явитися кристал R20000. [8].

До числа основних переваг архітектури NUMAflex слід віднести відмову від використання великих системних плат, характерних для систем з великим числом процесорів. Це не тільки здешевлює конструкцію, але і підвищує загальну надійність, оскільки усуває системну шину як загальну точку збою для відносно великого числа процесорів. Архітектура NUMAflex будується на базі модулів семи різних типів, які називаються "цеглою" (brick): C-brick - процесорний модуль, I-brick - модуль базового введення-виведення, R-brick - комутаційний модуль, P-brick - модуль розширення PCI , D-brick - дисковий модуль, X-brick - розширення XIO для високопродуктивного введення-виведення і G-brick - графічний модуль (InfiniteReality). «Цеглина» забезпечена власними джерелами живлення, що підвищує відмовостійкість

системи в цілому. Вони поміщаються в стандартну стійку, причому для невеликих систем рекомендується конструктив 17U [22].

Один з найважливіших компонентів архітектури NUMA 3 - спеціалізована мікросхема Bedrock. Цей матричний комутатор з 8 входами і 6 виходами діє як контролер між процесорами і локальної та дистанційної пам'яттю. Пікова продуктивність кристала досягає 3,2 Гбайт / с. Крім того, цей чіп надає процесору канал в систему введення-виведення. Відзначимо також, що 8-портовий високопродуктивний матричний комутатор знаходиться в вузлі маршрутизатора [9].

Основний модуль - C-цегла (висота 3U) містить два або чотири процесори, а також від 512 Мбайт до 8 Гбайт пам'яті (типу ECC SDRAM або DDR), розділених на чотири банки. Перевага Origin 3x00 перед Origin 2000 - це і можливість використання різних моделей R1x000 (з різною тактовою частотою) в різних C-цеглинах одного сервера. Є один інтерфейс типу NUMAlink3 (1,6 Гбайт / с в кожному напрямі, 3,2 Гбайт / с - повний дуплекс) для підключення до іншого C-цеглі або R-цеглі. Один канал зв'язку Xtown2 (1,2 Гбайт / с на кожен напрямок) дає змогу підключити до цеглини введення-виведення [8].

I-цегла (висота 2U) забезпечує основний рівень введення-виведення для систем сімейства SGI Origin 3x00. До складу I-цегли входять системний диск 18 Гбайт з інтерфейсом Fibre Channel, привід CD-ROM і п'ять слотів PCI з "гарячої" заміною. I-цегла також забезпечує доступ до локальної мережі через порт 10 / 100Base-T. Для підключення периферійних пристроїв служать один інтерфейс IEEE 1394 і два канали USB. I-цегла також забезпечений двома Xtown2-портами, які з'єднуються з аналогічними портами на C-цеглинах.

P-цегла (висота 4U) забезпечує додаткове PCI-розширення. Він підтримує до 12 64-розрядних слотів PCI з "гарячої" заміною, розподілених по шести незалежним шинам, кожна з яких підтримує два слоти, що конфігуруються на 33 або 66 МГц. P-цегла має два Xtown2-порту.

У X-цеглі є чотири слота половинної висоти XIO (високошвидкісний канал введення-виведення), повністю сумісних з аналогічними слотами в системах SGI Origin 2000. Це дозволяє застосовувати вже існуючі плати, наприклад, для HIPPI, GSN, VME і цифрового відео. X-цегла має два Xtown2-порту [9].

R-блок (висота 2U) як структурний стандартний блок системи замінює системну шину. По суті, це швидкодіючий комутатор, який з'єднує процесори і пам'ять, а також дозволяє обслуговувати і модернізувати кожної компонент системи індивідуально. З восьми портів NUMAlink 3 на R-цеглі чотири порти з'єднують з аналогічними на C-цеглинах. Таким чином, організовується 16-процесорна система, зав'язана на один R-цегла. Решта чотири порти служать для зв'язку з іншими вузлами маршрутизаторів, що дозволяє нарощувати систему до 128 вузлів або 512 процесорів. Існує три різновиди R-цегли: 6-портовий для 32-процесорних систем (Origin 3400), 8-портовий для 128-процесорних систем (Origin 3800) і мета-маршрутизатор для 512-процесорних систем [8].

D-цегла (висоти 4U) підтримує дискові масиви типу JBOD або RAID з інтерфейсом Fibre Channel з дванадцяти дисководів ємністю 18, 36 або 73 Гбайт.

G-цеглини відповідають за високопродуктивну графіку, перетворюючи сервери Origin 3x00 в графічні суперкомп'ютери Onyx 3x00. Кожна стійка може містити від одного до двох G-цегл. Кожен конвеєр G-цегли з'єднується з каналом Xtown2 I-, P- або X-цегли через кабель DNET [22.

У блоці живлення розміщується від трьох до шести розподілених джерел живлення з "гарячої" заміною, що забезпечують напругу 48 В для C-, I-, P-, X- і R-цегл. Джерела завжди встановлюються в надлишкової конфігурації N + 1, щоб відмова будь-якого з них не торкнувся роботу сервера в цілому [9].

Своєю високою гнучкістю, що дозволяє зберігати інвестиції користувача при модернізації, архітектура NUMAflex зобов'язана

використанню замінних модулів. Купуються тільки ті цеглини, які дійсно необхідні, і вже з них складається комп'ютер потрібної конфігурації.

Відзначимо також, що системи Origin 3x00 з 512 процесорами можна розділити на 32 розділу. Мінімальний домен повинен містити один обчислювальний модуль, що включає два або чотири процесори. Кожна розбиття має власні засоби введення-виведення, свій IP-адресу і працює під управлінням власної версії ОС. Для забезпечення зв'язку між розділами-вузлами кластера використовується NUMAlink 3 - стандартне з'єднання з набагато більш високою пропускною здатністю і більш низькими затримками, ніж традиційні канали зв'язку між вузлами кластерів [22].

При розбитті NUMAflex-сервера на розділи з переходом до кластерної структурі можна побудувати систему високої доступності, причому від звичайного кластера цього типу вона буде відрізнятися високою продуктивністю каналів, що з'єднують вузли. Використання доменів забезпечує ефективне управління робочим навантаженням, т. Е. Поділ ресурсів між колективами користувачів або групами завдань. Крім того, допускається модернізація ОС при одночасному продовженні промислової експлуатації поточної версії. Підвищення відмовостійкості відбувається за рахунок усунення спільних точок збою і освіти кластерів високої доступності [8].

Буквально відразу після виходу серверів з новою архітектурою SGI отримала замовлення від ряду великих комерційних і державних установ. Першими користувачами сімейства SGI 3000 стали збройні сили США, NASA, технічний університет в Дрездені і корпорація Sony Computer Entertainment [9].

Військово-морський центр метеорології та океанографії вирішив придбати 128 і 512-процесорні системи SGI Origin 3800. Центр здійснює перехід з супер-комп'ютера Cray C90 на технологію архітектури NUMAflex. NASA також замовила дві 512-процесорні системи SGI Origin 3800, об'єднані

в єдиний обчислювальний комплекс із загальною пам'яттю 1 Тбайт. В рамках програми модернізації високопродуктивних комп'ютерних ресурсів армійське лабораторія міністерства оборони США придбала ще два сервера SGI Origin 3000 - один з 512 процесорами, інший - з 256. Крім того, в інженерно-дослідному центрі Пентагону встановлений 512-процессорний комп'ютер SGI Origin 3800[8].

Компанія Piranha, що спеціалізується на рішеннях стиснення даних, оголосила про те, що буде використовувати сервери серії SGI Origin 3000 в якості платформи розробки для своєї багатопроцерної програмної системи шифрування, яка дозволяє обробляти зображення в реальному часі, а також реалізувати зарекомендував себе в галузі алгоритм стиснення відео [9].

Перший в світі 1024-процесорний сервер SGI Origin 3000 був встановлений в Національному суперкомпьютерному центрі Нідерландів. Він допоможе голландському науковому співтовариству вирішувати найскладніші наукові, технічні та медичні проблеми.

Супер комп'ютер в першу чергу передбачається використовувати в дослідженнях клімату; в областях медицини, які вимагають інтенсивних обчислень; в управлінні водними ресурсами і розрахунках якості води; в розрахунках динаміки рідини і моделювання турбулентних потоків; в обчислювальної хімії, в тому числі для розробки нових лікарських препаратів [22].

З характеристик сервера з 1024 процесорами MIPS варто відзначити пікову продуктивність, яка перевищує трильйон операцій в секунду. Ємність дискової пам'яті становить 10 Тбайт, а зовнішньої (StorageTek) - 100 Тбайт.

На думку одного з директорів IDC, великий потенціал SGI Origin 3x00 у вирішенні нових завдань, пов'язаних з Інтернетом, так як системи володіють необхідною для цього гнучкістю, масштабованість і модульність[8].

2.4. Архітектура FAME

Складність завдань, що стоять сьогодні перед великими комерційними, державними та науково-дослідними організаціями, постійно зростає.

Відповідно до цих завдань від сучасних великих серверів потрібно достатня потужність, що дозволяє отримувати інформацію з великих масивів даних складної структури (що включають мультимедіа і неструктуровані дані), виконувати великомасштабні розрахунки, підтримувати дуже великі каталоги з інформацією про сотні тисяч користувачів, які можуть звертатися до онлайнових сервісів через Інтернет [17].

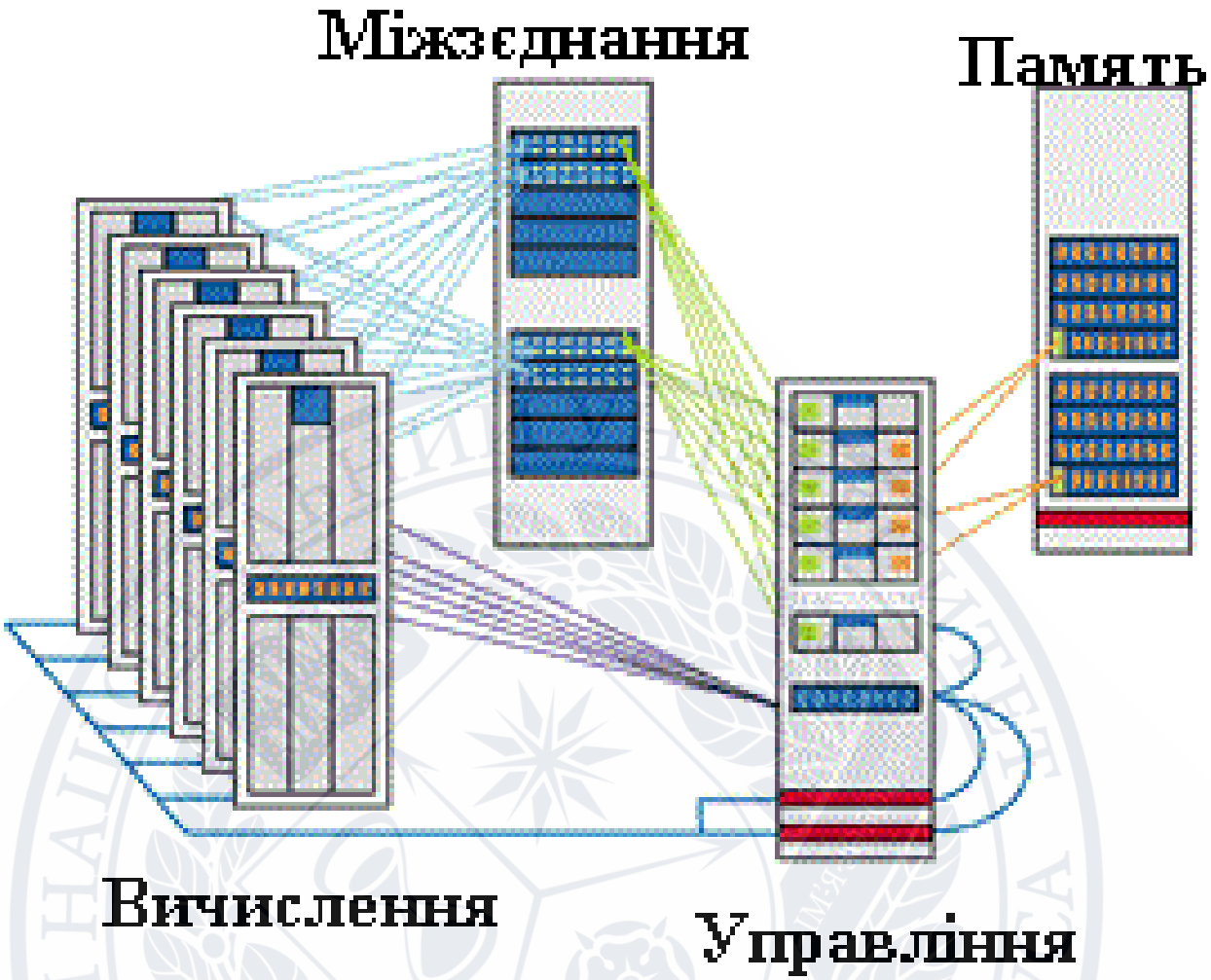
Сервери також повинні справлятися з непередбачуваними стрибками навантаження, пов'язаними з обробкою транзакцій, працювати цілодобово (24 години в день, 365 днів у році), інтегруватися в повністю гетерогенную середу.

Крім чисто технологічних характеристик, не слід забувати і про техніко-економічних показниках. Тут чималу роль відіграють такі чинники:

- співвідношення ціна / продуктивність;
- можливість динамічно адаптувати апаратні ресурси, що виділяються кожному з паралельно працюючих на одній системі додатків в залежності від їх завантаження (причому додатки можуть працювати і під управлінням різних операційних систем);
- зручна система адміністрування для моніторингу операцій і оптимізації ресурсів, які виділяються кожній завданню;
- доступність великої портфеля додатків.

Оскільки архітектура FAME заснована на стандартних компонентах, що виготовляються Intel в великих обсягах, це значно покращує співвідношення ціна / продуктивність.

На її основі розроблений і інтегрований ряд функцій високої продуктивності для побудови систем SMP, що містять від 8 до 32 процесорів. Використання стандартних компонентів гарантує і доступність безлічі додатків [17].



Оскільки сервери класу high-end, як правило, інтегровані в саму стратегічно важливу середу підприємства, вони повинні працювати безперервно, без простоїв. В архітектурі FAME це гарантується не тільки резервуванням компонентів, але і можливістю ізолювати і замінити дефектний елемент, не перериваючи функціонування системи.

Завдяки функції динамічних розділів архітектура FAME забезпечує консолідацію на одному сервері декількох додатків, що працюють під управлінням різних ОС - Windows, Linux, GCOS 7 і GCOS 8. В умовах непередбачуваних стрибків навантаження така можливість дозволяє системі динамічно адаптуватися до змін навантаження без зупинки роботи [17].

РОЗДІЛ 3
МЕТОДИ ЗАХИСТУ БАГАТОПРОЦЕСОРНИХ СИСТЕМ ВІД КІБЕРЗАГРОЗ

3.1. Основні види загроз

На сучасному етапі все більшу роль в подальшому розвитку інформаційних ресурсів грають паралельні обчислювальні системи і обчислення. Подібні системи знаходять застосування в сфері економічних, технологічних та інших процесів [1, с. 59].

У зв'язку з їх розвитком, впровадженням та вдосконаленням широкого поширення набули методи завдання шкоди таким ресурсам. Найбільший інтерес викликають проблеми дослідження методів і засобів захисту інформації в паралельних обчислювальних процесах.

В даний час подібні дослідження не набули належного розвитку. Вивчення і розробка подібної проблематики надасть можливості для подальшого розвитку нових і вже існуючих методів захисту інформації. Використання існуючих алгоритмів їх доопрацювання зміни оптимізація, а також подальша програмна і апаратна реалізація. Таким чином, однією з основних проблем використання багатопроцесорної і паралельної обчислювальної системи є реалізація методів захисту інформації.

Проблема реалізації методів захисту має два аспекти:

- розробка засобів які реалізують криптографічні алгоритми,
- методику використання цих ресурсів.

Запропоновані далі криптографічні методи можуть бути реалізована або програмно, або апаратно методом процедур.

Види інформаційних загроз можна розділити на дві великі групи:

1) відмови та порушення працездатності програмних і технічних засобів;

2) навмисні загрози, які заздалегідь плануються зловмисниками для завдання шкоди.

Виділяють наступні основні групи причин збоїв і відмов у роботі комп'ютерних систем:

- порушення фізичної і логічної цілісності структур даних, які зберігаються в оперативній і зовнішньої пам'яті, що виникають внаслідок старіння або передчасного зносу їх носіїв;

- порушення, які виникають в роботі апаратних засобів через їх старіння або передчасного зносу;

- порушення фізичної і логічної цілісності структур даних, які зберігаються в оперативній і зовнішньої пам'яті, що виникають внаслідок некоректного використання комп'ютерних ресурсів;

- порушення, які виникають в роботі апаратних засобів через неправильне використання або пошкодження, в тому числі через неправильне використання програмних засобів;

- не усунуті помилки в програмних засобах, які не виявлені в процесі налагодження і випробувань, а також що залишилися в апаратних засобах після їх розробки.

3.2. Найчастіші загрози

Щоб провести якісний аналіз вразливостей інформаційної структури, необхідно розрізняти види загроз, які можуть виникнути в системі конкретної організації. Такі загрози поділяються на окремі класи.

0 клас. Зловмисник має прямий фізичний доступ до апаратного забезпечення. При цьому стають можливими наступні загрози:

- установка засобів логування та перехоплення інформації з пристроїв вводу та виводу
- зняття копій з носіїв інформації ІС
- установка шкідливого програмного забезпечення всередину операційної системи ІС

- паравіртуалізація операційних систем та логування змісту оперативної пам’яті.

1 клас. Потенційне джерело загрози, який може знаходитися:

- безпосередньо в інформаційній системі (ІС)
- в межах видимості ІС (наприклад, пристрої для несанкціонованого звукозапису)
- поза зоною видимості ІС (перехоплення даних в процесі їх відправки куди-небудь)

2 клас. Вплив на ІС, яке може нести:

- активну загрозу (троян, вірус)
- пасивну загрозу (копіювання конфіденційної інформації зловмисником)

3 клас. Метод забезпечення опосердкованого доступу, який може бути реалізований:

- безпосередньо (крадіжка паролів)
- за допомогою нестандартних каналів зв'язку (наприклад, уразливості операційної системи).

Головні цілі атаки на ІТ-інфраструктуру компанії:

- отримання контролю над цінними ресурсами та даними
- організація несанкціонованого доступу до корпоративної мережі
- обмеження діяльності підприємства в певній галузі

Другий метод найчастіше реалізується на замовлення недобросовісних компаній-конкурентів або політичними діячами.

Що конкретно може нести загрозу інформаційній безпеці будь-якого підприємства:

- шкідливе програмне забезпечення
- шахраї-хакери
- інсайдери-працівники, що діють зі злими намірами чи через необережність
- природні явища

Реалізувати загрозу можна декількома методами. Наприклад, організувати перехоплення даних, залишити програмну або апаратну «закладку» або порушити роботу локальних бездротових корпоративних мереж, організувати для інсайдерів доступ до інфраструктури компанії.

3.3. Методи боротьби

Обчислювальну мережу можна вважати безпечною в сенсі обробки інформації, якщо в ній передбачена централізована система керованих і взаємопов'язаних перешкод, які перекривають з гарантованою міцністю заданий відповідно до моделі потенційного порушника кількість можливих каналів несанкціонованого доступу і загроз, спрямованих на втрату або модифікацію інформації, а також несанкціоноване ознайомлення з нею сторонніх осіб.

При проектуванні захисту паралельних систем, необхідно провести аналіз місць зберігання інформації, інтерфейсу керування даними, і каналами передачі (локальними мережами або іншими подібними засобами обміну).

Причому основний аспект захисту повинен бути в першу чергу спрямований на захист інтерфейсу управління даними в другу чергу, на захист інтерфейсу обміну даними, і в подальшому на місця зберігання інформації, оскільки вони визначаються як найбільш вразливі місця для несанкціонованого доступу в паралельні системи їх своєчасний захист дозволить забезпечити, надійність і стійкість системи [27].

Розглянемо методи боротьби із різними типами загроз згідно їх класів.

Захист від загроз 0-го класу організовується за допомогою фізичної охорони елементів ІС та систем контролю фізичного доступу. До елементів останньої відносять:

- системи ідентифікації особистості (за допомогою карток з чіпами NFC, біометричних сенсорів, тощо)
- системи відеоспостереження

- пропускні системи (турнікети та двері з електронними замками)

1 клас. Потенційне джерело загрози, який може знаходитися або всередині ІС або ж в зоні її видимості. Для захисту від цього типу загроз застосовуються наступні методи:

- Організація ізоляції інформації всередині самої ІС.
- Фізичні ревізії апаратного забезпечення на наявність нерегламентованих технічних засобів.
- Автоматизована централізована система інвентаризації апаратного забезпечення, яка б встановлювала наявність нерегламентованих технічних засобів, підключених до системної шини обчислювальних машин ІС
- Для захисту від перехоплення даних в процесі їх відправки куди-небудь необхідно захиститися від двох основних методів атаки всередині мережі. Перший метод атаки – network spoofing, він же підміна службових даних всередині мережі. За допомогою цього типу атак можна, наприклад, підмінити основний мережевий шлюз обчислювальної машини або ж підмінити основний сервер трансляції доменних імен у мережеві адреси. Боротьба з цим типом атак покладається на проміжне мережеве обладнання. Проміжне мережеве обладнання або відслідковує та блокує нелегітимні пакунки інформації, або ж дозволяє відправляти службові дані лише верифікованим пристроям. Інший тип атаки – man in the middle, він же перехоплення інформації без зміни апаратної чи програмної конфігурації кінцевих чи проміжних мережевих пристроїв. При цьому частково змінюється ефективна топологія мережі на першому чи другому рівні мережевої моделі OSI. Для захисту від цього типу атак використовуються протоколи шифрування даних для передачі по мережі.

2 клас. Вплив на ІС, яке може нести активні та пасивні загрози

- Для захисту від цього класу застосовуються такі активні системи захисту, як антивіруси та брандмауери, як локальні, так і мережеві.

- Також застосовуються пасивні системи захисту, такі як системи верифікації програмного забезпечення та системи ізоляції програмного забезпечення

3 клас. Метод опосередкованого доступу. Для захисту від цього типу атак застосовують наступні підходи:

- Політика ротація паролів, ключів доступу та ключів шифрування.
- Своєчасне оновлення апаратного та програмного забезпечення компонентів ІС.
- Застосування ізоляції компонентів ІС на етапі мережевого зв’язку, на рівні операційної системи та ізоляції програмного забезпечення всередині операційної системи.

РОЗДІЛ 4
ІЗОЛЯЦІЯ ПРОМНОГО ЗАБЕЗПЕЧЕННЯ ТА ДАНИХ ВСЕРЕДИНІ ОПЕРАЦІЙНОЇ СИСТЕМИ

4.1. Вимоги до системи ізоляції програмного забезпечення та даних

Висунемо вимоги до нашої системи забезпечення безпеки:

- Має мати мінімальну кількість точок входу. В ідеальному випадку це має бути одна папка або директорія на несучій машині, та чітко визначений список процесів. Це необхідно для подальшого посилення безпеки системи ізоляції сторонніми засобами.
- Має підтримуватись різними операційними системами.
- Бути розширюваною, мати API або відкритий код.
- Дозволяти ізолювати виконуваний програмний код.
- Дозволяє ізолювати файли застосунку на диску.
- Має мати змогу підключення до централізованої системи керування.

Розглянемо спочатку підходи для ізоляції, а потім вже обиратимемо продукт, який підходить для ізоляції. Методи можна назвати наступні:

- ізоляція за допомогою кілець безпеки операційної системи,
- ізоляція за допомогою додаткових модулів операційної системи, як наприклад: Windows Security policies, Linux jailkit, SE
- ізоляція за допомогою віртуалізації
- ізоляція за допомогою контейнеризації

Ізоляція за допомогою кілець безпеки не є рішенням даної проблеми, оскільки реалізація залежить від операційної системи. Варто згадати складність відслідковування проблем у такому програмному забезпеченні при виникненні помилок. Таке відслідковування часто потребує хороших технічних навиків та спеціалізованого програмного забезпечення. Крім того, виконання програмного коду із помилками на понижених кільцях безпеки часто призводить до аварійної зупинки операційної системи-носія. Однак

варто зазначити, що типово виробники операційних систем надають дуже розгорнуту документацію для понижених кілець безпеки.

Ізоляція за допомогою модулів операційної системи зазвичай безпечніша для операційної системи носія, оскільки помилки в програмному коді не призводять до аварійної зупинки операційної системи. Однак часто доводиться описувати набір правил для кожного файлу чи процесу цільового застосунку. Варто згадати, що типово такі модулі не є портативними, тобто не можуть бути запущеними на операційних системах різного сімейства.

Розглянемо ізоляцію за допомогою віртуалізації. Віртуалізація – це метод запустити повноцінну операційну систему всередині емульованого середовища. Програмні системи, які дозволяють проводити віртуалізацію, називаються гіпервізори. Гіпервізори можуть бути трьох типів.

Тип1. Автономні гіпервізори. Програмна система гіпервізора безпосередньо є операційною системою. Перевагою такої системи віртуалізації є те, що система таким чином споживає куди менше ресурсів, оскільки працює напряму з апаратним забезпеченням.

Тип2. Гіпервізор на основі базової операційної системи. Перевагою таких гіпервізорів є те, що вони більш легкі в налаштуванні та менш вимогливі до набору апаратного забезпечення. Але оскільки такі гіпервізори працюють поверх іншої операційної системи, то вони не можуть ефективно розподіляти апаратні ресурси між емульованими операційними системами та операційною системою-носієм

Тип3. Гібридний гіпервізор. Інтерфейс такого типу гіпервізорів теж будується поверх основної операційної системи, але сам програмний код гіпервізора запускається всередині спеціалізованої операційної системи, яка працює на нижніх кільцях безпеки операціної-системи носієм.

Системи віртуалізації надзвичайно добре підходять для ізоляції застосунків. Але загалом мають кілька недоліків.

Внутрішні стандарти збереження файлів з віртуалізованими операційними системами не є сумісним між різними гіпервізорами або ж

потребує додаткових коверторів. Крім того різні гіпервізори емулють різне апаратне забезпечення, тому після процесу зміни гіпервізора потрібне додаткове налаштування віртуалізованої операційної системи.

Гіпервізори першого типу не можуть бути запущені поверх іншої операційної системи, оскільки є самостійними операційними системами.

Гіпервізори третього типу не можуть бути запущені поверх сторонніх операційних систем, оскільки такі системи розробляються під певну несучу операційну систему.

Загалом можна сказати, що гіпервізори не є оптимальним рішенням, оскільки після збірки власної програмної системи поверх віртуалізованої операційної системи, віртуалізована операційна система разом із створеним програмним комплексом є прив’язаною до несучої системи віртуалізації.

Далі розглянемо контейнеризацію.

Контейнеризація – це система ізоляції застосунків, яка дозволяє застосунку бути запущеним у власному просторові імен. Простори імен контролюються на низьких рівнях кілець захисту операційної системи та дозволяються повністю ізолювати оперативну пам’ять, яка даних так і програмного коду, від інших процесів. Типово системи контейнеризації дозволяють підключати емульовану файлову систему. Фактично це значить, що застосунок має власний набір бібліотек, необхідних для запуску. Цей набір бібліотек є повністю незалежним від основної операційної системи. Крім того системи контейнеризації дозволяють застосунку використовувати власний мережевий стек, який не перетинається з основною операційною системою. Значною перевагою такого методу ізоляції застосунків є те, що не потрібно встановлювати операційну систему повністю, оскільки контейнер використовує не повноцінну операційну систему, а ядро основної операційної системи. Фактично це означає що не використовуються додаткові апаратні ресурси на емуляцію ядра операційної системи та набору необхідних драйверів у порівнянні з гіпервізорами.

Крім того, на даний момент існує відкритий стандарт для систем контейнеризації Open Container Initiative Distribution v1.0, який описує властивості, які має мати контейнер, щоби працювати на різних системах контейнеризації.

На даний момент на ринку є кілька продуктів, які підтримують систему контейнеризації. Але основними можна назвати наступні https://docs.microsoft.com/en-us/virtualization/windowscontainers/deploy-containers/containerd

- Hyper-V Containers
- Containerd
- Docker

Hyper-V Containers підтримуються лише операційною системою Windows. Containerd працює лише на апаратному забезпеченні, яке базується на процесорах сумісних із стандартом x86\amd64 . Docker же може працювати на операційних системах сімейства Windows та *nix. Також докер портований на arm Linux, тобто підтримує різні апаратні платформи. Варто зазначити, що і Docker, і Containerd, і Hyper-V Containers підтримуються централізованою системою управління контейнерами

Для подальшої розробки використовуватимемо Docker, оскільки дана система підтримує установку на більшу кількість операційних систем та на більшу кількість апаратних платформ.

Програмна система Docker дозволяє захищати програмне забезпечення від загроз 1, 2 та 3-го класів. Але не є стійкою до загроз 0-го класу. Звісно, є фізичні методи захисту, але застосовувати превентивні пасивні системи захисту теж потрібно. Тим паче, такі міри дозволять захистити саму програмну систему Docker від загроз 1, 2 та 3-го класів.

4.2 Захист програмного забезпечення та даних, ізольованих за допомогою програмної системи Docker, від несанкціонованого перехоплення пристроїв вводу.

Описане нижче рішення дозволяє перенести ввід з клавіатури у мережевий стек. Таким чином, клас загрози знижується з 0 до 1-го. Методи боротьби відомі. Для захисту від мережевих загроз 1-го класу пропонується використовувати довірену машину та шифрований метод зв'язку. Шифрування реалізоване дякуючи використанню протоколу віддаленого зв'язку ssh.

Життєвий цикл контейнера Docker складається з кількох етапів:

1. Створення образу для розгортання контейнерів. Під час цього етапу також у образ встановлюються необхідні бібліотеки для запуску застосунку, а також сам код застосунку. Керування процесом створення виконується за допомогою директив, записаних у так званий Dockerfile.
2. Розгортання контейнера з образу, підключення до мережевих інтерфейсів, запуск контейнера.
3. Виконання додаткових інструкцій для наповнення контейнера даними.
4. Зупинка контейнера. Цей пункт не є обов'язковим.
5. Повторний запуск контейнера після зупинки. Цей пункт не є обов'язковим.
6. Зупинка та видалення контейнера.
7. Видалення образу для розгортання контейнерів. Цей пункт не є обов'язковим.

Особливістю роботи програмної системи Docker є те, що у контейнері може бути тільки один головуючий процес, всі інші мають бути його нащадками. При завершенні роботи головуючого процесу контейнер примусово зупиняється. Тому якщо потрібно запустити всередині контейнера кілька сервісів, наприклад сервіс ssh та сервіс ізольованого застосунку,

необхідно використовувати додатковий менеджер процесів. Це може бути самописний код, а можна використати менеджер процесів supervisord. Розглянемо процес створення образу контейнера та запуску контейнера. Код автоматизації створення образу та запуску контенера написаний на мові bash. У цьому образі у якості ізольованого програмного забезпечення використовується вебсервер nginx.

```
#!/usr/bin/bash
# Зазначення портів, які будуть використовуватися у
образі
hostPort1="443"
hostPort2="80"
hostPort3="23"
servicePort1="443"
servicePort2="80"
servicePort3="23"
# Ім’я гілки git, яка використовується у проекті
branchName="develop"
# Автоматичне визначення імені проекту
serviceName=$(echo "${PWD##*/}")
# Зупинка попередньої версії контейнера
docker stop ${serviceName}_${branchName}
# Видалення попередньої версії контейнера
docker rm ${serviceName}_${branchName}
# Видалення попередньої версії образу контейнера
docker rmi ${serviceName}:${branchName}
# Створення образу контейнера та передача параметрів
збірки. Крапка у команді вказує, що Dockerfile
знаходиться у поточній директорії
docker build -t ${serviceName}:${branchName} . --no-
cache=false --build-arg servicePort1=${servicePort1} --
```

```
build-arg servicePort2=${servicePort2} --build-arg
servicePort3=${servicePort3} --build-arg
serviceName=${dockerInternalName}
# Запуск контейнера
docker run --name ${serviceName}_${branchName} --
privileged=true --restart always -p
${hostPort1}:${servicePort1} -p
${hostPort2}:${servicePort2} -p
${hostPort3}:${servicePort3} -d
${serviceName}:${branchName}
```

Цьому скрипту для роботи необхідний Dockerfile. Він описаний нижче

```
# Цей образ створений на базі образу ubuntu:20.04 .
Якщо інші параметри не вказані, то образ буде отриманий
з репозиторію DockerHub
FROM ubuntu:20.04
# Отримання аргументів від скрипта автоматизації
ARG servicePort1
ARG servicePort2
ARG servicePort3
ARG serviceName
# Оновлення списку пакунків
RUN apt-get update
# Конфігурація часової зони
RUN export DEBIAN_FRONTEND=noninteractive
RUN apt-get install tzdata
RUN ln -fs /usr/share/zoneinfo/Etc/UTC /etc/localtime
RUN dpkg-reconfigure --frontend noninteractive tzdata
RUN export DEBIAN_FRONTEND=dialog
# Установка додатково програмного забезпечення для
відслідковування проблем.
```

```
RUN apt-get install -y \
 procps \
 vim \
 lsof \
 tree \
 htop \
 iputils-ping \
 dnsutils \
 telnet \
 net-tools \
 tcpdump
# Установка програмного забезпечення для роботи з
репозиторіями
RUN apt-get install -y \
 git \
 wget \
 curl \
 unzip \
# Установка сервера ssh, системи керування процесами
supervisord та вебсервера nginx
RUN apt-get install -y openssh-server supervisor nginx
# Зміна паролю для користувача за замовчанням
RUN echo 'root:password' | chpasswd
# Створення директорій, необхідних для роботи
програмного забезпечення всередині контейнера
RUN mkdir -p /var/lock/apache2 /var/run/apache2
/var/run/sshd /var/log/supervisor /run/sshd

#Встановлення дозволу на доступ по ssh користувачу за
замовчанням
```

```
RUN sed -i 's/PermitRootLogin without-
password/PermitRootLogin yes/' /etc/ssh/sshd_config
# Копіювання конфігурації supervisord із поточної
директорії збірки. Опис конфігурації буде даний нижче.
COPY supervisord.conf
/etc/supervisor/conf.d/supervisord.conf
# Підміна порта ssh на 23 для того, щоби він не
конфліктував із shh на несучій машині.
RUN echo "Port ${servicePort3}" >> /etc/ssh/sshd_config
RUN echo "PermitRootLogin yes" >> /etc/ssh/sshd_config
# Директиви, які вказують на те, які порти можуть бути
використані образомж
EXPOSE ${servicePort1}
EXPOSE ${servicePort2}
EXPOSE ${servicePort3}
# Директива, яка показує, яку саму команду потрібно
запустити після створення контейнера
CMD ["/usr/bin/supervisord"]
```

У Dockerfile для збірки використовується файл `supervisord.conf` . Опис цього файлу нижче.

```
[supervisord]
# Конфігурація supervisord для роботи у Docker
nodaemon=true
# Конфігурація ssh
[program:sshd]
command=/usr/sbin/sshd -D
# Конфігурація nginx
[program:nginx]
command=/usr/sbin/nginx -g "daemon off;"
priority=900
```

```
stdout_logfile= /dev/stdout
stdout_logfile_maxbytes=0
stderr_logfile=/dev/stderr
stderr_logfile_maxbytes=0
username=www-data
autorestart=true
```

Вказаний вище код може працювати на на сімействі операційних систем *nix, на сімействі операційних систем Windows за допомогою підсистеми WSL2 та на ARM Linux.

4.3 Захист програмної системи Docker від несанкціонованого фізичного доступу до обладнання ІС.

Найпростіший метод захисту від несанкціонованого фізичного доступу – це шифрування холодних даних на диску після вимикання апартаного програмного забезпечення. Для цього у запропонованому нижче рішенні усі дані програмної системи Docker примусово шифруються та після зупинки операційної системи зберігаються у зашифрованому файлі. Після запуску операційної системи можна запустити програмну систему Docker з даними із зашифрованого файлу. Для програмної системи Docker таке рішення виглядає прозорим та не потребує її реконфігурації, оскільки операційна система Linux дозволяють прозоро підміняти директорії у дереві каталогів за допомогою монтування. Запропоноване нижче рішення дозволяє використовувати різні утиліти для шифрування, але в даному конкретному випадку використана утиліта cryptofs, яка використовує стандартизований модуль шифрування LUKS операційної системи Linux. За замовчанням в даній конфігурації використовується асиметричне шифрування AES із парольною фразою. Описаний нижче код написаний для операційної системи Ubunu Server 20.04 LTS на інтерпретованій мові програмування bash. Код складається із трьох файлів:

- bootstrap.sh призначений для початкової конфігурації захищеного тому.
- mount.sh для відновлення роботи після перезавантаження несучої операційної системи.
- unmount.sh для зупинки роботи зашифрованого тому та Docker перед вимкненням операційної системи.

Для початку розглянемо принцип роботи bootstrap.sh. У цьому файлі спочатку створюється файл, який потім буде змонтований як віртуальний диск. Далі диск шифрується. Наступним кроком іде створення на цьому дискові файлової системи. Далі файлова система декриптується і підключається як звичайний блочний пристрій. Після на неї переносяться дані Docker. Після цього директорії файлової системи підміняють системні директорії Docker. Опис коду bootstap.sh нижче:

```
#!/usr/bin/bash
# Задаємо директорію, у якій буде зберігатись файл
зашифрованого тому
CRYPTFS_ROOT=/cryptfs
# Встановлюємо групу утиліт cryptofs
# apt-get update
# apt-get -y install cryptsetup
#Створюємо директорію для зберігання файлу
зашифрованого тому
mkdir -p $CRYPTFS_ROOT
# Створюємо файл зашифрованого тому та змінюємо права
доступу до нього
truncate -s 30G $CRYPTFS_ROOT/disk
chmod -R 700 "$CRYPTFS_ROOT"
#Підключаємо файл як віртуальний жорсткий диск
LOOP_DEVICE=$(losetup -f)
losetup $LOOP_DEVICE $CRYPTFS_ROOT/disk
```

```
# Заповнюємо файл том випадковими послідовностями
badblocks -s -w -t random -v $LOOP_DEVICE
#Шифруємо том
cryptsetup -y luksFormat $LOOP_DEVICE
# Підключаємо том як віртуальний пристрій
cryptsetup luksOpen $LOOP_DEVICE cryptfs
# Створюємо файлову систему
mkfs.ext4 /dev/mapper/cryptfs
# Створюємо директорію для монтування декриптованої
файлової системи
mkdir -p /mnt/cryptfs
# Монтуємо декриптований том
mount /dev/mapper/cryptfs /mnt/cryptfs
# Зупиняємо службу Docker
systemctl stop docker
# Налаштовуємо директорії для переносу
for DIR_NAME in var/lib/docker
do
   Cp -Ra /${DIR_NAME} /mnt/cryptfs/
# Робимо монтування з декриптованої файлової системи
зашифрованого тому із маскуванням оригінальних
директорій Docker
   mount --bind /mnt/cryptfs/${DIR_NAME} /$DIR_NAME
done
# Запускаємо Docker уже захищеним
systemctl start docker
# Перевіряємо коректність запуску Docker
systemctl status docker
```

Нижче описаний код скрипта mount.sh

```
#!/usr/bin/bash
```

```
# Задаємо директорію, у якій буде зберігається файл
зашифрованого тому
CRYPTFS_ROOT=/cryptfs
#Підключаємо файл як віртуальний жорсткий диск
LOOP_DEVICE=$(losetup -f)
losetup $LOOP_DEVICE $CRYPTFS_ROOT/disk
cryptsetup luksOpen $LOOP_DEVICE cryptfs
# Монтуємо декриптований том
mount /dev/mapper/cryptfs /mnt/cryptfs
# Зупиняємо службу Docker
systemctl stop docker
# Робимо монтування з декриптованої файлової системи
зашифрованого тому із маскуванням оригінальних
директорій Docker
for DIR_NAME in var/lib/docker
do
   mount --bind /mnt/cryptfs/${DIR_NAME} /$DIR_NAME
done
# Запускаємо Docker уже захищеним
systemctl start docker
# Перевіряємо коректність запуску Docker
systemctl status docker
```

Нижче описаний код скрипта unmount.sh.

```
#!/usr/bin/bash
# Задаємо директорію, у якій буде зберігається файл
зашифрованого тому.
CRYPTFS_ROOT=/cryptfs
# Знаходимо пристрій, за допомогою якого був
змонтований віртуальний том.
```

```
LOOP_DEVICE=$(losetup -a | grep $CRYPTFS_ROOT | grep -
oP "^[^:]*")
# Зупиняємо службу Docker
systemctl docker stop
# Розмонтовуємо маскуючі директорії
for DIR_NAME in var/lib/docker; do
  umount /$DIR_NAME
done
# Розмонтовуємо зашифрований том
umount /mnt/cryptfs
cryptsetup luksClose cryptfs
losetup -d $LOOP_DEVICE
```

Вказаний вище код може працювати на на сімействі операційних систем *nix, на сімействі операційних систем Windows за допомогою підсистеми WSL2 та на ARM Linux.

ВИСНОВКИ

Отже, підводячи підсумки в даній дипломній роботі, попередньо проаналізувавши весь вище викладений матеріал, ми можемо зробити наступні висновки:

В першому розділі даної дипломної роботи, ми досліджували та аналізували предметну область, нами було досліджено розвиток багатопроцесорних систем, їх видів та класифікації.

На підставі проведено дослідження, нами було встановлено, що з ростом числа процесорів просто неможливо обійти необхідність реалізації моделі розподіленої пам'яті з високошвидкісною мережею для зв'язку процесорів.

У другому розділі проведено порівняння різних архітектур багатопроцесорних систем та підходів до організації зв’язку між обчислювальними вузлами.

В третьому розділі даної дипломної роботи, ми досліджували методи захисту багатопроцесорних систем від кіберзагроз, нами було досліджено основні види кіберзагроз. Наведена класифікація кіберзагроз та методи боротьби з різними класами загроз.

На підставі проведено дослідження, нами було встановлено, що на сучасному етапі все більшу роль в подальшому розвитку інформаційних ресурсів грають паралельні обчислювальні системи і обчислення. Подібні системи знаходять застосування в сфері економічних, технологічних та інших процесів.

В четвертому розділі висунуті вимоги до програмної системи ізоляції застосунків та даних на основі класифікації кіберзагроз. Було проведене дослідження представлених на ринку методів та засобів ізоляції програмного забезпечення. Як основу для подальшої роботи було обрано програмну систему Docker, як таку, що відповідає усім вимогам та має певні переваги над конкурентами, а саме підтримку різних операційних систем та апаратних платформ.

В цьому ж розділі розділі розглянуто життєвий цикл контейнера Docker. Показана нестійкість контейнерів Docker до загроз 0-го класу.

Запропоновані методи захисту від таких атак, як перехоплення вводу з клавіатури та несанкціоноване зчитування даних контейнерів при фізочному доступові.

Методи реалізовані програмно у вигляді скриптів bash та директив для збірки образу Docker.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. . Лацис А.О. Как построить и использовать суперкомпьютер / А.О. Лацис. – М.: Бестселлер, 2003. – 240 с.

2. Beowulf Introduction & Overview [Електронний ресурс]. – Режим доступу: http:#www.beowulf.org.

3. Blanke, M., Kinnaert, M., Lunze, J., & Staroswiecki, M. (2015). Diagnosis and Fault-tolerant Control, 3rd Edition. (3. edition ed.) Springer. DOI: 10.1007/978-3-662-47943-8.

4. C+M+P architecture – URL : http:#www.cmparchitecture.it/ (дата звернення: 20.04.2021).

5. K. Abrougui and M. Elhadef. Parallel Self-Diagnosis of Large Multiprocessor Systems Under the Generalized Comparison Model. In Proc. of the 11th Int. Conf. on Parallel and Distributed Systems, Fukuoka, Japan. (To appear)., Jul. 2005.

6. Kusano K., Sato M., Hosomi T., Seo Y. The Omni OpenMP Compiler on the distributed shared memory of Cenju-4 # Lect. Notes Comput. Sci. — 2011. — Vol. 2104. — P. 20–30.

7. Math-Net.Ru; Vedeshenkov V.A., “On Diagnosis of Failed Modules in Digital Systems by Three-Module Chains”, Autom. Remote Control, 61:8, Part 2 (2000), 1382–1389.

8. NUMAFLEX – URL : http:#www.fatihanmakina.com/en/product-detail-7411-numaflex (дата звернення: 20.04.2021).

9. NUMAFLEX (NA) – URL : https:#www.unitedconveyor.com/sdm_downloads/numaflex-na/ (дата звернення: 20.04.2021).

10. NUMA-Q – URL : https:#www.pcmag.com/encyclopedia/term/numa-q (дата звернення: 20.04.2021).

11. Prepatata FP, Metze G., Chien RT On the Connection Assignment Problem of Diagnosable Systems. / / IEEE Trans. Electronic Comput. - 1987. - Vol.EC16, № 6. - P.848-854.

12. Romankevich V. A. Self-testing of multiprocessor systems with regular diagnostic connections / V. A. Romankevich # Automation and Remote Control. – 2017. – Vol. 78, Issue 2. – P. 289 – 299.

13. SMT and CMP Architecture – URL : https:#www.brainkart.com/article/SMT-and-CMP-Architectures_8858/ (дата звернення: 20.04.2021).

14. Аладышев О.С., Дикарев Н.И., Овсянников А.П., Телегин П.Н., Шабанов Б.М. СуперЭВМ: области применения и требования к производительности # Известия ВУЗов. Электроника. – 2014. – № 1. – С. 13-17.

15. Алишов Н.И. Развитые методы взаимодействия ресурсов в распределенных системах / Н.И. Алишов. – К.: Сталь, 2009. – 448 с.

16. Архитектура NUMA-Q – URL : https:#www.itweek.ru/idea/article/detail.php?ID=57531 (дата звернення: 20.04.2021).

17. Архітектура FAME в серверних рішеннях – URL : https:#www.bytemag.ru/articles/detail.php?ID=8576 (дата звернення: 20.04.2021).

18. Архітектура глобальної пам'яті із загальним доступом SGI NUMAflex і інтерконекту SGI NUMAlink в серверах і суперкомп'ютерах Silicon Graphics – URL : https:#sapr.ru/article/15597 (дата звернення: 20.04.2021).

19. Ведешенков В.А., “О диагностировании отказавших модулей в цифровых системах с помощью цепочек из трех модулей”, АиТ, 2000, №8, 156–164

20. Гергель В.П. Основы параллельных вычислений для многопроцессорных вычислительных систем: учеб. пособие / В.П. Гергель, Р.Г. Стронгин. – Н.Новгород: Н.НГУ, 2003. – 184 с.

21. Довганюк А. О. О последовательном диагностировании многопроцессорных систем/ Т.Г. Сапсай, Довганюк А.О. # Системный анализ и информационные технологии: материалы 20-й Международной научно-практической конференции SAIT 2018. – К.: УНК «ИПСА» КПИ им. Игоря Сикорского, 2018. – С. 186.

22. Евстигнеев В. А. NUMA-архитектура: некоторые особенности компиляции и генерации кода # Поддержка супервычислений и Интернет-ориентированные технологии — Новосибирск: ИСИ СО РАН. 2011. — С. 44–53.

23. Каляев А.В. Многопроцессорные вычислительные системы с программируемой архитектурой. – М.: Радио и Связь, 2014. – 240 с.

24. Каляев А.В. Программирование виртуальных архитектур в суперкомпьютерах с массовым параллелизмом # Информационные технологии и вычислительные системы. – 2000. – № 2.

25. Ковалевский В., Максимов В. Криптографические методы. # КомпьютерПресс. - 1993. - № 5. - с. 31-34.

26. Makimito T. The Rising Wave of Field Programmability # Proc. of Tehth International Conf. on FieldProgrammable Logic and Applications FLP-2000. – Villach (Austria). – 2000. – Springer Lecture Notes in Computer Science 2016. – P. 1-6.

27. Малюк А.А. Информационная безопасность – концептуальные и методологические основы защиты информации / А.А. Малюк. – М.: Горячая линия – Телеком, 2004. – 280 с.

28. Мафтик С. Механизмы защиты в сетях ЭВМ. - М.: Мир, 1996.

29. Романкевич В.О. Методи і засоби оцінки технічних характеристик гарантоздатності відмовостійких багатопроцесорних систем управління складними об’єктами: Дис. на здобуття наукового ступеня доктора технічних наук: 05.13.05; - Захищена 29.01.2018; Затв. 20.03.2018.- К., 2017.- 388 с.

30. Серверы CMP-архитектуры – URL : https:#www.itweek.ru/idea/article/detail.php?ID=56219 (дата звернення: 20.04.2021).

31. Что такое CMP? – URL : https:#www.osp.ru/cw/1998/33/31388 (дата звернення: 20.04.2021).

32. Ю. К. Димитриев, Эффективность локального самодиагностирования в вычислительных системах с циркулянтной диагностической структурой,

ПДМ, 2008, номер 2, 96–101

33. Ясинявичус Р. Параллельные пространственно-временные вычислительные структуры. – Вильнюс: Мокслас, 2018. – 183 с.

34. Ярков В. В. Статистичне дослідження показників кіберзлочинності: методологічний аспект / В. В. Марков # Право і безпека. – 2015. – № 2. – С. 136-140.

35. Погорецький М. Кіберзлочини : до визначення поняття / М. Погорецький, В. Шеломенцев # Вісн. прокуратури. – 2012. – № 8. – С. 89-96.

36. Anton D., Simion E. Linux Unified Key Setup (LUKS) - The Good, the Bad, the Ugly // 2018 10th International Conference on Electronics, Computers and Artificial Intelligence (ECAI). , 2018. С. 1–6.

37. Bajrami V. Configuring LUKS: Linux Unified Key Setup [Електронний ресурс]. URL: https://www.redhat.com/sysadmin/disk-encryption-luks (дата доступу 20.05.2021).

38. Bhattacharjee A., Lustig D. Architectural and Operating System Support for Virtual Memory / под ред. M. Martonosi. : Morgan & Claypool Publishers, 2017. 176 с.

39. Bugnion E., Nieh J., Tsafrir D. Hardware and Software Support for Virtualization. San Rafael, CA: Morgan & Claypool Publishers, 2017. 208 с.

40. Fruhwirth C. LUKS1 On-Disk Format Specification Version 1.2.3 С. 15.

41. Windows container platform [Електронний ресурс]. URL: https://docs.microsoft.com/en-us/virtualization/windowscontainers/deploy-containers/containerd (дата доступу 20.05.2021).

42. Stoneman E. Docker on Windows: From 101 to production with Docker on Windows. Birmingham Mumbai: Packt Publishing, 2017. 358 с.

43. About Windows containers [Електронний ресурс]. URL: https://docs.microsoft.com/en-us/virtualization/windowscontainers/about/ (дата доступу 20.05.2021).

44. Intro to Windows support in Kubernetes [Електронний ресурс]. URL: https://kubernetes.io/docs/setup/production-environment/windows/intro-windows-

in-kubernetes/ (дата доступу 20.05.2021).

45. Lesson 18: Application Isolation — OSU DevOps BootCamp 0.0.1 documentation [Електронний ресурс]. URL: https://devopsbootcamp.osuosl.org/application-isolation.html (дата доступу 20.05.2021).

46. Run Linux containers on Windows [Електронний ресурс]. URL: https://ubuntu.com/tutorials/windows-ubuntu-hyperv-containers (дата доступу 20.05.2021).