Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

# КВАЛІФІКАЦІЙНА РОБОТА

## Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Метод побудови віртуального операційного оточення для забезпечення ефективності обчислень

(тема)

Виконав:

студент II курсу, групи СПзм-22-1

Шевцов О.В.
(прізвище, ініціали)

Спеціальність

123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма

Системне програмування
(повна назва освітньої програми)

Керівник: доц. Федорченко В.М.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

# ЗАВДАННЯ

## НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Шевцову Олексію Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи Метод побудови віртуального операційного оточення для забезпечення ефективності обчислень

затверджена наказом по університету від " 01 " квітня 2024 р. № 45 Стз

2. Термін подання студентом роботи до екзаменаційної комісії 15 червня 2024 р.

3. Вхідні дані до роботи
паралельна програма
реконфігурована система
інформаційний граф

4. Перелік питань, що потрібно опрацювати у роботі
Класифікація розподілених обчислень
Проблеми розвитку ефективних розподілених комп'ютерних обчислень
Розробка методу для будівництва ефективного операційного середовища

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 17 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1 )

| Найменування розділу | Консультант (посада, прізвище, ім'я, по батькові) | Позначка консультанта про виконання розділу | |
|---|---|---|---|
| | | підпис | дата |
| | | | |
| | | | |

## КАЛЕНДАРНИЙ ПЛАН

| № | Назва етапів роботи | Термін виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Отримання завдання та аналіз літератури | 01.04.2024 – 06.04.2024 | |
| 2 | Огляд існуючих рішень та методів | 07.04.2024 – 12.04.2024 | |
| 3 | Розробка моделі | 13.04.2024 – 18.04.2024 | |
| 4 | Вибір програмних засобів | 19.04.2024 – 25.04.2024 | |
| 5 | Програмна реалізація | 26.04.2024 – 02.05.2024 | |
| 6 | Аналіз отриманих результатів | 03.05.2024 – 16.05.2024 | |
| 7 | Оформлення записки | 17.05.2024 – 14.06.2024 | |
| 8 | Представлення роботи в ЕК | 15.06.2024 | |
| | | | |
| | | | |
| | | | |
| | | | |

Дата видачі завдання 01 квітня 2024 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Федорченко В.М.
(підпис) (посада, прізвище, ініціали)

# РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 78 с., 24 рис., 1 дод., 25 джерело.

ПАРАЛЕЛЬНА ПРОГРАМА, РОЗПОДІЛЕНА ОБРОБКА ДАНИХ, ІНФОРМАЦІЙНИЙ ГРАФ, РЕКОНФІГУРОВАНА СИСТЕМА.

Метою кваліфікаційної роботи *є* аналіз методів побудови розподілених віртуальних обчислювальних систем.

У ході виконання кваліфікаційної роботи був розроблений новий підхід до побудови віртуального оточення, що забезпечує збільшення обчислювальної потужності за рахунок динамічного балансування навантаження, управління ресурсами і міграцією процесів на вузлах обчислювального середовища. Розроблен новий метод інтеграції різнорівневих програм на базі проміжного програмного забезпечення, що надається платформою для ефективного управління ресурсами обчислювальної системи.

# ABSTRACT

Master’s thesis: 70 pages, 24 figures, 1 appendices, 25 sources.

PARALLEL PROGRAM, DISTRIBUTED DATA PROCESSING, INFORMATION GRAPH, RECONFIGURED SYSTEM.

The purpose is to analyze the methods of building distributed virtual computing systems.

In order to a new approach to building a virtual environment was developed, which provides an increase in computing power due to dynamic load balancing, resource management and migration of processes on nodes of the computing environment. A new method of integration of different level programs based on middleware, which is provided by a platform for effective management of computing system resources, has been developed.

ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ОС – операційна система

GSI – Інфраструктура безпеки мережі

SSI – образ операційної системи

VLAN – Virtual Local Network

VMS – віртуальна машина

VPN – Virtual Private Network

# ВСТУП

Продуктивність обчислювальних середовищ визначається їх доступністю, переважно оцінюється системами балансування навантаження, системами управління ресурсами, виконанням робочого навантаження та обробкою даних для вирішення величезної кількості завдань.

Щоб задовольнити всі ці вимоги, нам потрібне потужне обчислювальне середовище з надійними системами балансування навантаження, загальною системою пам'яті та надійною системою обробки даних з величезним обсягом даних. Отже, кілька років тому використання віртуальних машин дало великі можливості для управління ресурсами за допомогою динамічної міграції віртуальних машин.

За допомогою технологій віртуалізації фізичний сервер можна розділити на кілька ізольованих середовищ виконання, розгорнувши шар поверх апаратних ресурсів або операційної системи (ОС). Серверні середовища виконання та віртуальні машини (VMS) можуть працювати одна на одній, не перебиваючи одна одну. Кожна віртуальна машина має власну ОС та додатки.

На самому початку технології віртуалізації не використовувались широко з різних причин. У хмарних обчисленнях великі компанії можуть об'єднати свої запасні апаратні ресурси та орендувати їх клієнтам на платній основі, а користувачі можуть швидко розпочати роботу на віртуальній машині, не витрачаючи величезних витрат на придбання та обслуговування обладнання. Оскільки всі користувачі вибирають хмарні центри обробки даних для зберігання своїх програм, ефективне управління віртуальними машинами стало дуже актуальним у центрі обробки даних.

Згідно з останніми експериментами, поєднання технології віртуалізації та SSI дозволяє нам отримувати кращі рішення та досягати гнучкості та простоти використання ресурсів. Віртуальна машина додає рівень гнучкого

управління між апаратним забезпеченням та програмами, тоді як SSI забезпечує абстракцію розподілених ресурсів. Одночасне використання двох технологій дозволяє збільшити загальне використання ресурсів платформи, продуктивність та ефективність кластерних додатків.

Сіткові та хмарні обчислення - це сучасні обчислювальні технології, які складаються з різних ресурсів, що підтримуються різними організаціями. Управління такими системами занадто складне. Тому дослідники використовують різні методології та інструменти для аналізу, оцінки та прогнозування запропонованих ними алгоритмів та методів. Більшість опублікованих досліджень оцінювались на реальних інфраструктурах мережевого середовища або з використанням засобів цих технологій. Ці розподілені системи мають велику обчислювальну потужність, що дозволяє нам вирішувати найскладніші завдання, такі як прогнозування погоди та моделювання землетрусів.

Це надзвичайно складні системи географічно розподілених та незалежно керованих ресурсів. Деякі з багатьох аспектів, які слід врахувати, включають: управління ресурсами, неоднорідність, відмовостійкість, продуктивність мережі, безпеку, пристосованість, масштабованість та прозорість.

Метою даної роботи аналіз методів побудови розподілених віртуальних обчислювальних систем.

Завдання:

- розробка/модифікація методу побудови віртуального операційного оточення, який забезпечить підвищення обчислювальних потужностей;

- дослідження системи керування розподіленими ресурсами в віртуальному обчислювальному середовищі.

Об’єкт дослідження – розподілені віртуальні обчислювальні системи.

# 1 КЛАСИФІКАЦІЯ РОЗПОДІЛЕНИХ ОБЧИСЛЕНЬ

## 1.1 Розподілені обчислення

Розподілені обчислення (рисунок 1.1) - це підхід до вирішення ресурсоємних завдань, що використовує кілька комп'ютерів, підключених до мережі.

Розподілена обчислювальна система - це сукупність незалежних комп'ютерів, з'єднаних каналами зв'язку, які з точки зору користувача певного програмного забезпечення виглядають як єдиний блок. Вони є особливим випадком паралельних обчислень.

Паралельна програма включає кілька процесів, що працюють разом, щоб виконати певне завдання. Паралельна програма включає кілька процесів, що працюють разом для виконання певного завдання. Кілька процесів працюють разом, взаємодіючи між собою, і потрібна синхронізація (наприклад, взаємне виключення або умовна синхронізація).

Сфера використання паралельних обчислень постійно розширюється, охоплюючи нові сфери. Використання розподілених систем стає майже повсюдним не лише для наукових розрахунків, але і для повсякденних ділових завдань (лише пам’ятайте про веб-сервери з великим навантаженням) [6].

Для запуску паралельних додатків використовуються багатопроцесорні системи, окремий випадок яких є розподіленими системами. Багатопроцесорні системи існують у різних конфігураціях.

Найпоширенішими типами таких систем є:

- системи для високопродуктивних обчислень - системи, призначені для паралельних обчислень. Багатопроцесорні обчислювальні системи для високопродуктивних обчислень зазвичай збираються з багатьох комп’ютерів, тому розробка таких систем є складним процесом, що вимагає постійного

узгодження таких питань, як одночасне управління великою кількістю комп’ютерів, спільний доступ до ресурсів тощо;

- системи високої надійності - це набір програмного та апаратного забезпечення, призначеного для забезпечення безперервної безперебійної роботи систем;

- багатопотокові системи - системи, що використовуються для забезпечення єдиного інтерфейсу до ряду ресурсів, які можуть бути довільно збільшені (або зменшені) з часом, типовим прикладом є група веб-серверів [2].

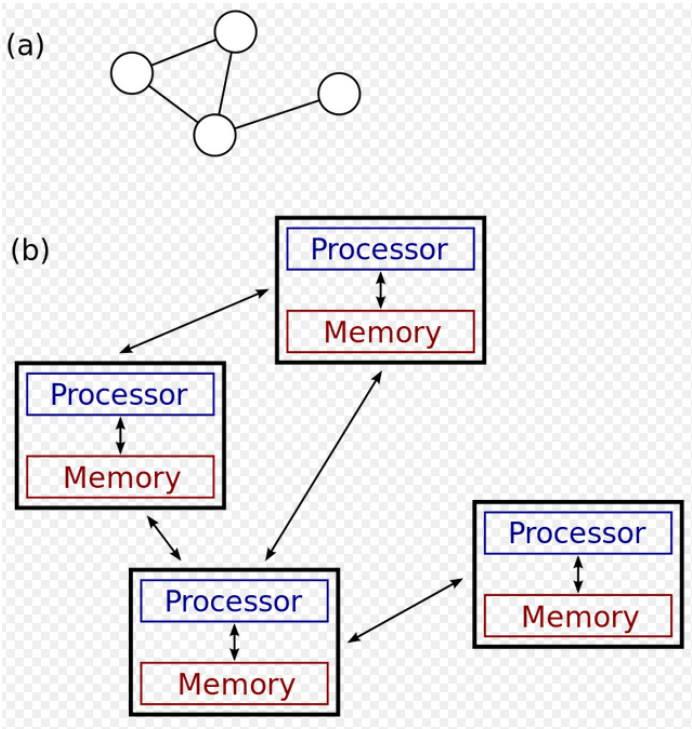


Рисунок 1.1 – Розподілені обчислення

Оптимізація конфігурації та підтримка багатопроцесорної системи є досить складними завданнями, не кажучи вже про написання паралельних програм для таких систем. Багато інструментів мають 134 створено для паралельного програмування у вигляді спеціальних мов програмування та API:

- потоки POSIX - стандарт POSIX для реалізації потоків (потоків), що визначає API для їх створення та управління ними;

- MPI (Message Passing Interface) - інтерфейс програми ( API) для передачі інформації, що дозволяє обмінюватися повідомленнями між процесами, які виконують одне завдання;

- PVM (Parallel Virtual Machine) - загальнодоступний програмний пакет, що дозволяє поєднувати різноманітний набір комп'ютерів у загальний обчислювальний ресурс;


Рисунок 1.2 – Класифікація розподілених систем

- OpenMP (Open Multi-Processing) - відкритий стандарт для розпаралелювання програм на C, C ++ та Fortran;

- OpenCL (Open Computing Language) - стандарт для написання паралельних програм, що дозволяє використовувати не тільки центральний процесор, але і графічний процесор , а також спеціалізовані прискорювачі.

Крім того, існують спеціалізовані системи, що дозволяють об'єднати комп'ютери в єдиний обчислювальний комплекс і збалансувати завдання:

- MOSIX - це система управління кластерами, об'єднуючи їх як єдину

систему (Single-System). Це операційна система, яка дозволяє керувати кластером як єдиною системою;

- ScaleMP -система, яка дозволяє об'єднати стандартні комп'ютери в єдину віртуальну SMP-машину (платний продукт, на відміну від MOSIX) [11].

## 1.2 Паралельні системи

Існує кілька типів паралелізму:

- найпростіший вигляд - паралелізм бітового рівня. Ця форма паралелізму заснована на збільшенні розміру машинного слова. Як відомо, 4-розрядні процесори були замінені 8-розрядними, і ці 16-, 32- і, нарешті, 64-розрядні.

- наступний тип паралелізму передбачає наявність командного конвеєра і називається паралелізмом на рівні інструкцій. Сучасні процесори мають багатоступеневий конвеєр інструкцій. Кожна стадія конвеєра відповідає певній дії, яку виконує процесор на цій стадії.

- Класичним прикладом процесора з конвеєром є процесор RISC з 5 кроками: отримання інструкцій з пам'яті, 135кодування інструкцій, виконання інструкцій, доступ до пам’яті, запис результатів до регістрів.

- Паралель даних передбачає виконання тієї ж операції над масивом даних. Різні фрагменти такого масиву обробляються на векторному процесорі або на різних процесорах паралельної машини. Розподіл даних між процесорами, які беруть участь у програмі.

- У цьому випадку векторізація або розпаралелювання найчастіше виконується вже на етапі компіляції - перекладаючи вихідний код програми в машинні команди. Роль програміста в цьому випадку, як правило, обмежується встановленням параметрів оптимізації векторного або паралельного компілятора, директив паралельної компіляції та використанням спеціалізованих мов для паралельних обчислень.

- Паралелізм завдань передбачає виконання окремого завдання на

кожному з обчислювальних вузлів [3].

## 1.3 Комп’ютерні архітектури

У 1966 році Майкл Флінн запропонував дуже зручну класифікацію обчислювальних систем. Він базувався на концепції потоку, яка розуміється як послідовність елементів, команд або даних, оброблених процесором. Відповідна система класифікації базується на врахуванні кількості потоків інструкцій та потоків даних та описує чотири архітектурні класи:

- SISD (одиничний потік інструкцій/одиночний потік даних) - одиночний потік інструкцій та одиночний потік даних Цей клас включає послідовні комп’ютерні системи, які мають один центральний процесор, здатний обробляти лише один потік послідовно виконуваних інструкцій. В даний час майже всі високопродуктивні системи мають більше центрального процесора танону, однак кожна з них виконує не пов'язані між собою потоки команд, що робить такі системи комплексами систем SISD, що працюють в різних просторах даних. Для збільшення швидкості обробки команд і швидкості арифметичних операцій можна використовувати конвеєризацію.

- MISD (кілька потоків інструкцій/один потік даних) - кілька потоків команд і один потік даних. Теоретично в цьому типі машин багато інструкцій слід виконувати в одному потоці даних. Але поки що не створено жодної реальної машини, що потрапляє до цього класу. Мабуть, роботу банку можна розглядати як аналог роботи такої системи. З кожного терміналу надсилаються різні команди, які, однак, мають справу з однією загальною базою даних.

- SIMD (одиничний потік інструкцій / декілька потоків даних) - одиничний потік команд і кілька потоків даних. Ці системи зазвичай мають велику кількість процесорів, від 1024 до 16384, які можуть виконувати одну і ту ж інструкцію щодо різних даних у жорсткій конфігурації. Одина інструкція виконується паралельно для багатьох елементів даних. Іншим

підкласом системи SIMD є векторні комп'ютери. Вони обробляють масиви подібних даних за допомогою спеціально розроблених векторних центральних процесорів. Слід зазначити, що до цієї категорії належить техніка GPGPU.

- MIMD (багатопотоковий потік інструкцій/безліч потоків даних) - кілька потоків інструкцій та декілька потоків даних. Цей тип машини одночасно виконує кілька потоків інструкцій на різних потоках даних. На відміну від багатопроцесорних машин SISD, згаданих вище, команди та дані в цьому випадку пов’язані, оскільки вони представляють різні частини одного завдання. Системи MIMD можуть одночасно виконувати багато підзадач, щоб скоротити час виконання основного завдання. Широке розмаїття систем, що потрапляють до цього класу, робить класифікацію Фліна не зовсім адекватною.

Дійсно, ця класифікація недостатньо детальна, тому необхідно ввести більш детальну класифікацію, що стосується різних архітектур та обладнання. Давайте розглянемо деякі конкретні архітектури обчислювальних систем.

SMP (Symmetric Multiprocessing) - симетрична багатопроцесорна архітектура. Головною особливістю систем з архітектурою SMP є наявність спільної фізичної пам'яті, спільної для всіх процесорів. Пам'ять служить, зокрема, для передачі повідомлень між процесорами, тоді як усі обчислювальні пристрої, отримуючи доступ до неї, мають рівні права та однакову адресацію для всіх комірок пам'яті. Тому архітектуру SMP називають симетричною. Остання обставина дозволяє дуже ефективно обмінюватися даними з іншими обчислювальними пристроями. Перш за все, переваги систем з такою архітектурою включають простоту їх програмування. Дійсно, архітектура SMP не накладає обмежень на модель програмування, яка використовується під час створення програми: зазвичай використовується модель паралельної гілки, коли всі процесори працюють незалежно один від одного. Однак моделі, що використовують

міжпроцесорний обмін, також можуть бути реалізовані. Використання спільної пам'яті збільшує швидкість такого обміну, програма також має негайний доступ до всієї пам'яті. Це повинно було слід зазначити, що для таких систем існують досить ефективні засоби автоматичної паралелізації. Також однією з переваг SMP-систем є простота експлуатації. Основний недолік - погана масштабованість. Причиною поганої масштабованості є той факт, що на даний момент шина здатна обробляти лише одну транзакцію, внаслідок чого виникають проблеми вирішення конфліктів, коли багато процесорів отримують доступ до одних і тих самих областей спільної фізичної пам'яті. Обчислювальні елементи починають заважати один одному. Коли виникає такий конфлікт, це залежить від швидкості зв'язку та кількості обчислювальних елементів [29].

MPP (Massive Parallel Processing) - масово паралельна архітектура. У цьому випадку система будується з окремих модулів, що містять процесор, локальну оперативну пам’ять, мережеві адаптери, іноді жорсткі диски та / або інші пристрої введення / виводу. Головною особливістю цієї архітектури є те, що пам’ять фізично розділена. Насправді такі модулі є повністю функціональними комп’ютерами. Головною перевагою таких систем є масштабованість. До недоліків можна віднести зменшенняшвидкості міжпроцесорного зв’язку (оскільки в цьому випадку немає спільної оперативної пам’яті, в результаті для реалізації обміну повідомленнями між процесорами потрібна спеціальна техніка програмування), обмежений обсяг оперативної пам’яті (оскільки тепер кожен процесор має свій власний банк оперативної пам'яті, як правило, не надто великий), висока ціна програмного забезпечення для таких систем (необхідно докласти зусиль для досягнення максимальної ефективності використання ресурсів).

NUMA (неоднорідний доступ до пам'яті) - архітектура з неоднорідним доступом до пам'яті. Гібридна архітектура поєднує в собі переваги систем із загальною пам'яттю та відносну дешевизну систем із загальною пам'яттю. Суть цієї архітектури полягає в особливій організації пам'яті, а саме: пам'ять

фізично розподіляється по різних частинах системи, але логічно вона є загальною, тому користувач бачить єдиний адресний простір. Система побудована з однорідних базових модулів (плат), що складаються з невеликої кількості процесорів та блоку пам'яті. Модулі поєднуються за допомогою високошвидкісного перемикача. Підтримується єдиний адресний простір, апаратно підтримується доступ до віддаленої пам'яті, тобто до пам'яті інших модулів. По суті, архітектура NUMA - це архітектура MPP, де вузли SMP приймаються як окремі обчислювальні елементи. Стів Волах запропонував ідею цієї архітектури.

cc-NUMA (когерентний когерентний нерівномірний доступ до пам'яті) - архітектура з неоднорідним доступом до пам'яті з когерентністю кеш-пам’яті. Удосконалення NUMA-систем, запропонованих Сеймуром Крей. Поняття когерентності кешу описується тим фактом, що всі центральні процесори отримують однакові значення однакових змінних у будь-який момент часу. Справді, оскільки кеш-пам’ять належить окремому комп’ютеру, а не всій багатопроцесорній системі в цілому, потрапляння даних у кеш одного комп’ютера може бути недоступним для іншого. Щоб цього уникнути, слід синхронізувати інформацію, що зберігається в кеші процесора.

PVP (паралельна векторна обробка) - паралельна архітектура з векторними процесорами. Ключовою особливістю таких систем є наявність векторних конвеєрних процесорів, які забезпечують інструкції щодо обробки однотипних незалежних векторів даних, які ефективно виконуються на конвеєрних функціональних пристроях. Як правило, кілька з цих процесорів працюють із спільною оперативною пам'яттю на одному вузлі. Кілька вузлів можна об'єднати за допомогою комутатора. Створення програм для таких систем передбачає векторизацію циклів та їх розпаралелювання.

Кластерна архітектура - набір робочих станцій загального призначення, пов’язаних між собою. Такі системи є найдешевшими, оскільки їх можна зібрати з існуючих комп’ютерів, у тому числі застарілих (згадайте різні

системи Beowulf). Зазвичай розрізняють такі типи кластерів: відмовостійкі кластери (відмінна особливість - надмірна кількість вузлів, що гарантують надання послуги у разі виходу з ладу одного або декількох серверів), кластери із збалансованим навантаженням (в даному випадку один або більше вхідних вузлів розподіляють запити між кількома вузлами, які обробляють ці запити), обчислювальні кластери (використовуються для ресурсоємних, як правило, наукових обчислень) [56].

GRID системи - фактично тип кластерної системи, що означає географічно розподілена архітектура.

## 1.4 Хмарні та GRID обчислення

Grid система - це «віртуальний суперкомп'ютер», що складається з кластерів та неміцно з'єднаних різнорідних комп'ютерів, що працюють разом для виконання величезної кількості завдань. З точки зору мережевої організації, сітка - це послідовне, відкрите та стандартизоване середовище, яке забезпечує гнучке, безпечне та скоординоване розділення обчислювальних та сховищних ресурсів, які є частиною цього середовища в рамках однієї віртуальної організації. Відмінними рисами є безпека мережі та географічно розподілені вузли, які можуть знаходитись де завгодно і зазвичай підключаються через Інтернет [3].

Наприклад, комп'ютери, розташовані в різних частинах світу, можуть працювати разом над одним завданням. Існує кілька типів мережевих систем: добровільна мережа (заснована на використанні безкоштовно наданого безкоштовного ресурсу персональних комп'ютерів), наукова та комерційна. Основною відмінністю хмарних обчислень від GRID є віртуалізація. Хоча GRID-системи забезпечують високе використання обчислювальних ресурсів, розподіляючи одне складне завдання між кількома обчислювальними вузлами, хмарні обчислення слідують шляху виконання кількох завдань на одному сервері як віртуальних машин. Віртуальне середовище дозволяє

розділити такі ресурси, як центральний процесор (CPU), пам'ять, введення-вивід та мережа з однієї хост-системи на кілька середовищ.

Обчислювальні ресурси GRID системи забезпечують користувачеві потужність обробки. Обчислювальними ресурсами можуть бути або кластери, або окремі робочі станції. При всьому різноманітті архітектур будь-яку комп'ютерну систему можна розглядати як потенційний обчислювальний ресурс сіткової системи. Необхідною умовою для цього є наявність проміжного програмного забезпечення, яке реалізує стандартний зовнішній інтерфейс з ресурсом і дозволяє зробити ресурс доступним для мережі-системи [5].

Основними загальними завданнями GRID є:

- створення широкомасштабних розподілених обчислювальних систем та систем для обробки, комплексного аналізу та моніторингу даних із серійного обладнання, джерела якого також можуть бути (глобально) розподілені;

- ефективність комп'ютерних технологій за рахунок надання тимчасових простоїв ресурсів у сітку. Пріоритет конкретного загального завдання, яке вирішується за допомогою сітки, визначається типом сітки та характером областей застосування, в яких вона використовується.

Як результат, досягнення наукових або інженерних цілей може бути важким, дуже дорогим, а часом і зовсім неможливим. Якщо ви можете використовувати ресурси багатьох персональних комп'ютерів, робочих станцій, кластерів, можливо навіть суперкомп'ютерів, а також сховищ даних, розташованих у різних частинах світу та належить різним людям та установам, на час виконання завдання чи проекту, тоді проблема може стати розв'язною. Цю можливість забезпечує середовище сітки для розподілу окремих частин великого завдання між географічно віддаленими ресурсами (якщо сам характер завдання дозволяє розділити його на частини) [35].

### 1.4.1 Архітектура GRID

Grid-технології - це розробка та узагальнення мета-обчислювальних ідей. В якості процесора розглядаються не тільки суперкомп'ютери, але і будь-які комп'ютери в цілому. Спільні ресурси: комунікації, дані, програмне забезпечення та час процесора. Grid - це послідовне, відкрите та стандартизоване середовище, яке забезпечує гнучке, безпечне, скоординоване спільне використання ресурсів у віртуальній організації. Grid характеризується відсутністю центру управління обчислювальними ресурсами, використанням відкритих стандартів та нетривіальним рівнем обслуговування. Обчислювальні вузли Grid, як правило, розташовані далеко один від одного, погано пов’язані між собою через Інтернет-канали , і наявність того чи іншого з них у будь-який момент часу не гарантується. Це накладає додаткові вимоги до управління ресурсами

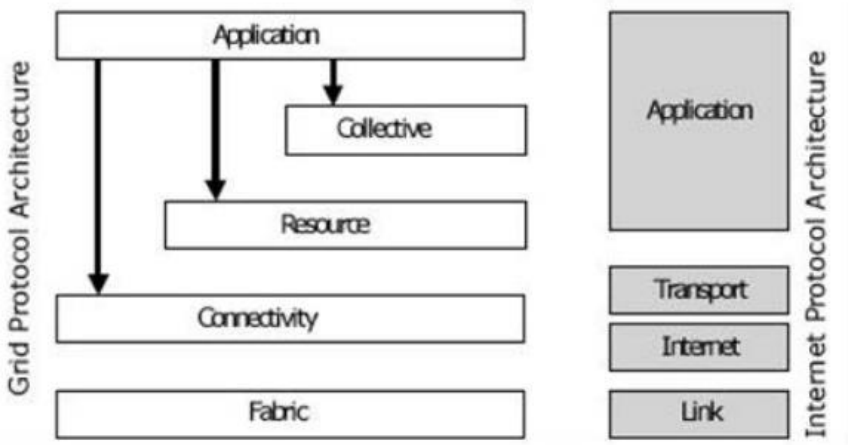


Рисунок 1.3 – Архітектура GRID

### 1.4.2 Архітектура SOA

Сервісно-орієнтована архітектура є основою для побудови надійних розподілених систем, які базуються на послугах зі стандартизованими

інтерфейсами та на принципі слабкого зв'язку між взаємодіючими службами.

Сервісно-орієнтована архітектура (SOA) має такі особливості.

- Архітектура розподілена. Функціональні модулі програми (системи) можуть бути розподілені між різними обчислювальними системами і здатні взаємодіяти за допомогою локальних або глобальних мереж. Інтерфейс функціональних модулів такий, що використання модулів не залежить від технології або платформи, в рамках якої вони реалізовані.
- Є можливість динамічного пошуку та підключення бажаних функціональних модулів.
- Архітектура базується на загальновизнаних галузевих стандартах [3].

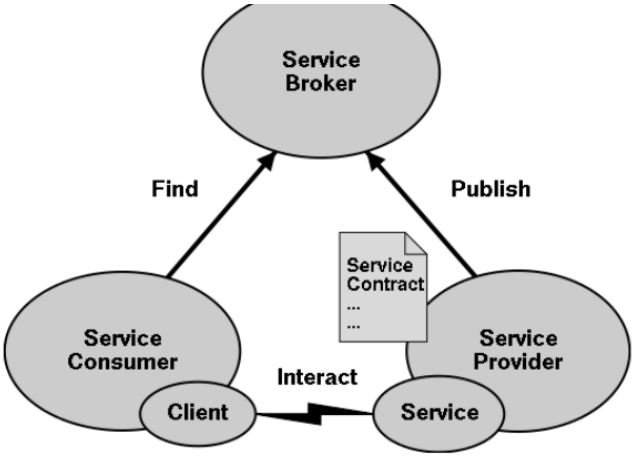


Рисунок 1.4 – Архітектура SOA

Технології хмарних обчислень пов’язані з темою високопродуктивних обчислень. Хмарні обчислення - це розподілена технологія обробки даних, при якій комп'ютерні ресурси та потужність надаються користувачеві як послуга Інтернету. Суть концепції хмарних обчислень полягає у наданні кінцевим користувачам віддаленого динамічного доступу до обчислювальних ресурсів, послуг та додатків (включаючи інфраструктуру та операційні системи) через Інтернет.

Таким чином, хмарні обчислення - це програмне та апаратне забезпечення, яке доступне користувачеві через Інтернет (або локальну

мережу) як послугу, що дозволяє використовувати зручний веб-інтерфейс для віддаленого доступу до виділених ресурсів (обчислювальних ресурсів, програм та даних). У той же час комп'ютер користувача діє як звичайний термінал, підключений до Мережі.

Комп’ютери, що виконують хмарні обчислення, називаються «обчислювальною хмарою». У цьому випадку навантаження між комп'ютерами, що входять до "обчислювальної хмари", розподіляється автоматично [8].

Можливості програмного забезпечення як послуги (SaaS) надаються споживачеві при використанні додаткових програм, що працюють на хмарній інфраструктурі. Додатки доступні для різних клієнтських пристроїв через тонкий клієнтський інтерфейс, наприклад веб-браузер (наприклад, мережева електронна пошта).

Споживач не управляє та не контролює базову хмарну інфраструктуру, включаючи мережу, сервери, операційні системи, сховище чи навіть окремі особливості програми, за винятком можливих обмежених конкретних параметрів конфігурації програми [17].

Платформа як послуга (PaaS). Споживачеві надається можливість розгортання на хмарній інфраструктурі, створеній споживачами або отриманих додатках, створених з використанням мов програмування та інструментів, що підтримуються постачальником. Споживач не управляє та не контролює базову хмарну інфраструктуру, включаючи мережу, сервери, операційні системи або сховище, але управляє розгорнутими програмами і, можливо, додатком, що змінює конфігурацію середовища. Інфраструктура як послуга (IaaS). Споживачеві надаються можливості у формі обробки, зберігання даних, мережевих та інших базових обчислювальних ресурсів, де споживач має можливість розгортати та виконувати довільне програмне забезпечення, яке може включати операційні системи та додатки. Споживач не керує та не контролює базову хмарну інфраструктуру, але управляє операційними системами, сховищем, розгорнутими програмами та

мережевими компонентами (наприклад, брандмауери хостів)

## 1.5 Неоднорідні обчислення

Використання графічних процесорів для загальних обчислень - нова перспективна сфера. Неоднорідні обчислення відкривають нові перспективи для вирішення складних завдань. Зараз ця сфера активно розвивається: наприклад, вже створений окремий стандарт. OpenCL - це відкритий стандарт паралельного програмування для різнорідних обчислень. Він визначає C-подібну мову для написання ядер та API. OpenCL дозволяє використовувати в розрахунках і CPU, і GPU. Її ключовою особливістю є переносимість коду. Відомо, що код для GPGPU завжди був специфічним для платформи. Різні постачальники пропонували власні API та мови програмування. Отже, перехід з однієї платформи на іншу, більш просунуту, може спричинити величезну проблему. Рішенням цієї проблеми є OpenCL. Найбільші виробники процесорів та графічних процесорів мають його реалізації.

CUDA (Compute Unified Device Architecture) - розроблена NVIDIA, яка дозволяє виконувати обчислення за допомогою графічних процесорів NVIDIA, що підтримують технологію GPGPU.

OpenCL (Open Computing Language) - це стандарт Khronos, основа для написання комп’ютерних програм, пов’язаних з паралельними обчисленнями на різних платформах. Центральна ідея OpenCL полягає у наданні програмісту універсального інструменту для використання всіх обчислювальних потужностей сучасних систем.

## 1.6 Технологі віртуалізації

Віртуалізація - це рівень програмного забезпечення, який знаходиться між апаратним забезпеченням та основною операційною системою і дозволяє

паралельно працювати з декількома операційними системами на одній фізичній машині. У цій галузі працює багато постачальників, приклади яких є Xen та VMWare. Віртуалізація забезпечує безпечне та ізольоване робоче середовище для всіх типів традиційного програмного забезпечення [10].

Віртуалізація серверних ресурсів - одна з основних програм віртуалізації. Такий підхід значно зменшує витрати на придбання та обслуговування серверів: вам не потрібно купувати та підтримувати багато серверів, які працюють за принципом "одна програма - один сервер", досить придбати потужний сервер, який буде рахувати багато додатків і підтримують його один. Ключовими моментами тут є безпека та відмовостійкість: серверні ресурси розподіляються між програмами так, що кожна програма використовує лише ті ресурси, які їй безпосередньо призначені.

Це усуває багато проблем: додатки більше не конфліктують через необхідність використання одних і тих же ресурсів (завдання з розподілу ресурсів призначаються відповідним інструментам), а "зависання" одного з додатків не впливає на роботу інших -бо кожен з них працює у своєму ізольованому середовищі.

Віртуалізація сервера існує у таких формах [10]: фізична (апаратна), використання віртуальної машини та віртуалізація на рівні операційної системи та у формі управління ресурсами сервера.

Для фізичної віртуалізації використовуються динамічні домени - апаратні розділи, засновані на декількох системних платах і з'єднані високошвидкісним перемикачем. Кожен з них працює як окремий комп'ютер, при цьому електрично ізольований, що дозволяє постійно модернізувати та обслуговувати. Цей підхід дозволяє динамічно перерозподіляти серверні ресурси та обслуговувати окремі домени, не призупиняючи роботу сервера в цілому.

Таким чином, обслуговування клієнтів не зупиняється ні на хвилину, і користувачі навіть не помітять збою в одному домені. Підтримка динамічних

доменів була вперше реалізована Sun Microsystems (нещодавно придбана корпорацією Oracle) більше десяти років тому, і вона залишається головний розробник інструментів, які використовують цю технологію донині. Прикладом рішення з використанням цієї технології є сервер Sparc Enterprise M9000, який підтримує до 24 динамічних доменів. У той час цей сервер був визнаний одним із найшвидших, демонструючи 2.023 TFlops на тесті LINPACK, який використовувався для визначення швидкості суперкомп'ютерів [14] [15].

Ідея віртуалізації додатків не нова і дуже популярна в корпоративному секторі. Це дозволило запускати застарілі програми в сучасному середовищі ОС. Віртуалізація породила подальший розвиток цієї ідеї та дозволила її значно вдосконалити. Віртуалізований додаток працює в окремому контейнері, який також містить необхідні налаштування середовища, змінні середовища, бібліотеки та ресурси. Це дозволяє зменшити взаємний вплив запущених програм між собою та операційною системою, а на практиці дозволяє одночасно запускати застарілі та несумісні програми, різні версії однієї програми, не враховуючи сумісність.

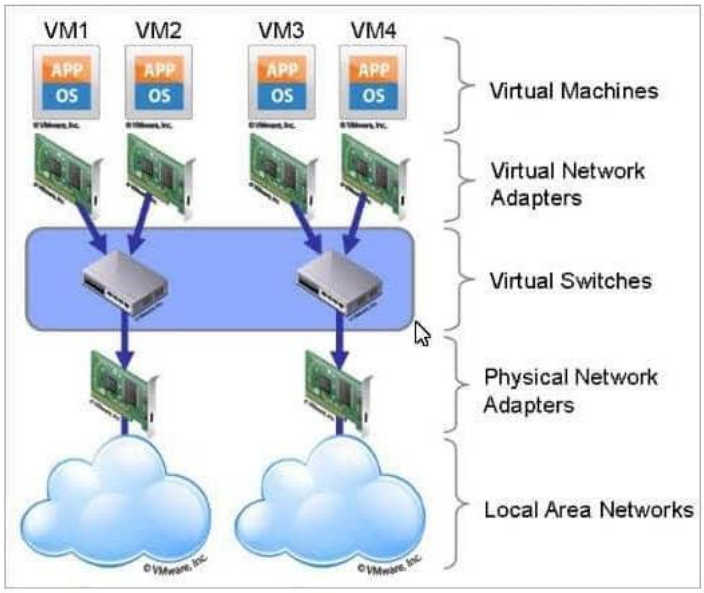


Рисунок 1.5 – Віртуалізація мережі

Віртуалізація пам'яті відноситься до інтеграції оперативної пам'яті у віртуальний пул пам'яті. У цьому випадку пам’ять окремого вузла з ним більше не пов’язана. Тепер будь-який комп'ютер у кластері має доступ до всього пулу пам'яті, тобто до пам'яті всіх комбінованих машин. Віртуалізація пам'яті дозволяє обійти фізичні обмеження обсягу пам'яті [1]. Це покращує загальну продуктивність системи та ефективність пам'яті. Існує два типи віртуалізації пам'яті.

Перший пропонує інтеграцію на рівні програми, коли окремі програми отримують доступ до спільної пам'яті безпосередньо через інтерфейси програмування. Другий - інтеграція на рівні ОС. У цьому випадку ОС встановлює з'єднання з пулом пам'яті і робить пам'ять пулу доступною для всіх програм. Необхідно розрізняти віртуалізацію пам'яті із загальною пам'яттю (Shared Memory).

Спільна пам’ять не дозволяє абстрагуватися від ресурсів. Віртуалізація пам’яті виключає зв’язок між логічними та фізичними ресурсами і призводить до ситуації, коли ресурси розподіляються за необхідністю. Технології NUMA та SMP використовують спільну пам’ять, але в рамках лише однієї багатопроцесорної системи, тоді як віртуалізація пам’яті передбачає об’єднання ресурсів кількох комп’ютерів у мережі.

Ця технологія передбачає абстракцію сховища інформації, при якій дані зберігаються на декількох пристроях, які об’єднані в загальну систему зберігання. Це забезпечує надійний контроль і консолідацію, об'єднання ресурсів зберігання в пул. Він забезпечує прозорий доступ до даних у багаторівневій системі. У той же час система зберігання, інтегрована у загальний віртуальний пул, який можна використовувати більш ефективно, оптимізує навантаження на окремий пристрій, виключаючи можливість вузьких місць, значно спрощуючи управління сховищем, зменшуючи адміністративні витрати та скорочуючи час і відновлення резервного копіювання, збільшує гнучкість та керованість, зменшує загальну вартість зберігання.

Цей тип віртуалізації впливає на область, яка стала однією з найважливіших у галузі обчислювальної техніки, комп'ютерних мереж. Фізичні ресурси мережі поділяються на кілька логічних віртуальних мереж. На основі фізичної мережі створюється кілька логічних, що використовують одне і те ж обладнання. Ви можете створити кілька віртуальних локальних мереж на базі одного підприємства для потреб окремих користувачів та груп, або ви можете створити віртуальну мережу поверх глобальної мережі, що найчастіше трапляється. Віртуальні локальні мережі (VLAN) використовуються для розподілу мережі між декількома підприємствами в одній будівлі або підрозділами одного підприємства, використовуючи ті самі фізичні ресурси для організації мережі. Віртуальні мережі поверх іншої мережі (VPN - Virtual Private Network) дуже зручні для віддаленого доступу до конфіденційних даних

# 2 ПРОБЛЕМИ РОЗВИТКУ ЕФЕКТИВНИХ РОЗПОДІЛЕНИХ КОМП’ЮТЕРНИХ ОБЧИСЛЕНЬ

## 2.1 Оптимізація прискорення обчислювального середовища

Проблема масштабованості була проблемою останніх років і навіть сьогодні, і вона відіграє великі успіхи в галузі інформаційних технологій та дослідницьких цілей. Потенційне прискорення, яке можна отримати від паралельної обробки, з тих пір було проведено великі дослідження в різних апаратних архітектурах.

Оцінка прискорення для критично важливих для часу програм є складною проблемою, що включає безліч факторів та точок зору, наприклад, справу з науковими програмами, що включають велике моделювання та граничні обмеження швидкості. Хоча велика кількість паралелізованих додатків вже добре працює між платформами клієнт-сервер.

Однак ці програми напрочуд паралельні, працюють на різних платформах, кожна з яких має свою власну операційну систему (ОС). Це дозволить нам краще оцінити виклики прискорення в галузі сучасних інформаційних технологій. Насправді прискорення також залежить від різних факторів, включаючи властивий паралелізм самій системі, апаратні архітектури машин та ОС з гнучкими об'єктами для розподілу та призначення процесорів та ресурсів пам'яті.

Закон Амдала заслуговує на особливу увагу. Коли організація має комп’ютер з вісьмома процесорами, на яких успішно працює паралельна програма, організація отримує кошти на розвиток і вирішує придбати більш продуктивний комп'ютер з 32 процесорами, очікуючи, що програма прискориться в чотири рази швидше. Але після покупки нового комп’ютера і запуску на ньому завдання виявляється, що програма почала працювати лише в 2 рази швидше. Причину розкриє закон Амдала. Джин Мірон Амдал -

інженер, який працював в IBM і засновник корпорації Amdahl. Він сформулював таку закономірність: час виконання обчислень у паралельній системі обмежується зверху часом виконання послідовної частини алгоритму. Іншими словами, «у випадку, коли завдання ділиться на кілька частин, загальний час його виконання в паралельній системі не може бути меншим за час виконання найдовшого фрагмента».

Отже, що передбачає закон Амдала? Наприклад, для алгоритму, 10% якого неможливо розпаралелювати, максимум, який ми можемо отримати, в десять разів перевищує прискорення. Цей факт проілюстровано на малюнку 1. Як бачите, навіть використовуючи 8000 процесорів для алгоритму, лише 5% з яких неможливо розпаралелювати, ми ніколи не отримаємо прискорення більше 20.

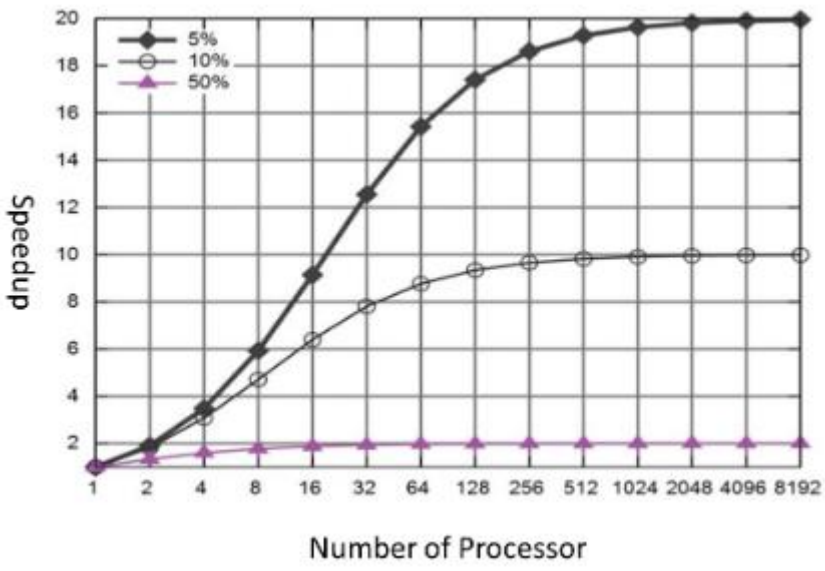


Рисунок 2.1 – Закон Амдала

Але на практиці ситуація ще гірша. Тут сприяє взаємодія між процесорами. Більше того, багато алгоритми передбачають інтенсивну взаємодію калькуляторів. Ця проблема особливо гостра для кластерів, підключених до мережі. Отже, затримки мережі стають тут головним обмежуючим фактором (порівняйте швидкість обміну інформацією між

процесорами на загальній платі та вузлами, підключеними до мережі за допомогою стандартного Ethernet). Ось чому при створенні кластерів особлива увага приділяється міжмережевій взаємодії: комутаторам, протоколам і середовищу передачі даних. Ось чому для високопродуктивних обчислень було створено кілька стандартів передачі даних (Infiniband, Myrinet та ін.).

Таким чином, коли кількість процесорів (вузлів у мережі) збільшується, настане момент, коли прискорення зупиняється, і процесори будуть заважати один одному, збільшуючи час виконання програми. У цьому випадку падіння прискорення буде досить різким, як це видно на рисунку 2.2.

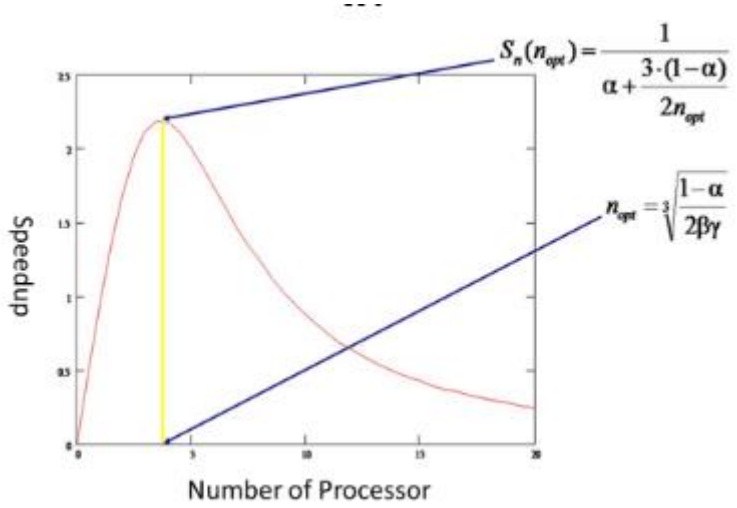


Рисунок 2.2 – Уточнення закону Амдала

З усього сказаного стає зрозумілим, наскільки важлива мережева інфраструктура та наскільки важливо вибрати оптимальну кількість вузлів, задіяних у розрахунках. Це також важливо з економічної точки зору: придбання навіть дорогих вузлів може не тільки пришвидшити обчислення, але і сповільнити їх.

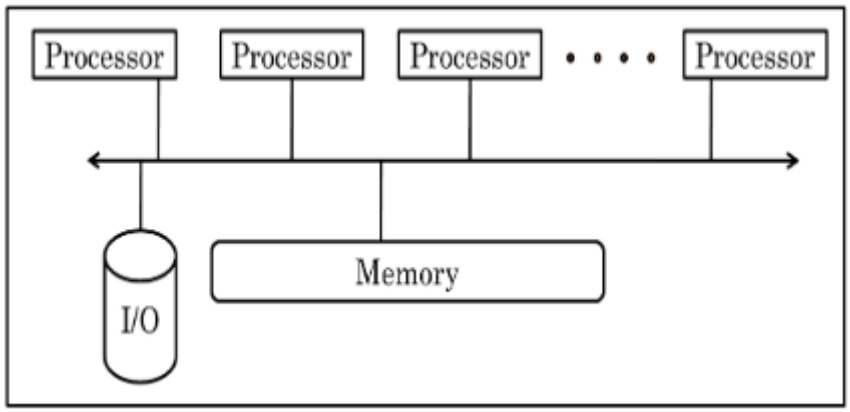


Рисунок 2.3 – Архітектура симетричних багатоядерних систем

Мережева взаємодія є важливою важливою проблемою паралельних обчислювальних систем. Проблеми організації міжмережної взаємодії є ключовими у розвитку розподілених систем. Саме вирішення цих проблем значною мірою визначає ефективність роботи системи в цілому. Таким чином, одним з найважливіших параметрів є співвідношення продуктивності процесора обчислювального вузла і пропускної здатності мережі. Ми аналізуємо закон Амдаля про моделі прискорення різних архітектур.

Симетричний багатоядерний чіп включає апаратну архітектуру, де двома або більше однаковими процесорами, підключеними до однієї спільної пам'яті, управляється за допомогою одного екземпляра ОС. Згідно із законом Амдала, прискорення симетричного багатоядерного чіпа (щодо використання одноядерного ядра BCE) залежить від програмної частки, яка паралелізується (f), загальної кількості чіп-ресурсів у BCE (n), ресурсів BCE (r) призначений для підвищення продуктивності кожного ядра.

В асиметричних мікросхемах процесори можуть мати різну продуктивність. Ця архітектура підтримує однопотокову продуктивність та багатопотокову пропускну здатність за менших витрат.

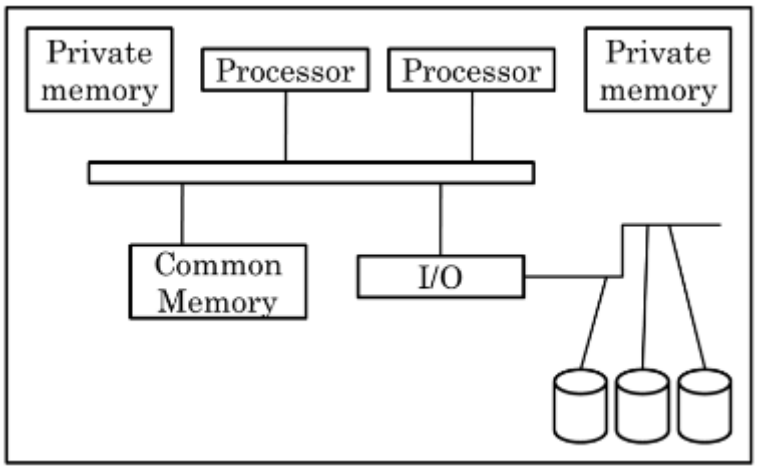


Рисунок 2.4 – Архітектура асиметричних багатоядерних систем

## 2.2 Проблеми організації взаємозв'язку в неоднорідних середовищах

Закон Амдала показав, що мережа є критичною і, мабуть, найважливішою проблемою паралельних обчислювальних систем. У випадку невеликої кількості процесорів на одному вузлі цілком достатньо загальної шини, до якої підключені як процесори, так і пам’ять. Але яка кількість процесорів велика? Звичайно, для побудови середовища перемикання потрібні інші підходи.

Спочатку розглянемо спосіб підключення вузлів (або процесорів) між собою. Сюди входять такі варіанти:

- плоска решітка - Плоска сітка - це найбільш природний і простий спосіб підключення процесорів. Навіть для системи з 16 процесорами цей метод неефективний.

- Куб або гіперкуб є другим найефективнішим способом з'єднання, проте найбільш інтуїтивним. Розмір гіперкуба буде визначений залежно від кількості процесорів, які потрібно підключити. Отже, для підключення 16 процесорів знадобиться чотиривимірний гіперкуб.

- Жирове дерево (товсте дерево) - найефективніший варіант. Цей варіант був запропонований Чарльзом Лейзерсоном у 1985 році. Процесори

локалізовані як у листі дерева, тоді як внутрішні вузли дерева згруповані у внутрішню мережу. Дерева стручків можуть взаємодіяти між собою, не впливаючи на більш високі рівні мережі. Звичайно, також використовуються інші варіанти підключення процесора. Наприклад, тор, кільце, зірка.

### 2.2.1 Вибір розподілених системних комутаторів

Комутатори забезпечують зв'язок між адаптерами обчислювальних модулів. Використовуються прості перемикачі, а також композитні перемикачі, що складаються з набору простих перемикачів. Прості комутатори можуть підключати лише невелику кількість вузлів через фізичні обмеження, але вони забезпечують мінімальну затримку при встановленні з'єднання.

Складені комутатори, як правило, побудовані з простих багатоступеневих схем з використанням точкових ліній, долають обмеження невеликої кількості з'єднань, але збільшують затримку:

- з тимчасовим розділенням (шиною) - у цьому випадку всі пристрої підключені до та сама інформаційна магістраль і конкурувати за доступ до неї;

- з просторовими пристроями поділу, які дозволяють одночасно підключити будь-який вхід до будь-якого виходу (звичайний) або кілька виходів (неординарний).

Розглянемо конкретні приклади побудови комунікаційних середовищ з використанням того чи іншого типу підключення та типу комутації:

- гігабітний Ethernet є загальноприйнятою технологією, але ефективною лише при підключенні точка-точка, при підключенні декількох вузлів її ефективність різко падає, а при підключенні більше 5-6 вузлів це не перевищує навіть Fast Ethernet за продуктивністю. Крім того, цей варіант є досить дорогим у порівнянні з тим же Fast Ethernet. Тому при створенні багатопроцесорних обчислювальних систем перевага часто віддається таким

технологіям, як SCI, Myrinet або Infiniband.

- SCI (масштабований когерентний інтерфейс) - стандарт, прийнятий для досягнення високої швидкості передачі даних із низькою затримкою, забезпечуючи при цьому масштабовану архітектуру. SCI - це комбінація шини та локальної мережі, що забезпечує узгодженість кеш-пам’яті, розміщеної на хості SCI, за допомогою розподіленого механізму каталогів, що покращує продуктивність, приховуючи витрати на доступ до віддалених даних у розподіленій моделі спільної пам'яті. Пікова пропускна здатність становить близько 10 Гбіт / с;

- Технологія Myrinet - заснована на використанні багатопортових комутаторів з обмеженнями на довжину зв'язку між вузлами та портами комутаторів до декількох метрів. На відміну від непоєднаних мереж Ethernet та FDDI, які мають спільне середовище, загальна пропускна здатність мережі Myrinet зростає із збільшенням кількості машин. Сьогодні Myrinet найчастіше використовується як відносно невелика локальна мережа (LAN), що зв’язує між собою комп’ютери всередині кімнати чи будівлі. Пікова пропускна здатність становить 10 Гбіт / с (Myrinet 10G)).

- Infinband - це високошвидкісна комутована послідовна шина, яка використовується як для внутрішніх (внутрішньосистемних), так і міжсистемних з'єднань. Основною метою Infiniband є міжсерверні з'єднання, в тому числі для організації RDMA (віддалений прямий доступ до пам'яті).

Використовується просторове перемикання та топологія товстого дерева. Пікова пропускна здатність становить 10, 20, 30, 40 Гбіт / с (залежно від режиму роботи). Існує безліч варіантів організації мережі. Рішення використовувати той чи інший варіант залежить від вимог до системи та цілей організації. Проблеми роботи в мережі Інтернет є ключовими для розвитку розподілених систем. Саме вирішення цієї проблеми багато в чому визначає ефективність роботи системи в цілому. Отже, одним з найважливіших параметрів є співвідношення продуктивності процесора обчислювального вузла та пропускної здатності мережі.

### 2.2.2 Управління даними та програмами в розподіленому середовищі

Управління даними асоціюється з колективним максимальним використанням в обмеженому просторі мереж пропускної здатності та обчислювальних ресурсів. У додатку були побудовані вимоги до даних, за допомогою яких можна було визначити, як дані рухатимуться по інфраструктурі або як-небудь оброблятись безпечно та ефективно. Стандартизація набору протоколів Grid дозволить зв’язок між будь-яким джерелом даних, доступним при розробці програмного забезпечення. Особливо інтенсивні програми часто мають об'єднані бази даних.

Проміжне програмне забезпечення, таке як Globe Toolkit (рисунок 2.5), забезпечує GridFTP та глобальний доступ до середніх служб передачі даних у середовищі сітки. Об'єкт GridFTP, що забезпечує безпечну та надійну передачу даних між хостами сітки. Розробники та користувачі стикаються з декількома важливими проблемами управління даними, які слід враховувати при розробці та впровадженні додатків. Наприклад, для великих обсягів даних це не є практичним, і може бути неможливо перемістити дані до системи, де робота фактично буде працювати.

Використання реплікації даних для копіювання або іншим чином підмножина всього набору даних до цільової системи може запропонувати рішення. Якщо мережеві ресурси, які географічно розподіляються зі швидкістю з'єднання, обмежені мережею, слід враховувати міркування щодо проектування щодо повільного або обмеженого доступу до даних. Безпека, надійність та продуктивність стають проблемою при переміщенні даних через Інтернет.

Коли доступ до даних може бути повільним або запобігти, вам потрібно побудувати необхідну логіку для вирішення цієї ситуації. Щоб забезпечити доступність даних у відповідному місці на той час, коли робота вимагає цього, користувач повинен спланувати передачу даних заздалегідь.

Слід також мати на увазі кількість і розмір будь-яких паралельних

передач до або з одного ресурсу одночасно. На додаток до основних вимог, описаних вище, для додатків для ефективної роботи на мережевих інфраструктурах, таких як балансири навантаження, мережеві посередники, зв'язок між процесами та легкі портали доступу, та нефункціональні вимоги, такі як продуктивність, надійність, аспекти топології та врахування змішаних середовищ платформ.

### 2.2.3 Проблеми розробки програмного забезпечення в розподілених середовищах

Розробка правильного програмного забезпечення для комп'ютерного комплексу набагато дорожча, ніж проектування та створення цього комплексу. Є багато паралельних додатків, але вони використовують лише невелику кількість моделей прийняття рішень або парадигм.

Вони представлені нижче.

- Ітераційний паралеліз відбувається, коли виконується декілька процесів, кожен з яких містить один або кілька циклів (наприклад, один великий цикл можна розпаралелювати, виконуючи кожну ітерацію в окремому процесі (або потоці)). Найчастіше зустрічається в наукових розрахунках, що виконуються на декількох процесорах.

- Рекурсивний паралеліс відбувається, коли програма має одну або кілька рекурсивних процедур, і їх виклики незалежні, тобто кожен з них працює над своєю частиною загальних даних. Рекурсивний паралелізм використовується для вирішення таких комбінаторних проблем, як сортування, планування (проблема продавця), ігор (шахи тощо).

- Процеси взаємодії виробників та споживачів. Вони часто організовані в конвеєр, по якому проходить інформація. Кожен конвеєрний процес є фільтром, який споживає вихідні дані попередника та виробляє вхідні дані для свого наступника. Фільтри знаходять як на рівні програми, так і в самій ОС.

- Клієнти та сервери - найпоширеніша модель взаємодії в розподілених системах. Клієнтський процес запитує сервер і чекає відповіді. Сервер очікує запитів від клієнтів, а потім діє відповідно до цих запитів. Сервер може бути реалізований як єдиний процес, який не може одночасно обробляти декілька запитів клієнта, або (при необхідності паралельне обслуговування запитів) як багатопотокова програма.

- Взаємодія однолітків - остання парадигма взаємодії, що зустрічається в розподілених програмах у які кілька процесів для вирішення проблеми виконують один і той же код і обмінюються повідомленнями. Взаємодіючі однолітки використовуються для реалізації розподілених паралельних програм, особливо з ітеративним паралелізмом та децентралізованим прийняттям рішень у розподілених системах.

## 2.3 Аналіз технологій та програмних продуктів для побудови різнорідного середовища

Для побудови ефективного різнорідного середовища ми обрали різні програмні продукти та технології, такі як Univa Grid Engine для управління ресурсами, платформа MOSIX для розробки ефективного комплексу, що забезпечує кращу продуктивність за рахунок балансування навантаження, Globus Toolkit для організації систем доступу, Grid-протоколи, що забезпечують безпека та управління даними, GridFTP для управління даними, програмний інтерфейс DRMAA для інтеграції програмних продуктів тощо.

Сучасні комп’ютерні технології дозволяють створювати відносно дешеві багатомашинні комплекси із спільними обчислювальними ресурсами. Такі системи забезпечують відносно низькі обчислювальні витрати, є високомасштабними, мають високий рівень надійності та мають перевірені інструменти для проектування, налагодження та аналізу паралельних програм.

Ефективність обчислювальних середовищ визначається їх доступністю,

в основному оцінюється за рахунок балансування навантаження системи, системи управління ресурсами, продуктивність навантаження та обробка даних для вирішення величезної кількості. Для виконання цих типів вимог нам потрібне потужне обчислювальне середовище з надійними системами балансування навантаження, системами спільної пам'яті та надійною системою обробки даних з величезною кількістю.

Globus Toolkit - це високопродуктивний програмний інструмент з відкритим кодом для мережі, що включає програмні послуги та бібліотеки для моніторингу, безпеки та управління ресурсами. У цій статті розглядаються методи організації системи доступу користувачів до розподіленого обчислювального середовища та методи було вивчено авторизацію користувача за принципом єдиного вікна.

Щоб забезпечити зручний та безпечний доступ для користувачів, ми вивчили надійні методи автентифікації та авторизації програмного продукту Globus Toolkit. Набір інструментів Globus - один з основних інструментів, що використовується для побудови Grid-систем та їх застосувань. Він був розроблений у 1996 році для підтримки розвитку розподілених обчислень, орієнтованих на послуги. Проект Globus виріс завдяки стратегії з відкритим кодом.

Це провокує ширший і швидший розподіл і призводить до більших технічних нововведень, оскільки спільнота користувачів постійно покращує якість продукту. Метою його створення є надання додаткам можливості працювати з розподіленими неоднорідними обчислювальними ресурсами як одна віртуальна машина. Основний фокус цього проекту - обчислювальні системи Grid. Обчислювальна система Grid означає інфраструктуру апаратних та програмних ресурсів, що забезпечує надійний і повномасштабний доступ до високопродуктивних обчислювальних систем, незалежно від географічного розташування користувачів або ресурсів [16].

Globus надає набір стандартних веб-служб для безпеки , управління даними, інформаційні послуги та виконання процесів, а також набір

службових програм і бібліотек, що дозволяють цим службам бути незалежними від платформи, функціонувати на низькому рівні стека веб-служб (WSRF, WSN) і дозволяють реалізовувати додаткові веб-сервіси з використанням різні мови програмування, такі як Java, C та Python. Це дозволяє використовувати GT4 для побудови як розподілених об'єктних систем, так і сервісно-орієнтованих архітектур.

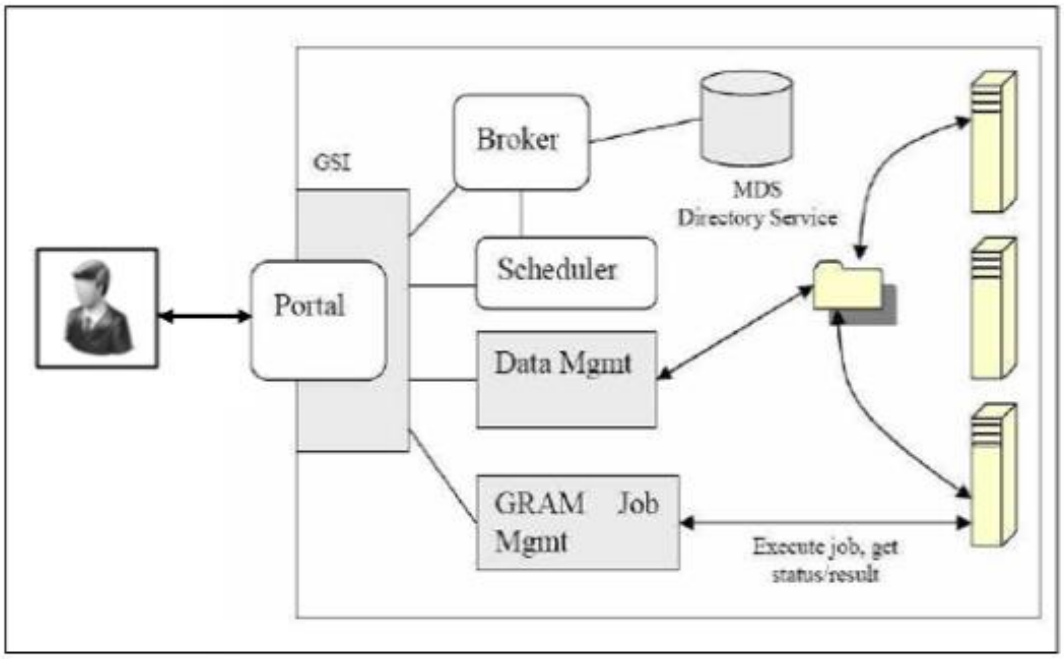


Рисунок 2.5 – Globus Toolkit

Основним елементом системи Globus є набір інструментів Globus, який описує основні послуги та можливості, необхідні для створення обчислювальних систем Grid. Система Globus надає додаткам високого рівня доступ до сервісів, кожен з яких додаток або розробник можуть використовувати для досягнення власних цілей. Цей спосіб роботи може бути реалізований лише в тому випадку, якщо існує висока ступінь ізоляції окремих служб і чітко визначений програмний інтерфейс для кожної наданої послуги.

## 2.4 MOSIX

Кластер MOSIX був створений на віртуальному тестовому сайті PM-PU для аналізу продуктивності з метою побудови їх ефективного обчислювального середовища. MOSIX: набір доповнень до ядра Linux для розподілу процесів між вузлами кластера. MOSIX можна логічно розділити на дві частини. Це міграція процесу та набір алгоритмів для ефективного динамічного розподілу ресурсів. Обидві частини реалізовані на рівні ядра операційної системи і є прозорими для програм.

Міграція здійснюється наступним чином. Процес розділений на два контексти - контекст користувача та контекст системи. Контекст користувача містить сегмент коду, стек, сегмент даних, вміст регістрів. Контекст системи містить інформацію про використовувані ресурси та стек ядра. Інтерфейс між користувачем та частинами системи встановлюється на рівні мережі. Користувацька частина може мігрувати, системна частина жорстко прикріплена до вузла, на якому був запущений процес [12,13]. У нашому кластері MOSIX ми перевірили ефективність наукового додатку CRYSTAL, який ефективно та одночасно працює з усіма розподіленими обчислювальними вузлами.

Crystal - це програмний засіб для хімії та фізики. Програма CRYSTAL була розроблена Групою теоретичної хімії та Науково-дослідною групою обчислювальних матеріалів в Туринському університеті. Він був розроблений насамперед для розрахунків кристалів (тривимірних), пластин (двовимірних) та полімерів (одновимірних) з використанням вимірів поступальної симетрії, але також може використовуватися для розрахунку окремих молекул. У цьому експерименті результати тесту на прискорення нашого обчислювального середовища MOSIX показують, що прискорення розрахунку наближається до ідеального. Результати випробувань наведені на рисунку (2.6).

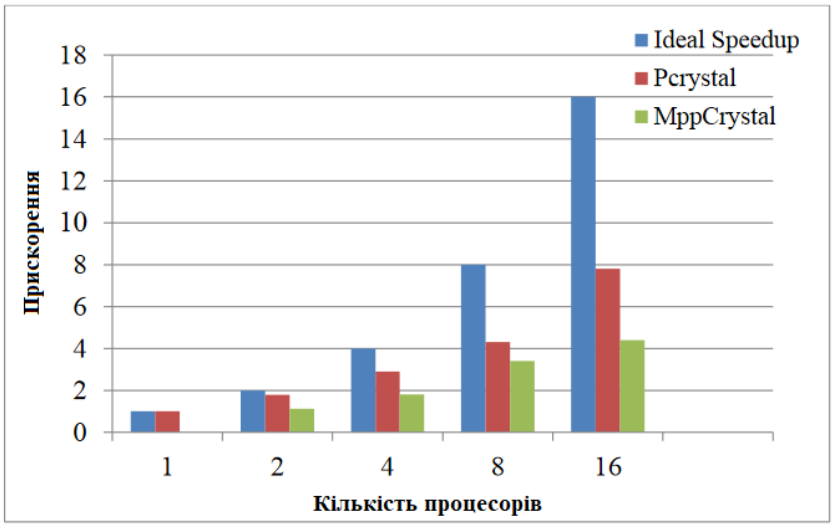


Рисунок 2.6 – *Аналіз продуктивності кластеру MOSIX*

Нещодавно розподілені обчислення є однією з привабливих архітектур високопродуктивних обчислень. Grid обчислювальна система - це розподілена обчислювальна система в масштабі Інтернету для розподілу розподілених ресурсів між традиційними прикордонними організаціями. У мережевих системах найважливішими питаннями є інтеграція динамічно неоднорідних розподілених ресурсів та підвищення ефективності використання цих інтегрованих ресурсів. Хоча ці різні сіткові проекти спрямовані на спільний доступ до розподілених ресурсів від різних віртуальних організацій (VO), все одно важко розподілити розподілені ресурси через різні цілі побудови іншої віртуальної організації [93]. Існує багато проміжних програм (наприклад, Globus Toolkit, Unicore, GLite тощо), розроблених для мережевих систем. Більшість із них зосереджені на наданні основних послуг проміжного програмного забезпечення для підтримки функціональних можливостей розробки додатків високого рівня. Однак, як правило, вони покладаються на спеціалізовані сервери для підтримки розподіленої інформації про ресурси.

# 3 РОЗРОБКА МЕТОДУ ДЛЯ БУДІВНИЦТВА ЕФЕКТИВНОГО ОПЕРАЦІЙНОГО СЕРЕДОВИЩА

У цій роботі ми використовували різні технології та програмні продукти, такі як Univa Grid Engine для управління різнорідними ресурсами, Єдиний образ операційної системи (SSI) для кращого балансування навантаження між операційними обчислювальними вузлами, технологію віртуалізації для консолідації та комбінування ресурсів обчислювального середовища з низькі витрати та економія енергії, а набір інструментів Globus використовується для безпечного моніторингу ресурсів та управління даними обчислювального середовища.

Інтерфейси та протоколи Grid забезпечують взаємодію між ресурсами, а також інтеграцію обчислювальних платформ. Більш високий рівень абстракції, який забезпечують обчислювальні платформи, може допомогти користувачам організувати прозоре та зручне надання ресурсів сіткової платформи та залучити нові групи користувачів до використання таких ресурсів. Для використання ресурсів користувач повинен пройти процедуру реєстрації та отримати особистий сертифікат користувача. На цьому етапі ми розробили автентифікацію та авторизацію користувачів із впровадженням сертифікатів для забезпечення безпеки системи додатків.

Інфраструктура безпеки мережі (GSI) - це компонент, який забезпечує API для цих цілей. Для шифрування інформації, яка надсилається в операційне середовище. GSI використовує асиметричну технологію шифрування з "відкритим ключем". GSI використовує цифрові сертифікати X. 509 як ідентифікатори користувачів та ресурсів.

Цифровий сертифікат - це відкритий ключ власника сертифіката з інтегрованою особистою інформацією, такою як ім’я користувача, електронна адреса, місце роботи, дата закінчення терміну дії сертифікату тощо. Кожен орган сертифікації має власну політику, яка визначає правила

створення та підписання сертифікатів. Globus використовує SAML (мова авторизації розмітки) для забезпечення захисту повідомлень, автентифікації, делегування та авторизації.

## 3.1 Розробка системи управління ресурсами для обчислювального середовища

Центральний компонент Univa Grid Engine керує розподіленою системою та кластером, приймаючи вхідні завдання від користувачів, призначаючи завдання ресурсам, контролюючи поточний стан кластера та обробляючи команди управління. Користувачі можуть надсилати завдання в основний демон за допомогою команди (qsub) і перевіряти їх стан (за допомогою команди qstat). Існує графічний інтерфейс (QMON) та інтерфейс програмування DRAMA (API розподіленого управління ресурсами), що дозволяє працювати з будь-яким іншим додатком або писати сценарії. Це інтерфейс, який ми використовували для інтеграції інструментів Globus та Linux. Користувачі Globus можуть запускати завдання за допомогою команди globusrun-ws через DRMAA.

## 3.2 Організація Univa Grid Engine та Mosix

Univa Grid Engine підтримує запуск паралельних програм із загальною та розподіленою пам'яттю. Такі паралельні програми вимагають певного паралельного середовища. Прикладами таких паралельних середовищ є:

- паралельні операційні системи із спільною пам'яттю;
- середовища розподіленої пам'яті з назвою Message Passing Interface (MPI);
- середовища розподіленої пам'яті з назвою Parallel Virtual Machine (PVM). Центральна частина інтеграції паралельного середовища з Univa Grid Engine полягає у правильній настройці процедур запуску та вимкнення.

Розповсюдження Univa Grid Engine містить різні приклади сценаріїв та програм C., які можуть бути використані як відправна точка для інтеграції PVM або MPI.

Існує багато таких систем, розроблених за останні роки для розподіленої та паралельної обробки на різних апаратних платформах. UGE може безпосередньо виконувати ряд завдань у паралельних середовищах, таких як PVM або MPI. Для планування завдань UGE пропонує простий інтерфейс, який визначається паралельним робочим середовищем. У системі UGE існує спеціальна структура даних, яка дозволяє враховувати особливості програм, що використовують технології паралельного програмування.

Для вирішення паралельних завдань UGE та Mosix були інтегровані з додатковим пакетом скриптів та бібліотекою Mpich2. Рішення проблеми управління процесами паралельних завдань для UGE базується на використанні механізму паралельного середовища разом з додатковим пакетом сценаріїв інтеграції UGE та MPICH2.

## 3.3 Інтерфейс програмного забезпечення DRMAA

API розподіленого управління ресурсами (DRMAA) - це інтерфейс програмування для програм розподіленого управління ресурсами. Для користувачів таких розподілених додатків DRMAA зберігає гнучкість і вибір системної архітектури. Специфікація DRMAA спрямована на уніфікацію інтерфейсів систем DRMS для досягнення мобільності між ними. Вона розробляється спеціальною робочою групою, яка є частиною OGF (Open Grid Forum). Через те, що представники різних комерційних та дослідницьких організацій брали участь у розробці стандарту DRMAA, він швидко був прийнятий громадою. В даний час існує кілька реалізацій DRMS, які підтримують цей API, серед яких Univa Grid Engine є найбільш повноцінним та стабільним. Специфікація API абстрактно описується мовою визначення

інтерфейсу IDL (Interface Definition Language), що дозволило реалізувати підтримку DRMAA для C / C ++, Java, Perl, Python, Ruby на рівні мов програмування. Специфікація DRMAA забезпечує що додаток Grid не залежить від використовуваного DRMS. Для цього в нього вводиться концепція категорії завдання. Відправляючи завдання в Grid, програміст встановлює категорію, яка відображається в певній системі, на набір параметрів, що може включати вказівку на вимоги до програми, пріоритет виконання та інші параметри, характерні для DRMS.

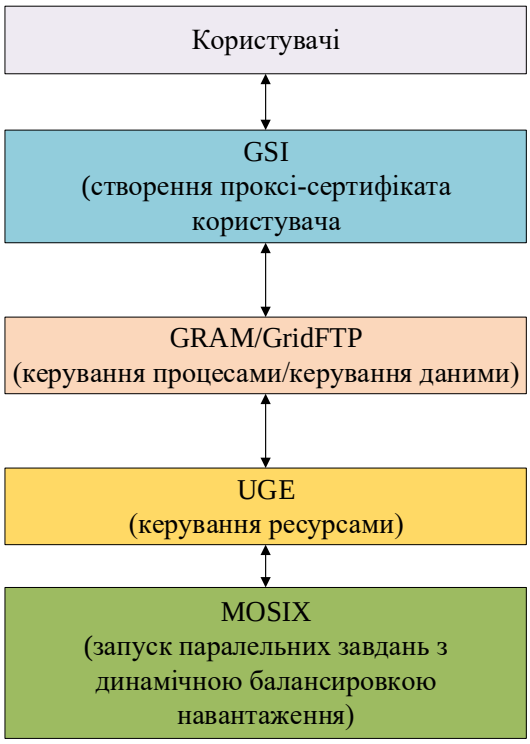


Рисунок 3.1 – Інтеграція програмних продуктів для створення робочого середовища розподіленого обчислювального середовища

## 3.4 Розробка методів динамічного вирівнювання навантаження обчислювального середовища

Однією з іманентних особливостей GRID є неоднорідність ресурсів, які повинні бути пов'язані між собою для вирішення однієї обчислювальної

проблеми. Для досягнення більш високої ефективності виконання в неоднорідному середовищі необхідні правильні стратегії збалансування навантаження. Балансування навантаження - це розподіл робочих навантажень між кількома комп’ютерами, вузлами кластерних обчислень, центральними процесорними блоками, дисководами чи іншими ресурсами для досягнення оптимального використання ресурсів, максимізації пропускної здатності, мінімізації часу спрацьовування та запобігання перевантаженню системи. Використання кількох компонентів, що врівноважують навантаження, замість одного компонента може підвищити надійність через надмірність.

Процесор передає інформацію з сильно завантажених серверів на недостатньо використовувані сервери. У агросистемі найважливіші питання включають, як інтегрувати динамічні різнорідні розподілені ресурси та як підвищити ефективність використання цих інтегрованих ресурсів. Для управління цими неоднорідними розподіленими ресурсами ми оптимізували найкращий спосіб міграції процесів з використанням одного образу операційної системи та створили віртуальний обчислювальний кластер під контролем гіпервізора та об’єднали обчислювальні ресурси в єдину обчислювальну систему, що показано на рисунку 3.2.

Без віртуалізації обчислювальне середовище не може управляти комп'ютерними процесами та динамічним балансуванням навантаження між вузлами для підвищення продуктивності при запуску паралельних та багатопоточних програм.

Процес міграції - це дія, яка передає процес між обчислювальними вузлами. Процеси міграції включають дані, стеки, зареєстрований вміст та стан основної операційної системи, такі як параметри, що стосуються оброблюваності, пам'яті та інформації про управління файлами. Процес міграції може поліпшити баланс навантаження та надійність розподілених обчислювальних систем.

Нещодавно деякі технології міграції були впроваджені шляхом

впровадження контрольної точки / перезапуску в процес міграції. Існує три цілі політики балансування навантаження, показані в наступному:

- розподілити навантаження з областей із високим навантаженням на зони з низьким навантаженням.
- забезпечити максимальне використання ресурсів.
- звести до мінімуму час виконання завдання.

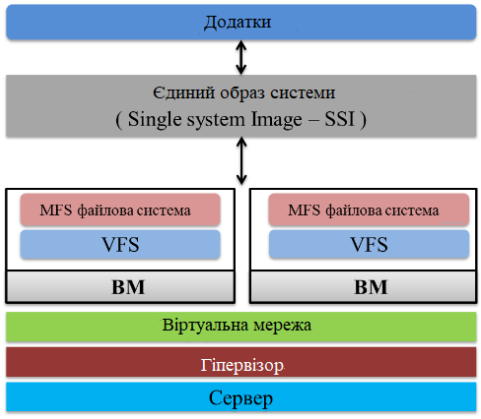


Рисунок 3.2 – Віртуальний кластер з одним зображенням операційної системи

Відповідно до підходу до прийняття рішень, політику балансування навантаження можна розділити на два типи. Статична політика балансування навантаження: приймає рішення про баланс щодо інформації про ресурси перед виконанням завдань. Легко реалізувати політику балансування статичного навантаження, а накладні витрати на реалізацію статичної політики нижчі, ніж реалізація динамічної політики. Однак статичному політику важче отримати оптимальну ефективність через те, що він не може регулювати рішення під час виконання. У системі високих варіаторів ефективність статичної політики дуже низька. Динамічна політика: динамічне балансування навантаження приймає рішення про розподіл ресурсів відповідно до інформації про виконання.

Хоча політика динамічного балансування приносить кращі показники,

її важко здійснити, оскільки вона повинна збирати динамічну інформацію, щоб прийняти найкраще рішення. Загалом, ця політика має кращі показники, ніж статична. Крім того, динамічне балансування може підвищити продуктивність системи в дуже різноманітних середовищах. З іншого боку, відповідно до підходу управління, політику балансування динамічного навантаження також можна розділити на два типи. Для збалансування навантаження в дисертації було обрано систему MOSIX.

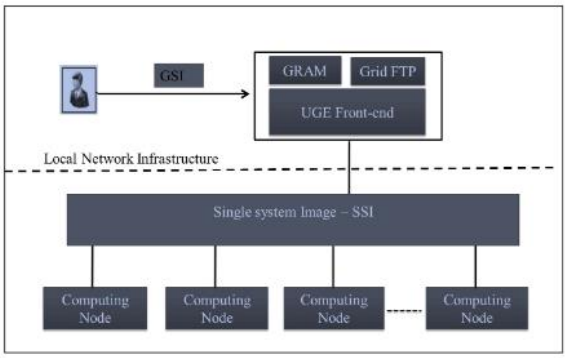


Рисунок 3.3 – Структура консолідації та інтеграції програмних систем

У нашому обчислювальному середовищі одним з найважливіших компонентів є файлова система. Як і в централізованих системах, у розподіленій системі функція файлової системи полягає у зберіганні програм і даних, а також наданні доступу до них за потреби. Файлова система MOSIX DirectAccess (DFSA), яка може покращити продуктивність кластерних файлових систем, дозволяючи процесу міграції безпосередньо отримувати доступ до файлів у поточному місці [7]. Така можливість у поєднанні з відповідною файловою системою може суттєво підвищити продуктивність вводу-виводу та зменшити перевантаження мережі, перемістивши процес, який інтенсивно використовує введення-виведення, на файловий сервер. DFSA підходить для кластерів, які керують спільним доступом пул дисків [1].

За допомогою DFSA ви можете переміщувати паралельні процеси з

вузла клієнта на файлові сервери для паралельного доступу до різних файлів. Щоб перевірити його ефективність, ми використали файлову систему MOSIX (MFS), яка забезпечує послідовні одночасні операції з різними файлами. Для використання DFSA був реалізований прототип файлової системи MOSIX (MFS). MFS забезпечує уніфікований перегляд усіх файлів у всіх встановлених файлових системах на всіх вузлах кластера MOSIX, ніби всі вони знаходяться в одній файловій системі/

## 3.5 Інтеграція системи віртуалізації та єдиного образу операційної системи (SSI)

Система (SSI) реалізована для спрощення складності управління та програмування кластерів. Ресурс у обчислювальному середовищі інтегрується за допомогою комбінації віртуалізації та розподіляється у спільній пам'яті (DSM), а для гостьової операційної системи в результаті єдиного образу системи досягається одна абстрактна фізична машина. Використовуючи розподілений монітор віртуальної машини (KVM) на основі апаратних технологій віртуалізації, DVMM містить деякі симетричні та спільні VMMS, розподілені по кількох вузлах. Таким чином, DVM може підтримувати немодифіковану застарілу операційну систему для роботи, очевидно, на розподілених вузлах кластера.

Наше рішення значно перевершує інші рішення SSI і має більшу прозорість, високу продуктивність та легкість впровадження. Віртуалізація дозволяє консолідувати та об’єднати ресурси.

Віртуалізація забезпечує логічну абстракцію фізичних обчислювальних ресурсів і створює обчислювальне середовище, яке не обмежується фізичною конфігурацією або реалізацією. Віртуалізація дуже важлива для обчислювальних мереж, оскільки надання послуг спрощується завдяки забезпеченню платформи для оптимізації різних ресурсів у масштабованій формі, що робить їх більш ефективними.

Гіпервізор відіграє важливу роль в апаратній віртуалізації. Це програмне забезпечення, яке забезпечує віртуалізоване апаратне середовище для підтримки роботи декількох операційних систем одночасно з використанням одного фізичного сервера. Віртуалізація в даний час широко використовується на підприємстві. Віртуалізація має інші основні переваги у високопродуктивних обчисленнях, такі як міграція, масштабованість гіпервізора тощо.

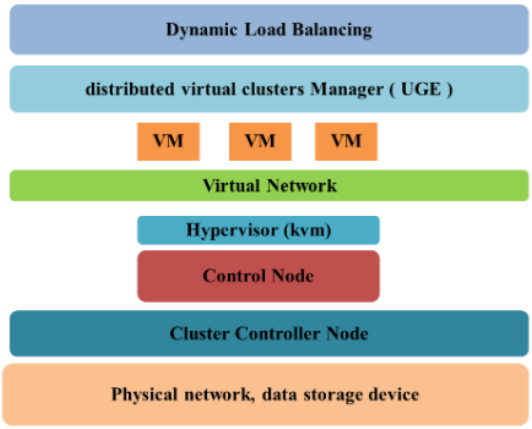


Рисунок 3.4 – Інтеграція віртуалізації та єдиного образу операційної системи (SSI)

### 3.5.1 Методи інтеграції віртуалізації

Віртуалізація є активною темою досліджень з 1970-х років, і новітні технології віртуалізації надають певні обчислювальні можливості (в останні роки нові машини з багатоядерними процесорами можуть конкурувати з кількома окремими серверами).

Спосіб використання цих додаткових функцій - це запуск VMS поверх фізичних машин. З нашої точки зору, концепція ВМ включає п'ять основних особливостей:

- ізоляція (ступінь ізоляції між оголеними VMS та програмами, що працюють на різних VMS);

- консолідація сервера (можливість переключення на ресурси, що виділяються до певної ВМ);

- переносимість додатка (можливість запуску немодифікованого додатка);

- переносимість ВМ (можливість міграції віртуальних середовищ через різні апаратні архітектури);

- зупинити / перезапустити (можливість зупинити/відновити VMS ).

Віртуалізація увімкнена в операційній системі, яка підтримує декілька ізольованих та віртуалізованих гостьових OSS на одному фізичному сервері з характеристикою того, що всі вони знаходяться в одному ядрі операційної системи з ексклюзивним контролем над апаратною інфраструктурою. Як операційна система, хост може переглядати віртуальні машини та керувати ними.

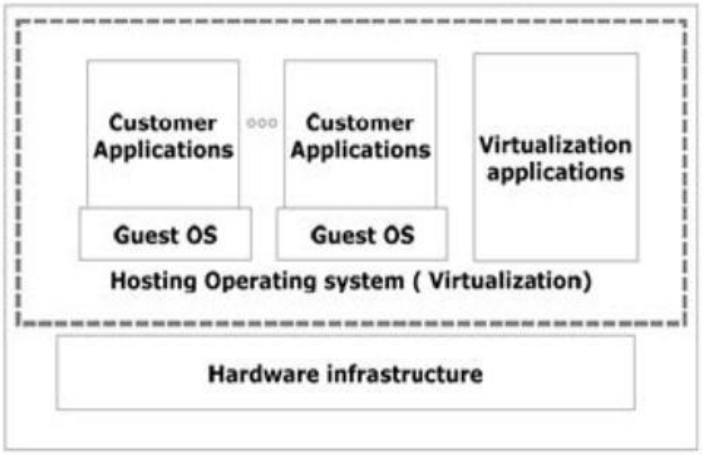


Рисунок 3.5 – Операційна система на основі віртуалізації

Застосунок на основі віртуалізації розміщується поверх розміщеної операційної системи. Ця програма віртуалізації, яка емулює всю віртуальну машину, містить свою гостьову операційну систему та відповідні програми.

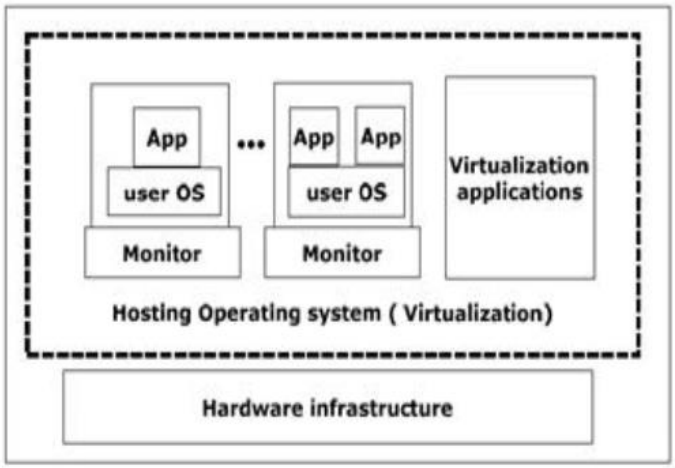


Рисунок 3.6 – Віртуалізація на основі додатків

Гіпервізор доступний під час завантаження машини для управління обміном системних ресурсів між кількома віртуальними машинами. Деякі з цих віртуальних машин є привілейованими розділами, які керують платформою віртуалізації та розміщеними віртуальними машинами. У цій архітектурі привілейовані розділи переглядають віртуальні машини та керують ними. Цей підхід створює найбільш контрольоване середовище і може використовувати додаткові функції безпеки, такі як система виявлення вторгнень.

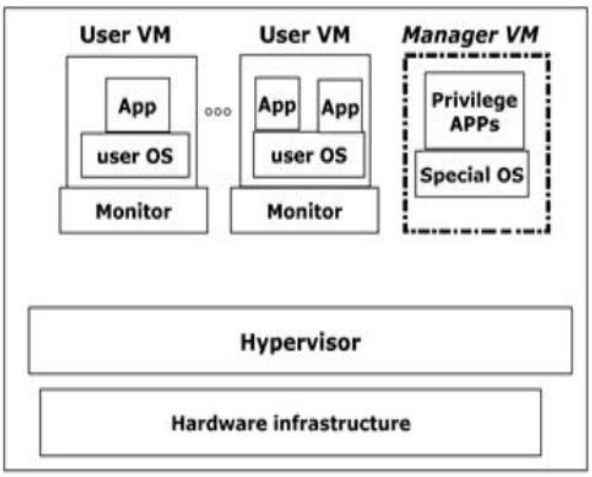


Рисунок 3.7 – Віртуалізація на основі гіпервізора

У нашому експерименті ми застосували метод віртуалізації на основі гіпервізора для консолідації та об’єднання ресурсів. Цей метод віртуалізації має деякі інші великі переваги у високопродуктивних обчисленнях, такі як процес міграції, масштабованість гіпервізорів та багато іншого.

Hypervisor, також відомий як монітор віртуальних машин (VMM), - це програмна платформа для віртуалізації. Гіпервізор дозволяє декільком операційним системам працювати на одному і тому ж апаратному хості. Кожна операційна система має власний процесор, пам’ять та ресурси для себе.

Гіпервізор керує центральним процесором та розподіленими ресурсами, які, в свою чергу, необхідні для кожної операційної системи, гарантуючи, що гостьові операційні системи або віртуальні машини не можуть порушувати один одного. У віртуальному середовищі монітор віртуальної машини (VMM) - це головна програма управління з максимальним рівнем привілеїв, і VMM керує однією або кількома операційними системами.

Гіпервізори поділяються на два типи.

Власні гіпервізори - програмні системи, які працюють безпосередньо на головному програмному забезпеченні як апаратне управління та операційна система монітора гостя. Таким чином, гостьова операційна система працює на іншому рівні над гіпервізором. Це класична реалізація архітектур віртуальних машин.

Хост гіпервізори - програмні додатки, що працюють у звичайному робочому середовищі. Враховуючи рівень гіпервізора, окремий програмний рівень, гостьові операційні системи таким чином працюють на третьому рівні над апаратним забезпеченням.

Ми розробили та впровадили монітор віртуальної машини (VMM), який дозволяє віртуалізувати спільну пам’ять багатопроцесорної кластерної машини. Ця функція значно полегшує користувачеві використання кластера. Наприклад, це дозволяє паралельним програмам спільної пам'яті для

багатопроцесорних систем працювати на кластерах, не змінюючи додатків. Крім того, операційні системи, що підтримують мультипроцесори (наприклад, Linux), можуть бути встановлені у VMS з невеликими змінами. Гіпервізор (VMM) забезпечує повний контроль над апаратною машиною та створює віртуальні машини, кожна з яких поводиться як цілісна фізична машина, що може запускати власну операційну систему.

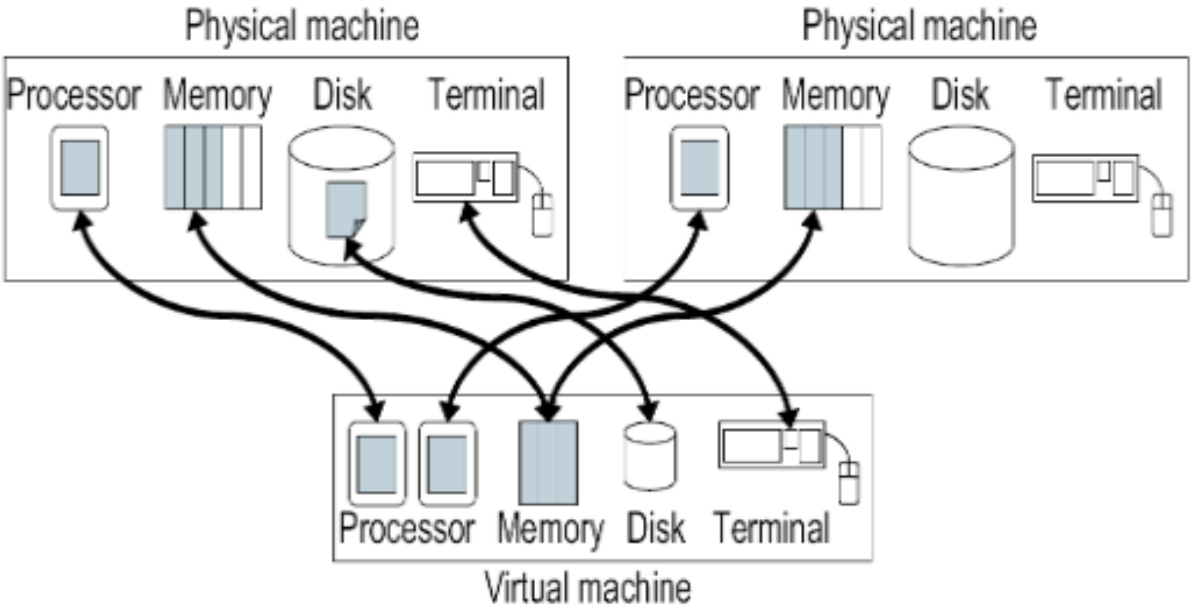


Рисунок 3.8 – Mapping між віртуальною машиною та фізичною машиною

Більша частина поточного інтересу до віртуалізації пов'язана з віртуальними серверами, частково тому, що віртуалізація серверів може призвести до значної економії коштів. Вираз VM відноситься до комп'ютерного програмного забезпечення, яке, як і фізичний комп'ютер, запускає операційну систему та додатки. Операційна система на віртуальній машині називається гостьовою операційною системою. Крім того, існує рівень управління, який називається Монітор віртуальних машин або Менеджер (VMM), який створює та керує всіма віртуальними машинами у віртуальному середовищі.

## 3.6 Впровадження DVMM для інтеграції розподілених ресурсів віртуального обчислювального середовища

Популярні монітори VM, такі як Xen та VMware, призначені в основному для віртуалізації на одному фізичному вузлі. У своїй роботі ми впровадили монітор VM для розподіленого міжвузла та для інтеграції обчислювальних системних ресурсів.

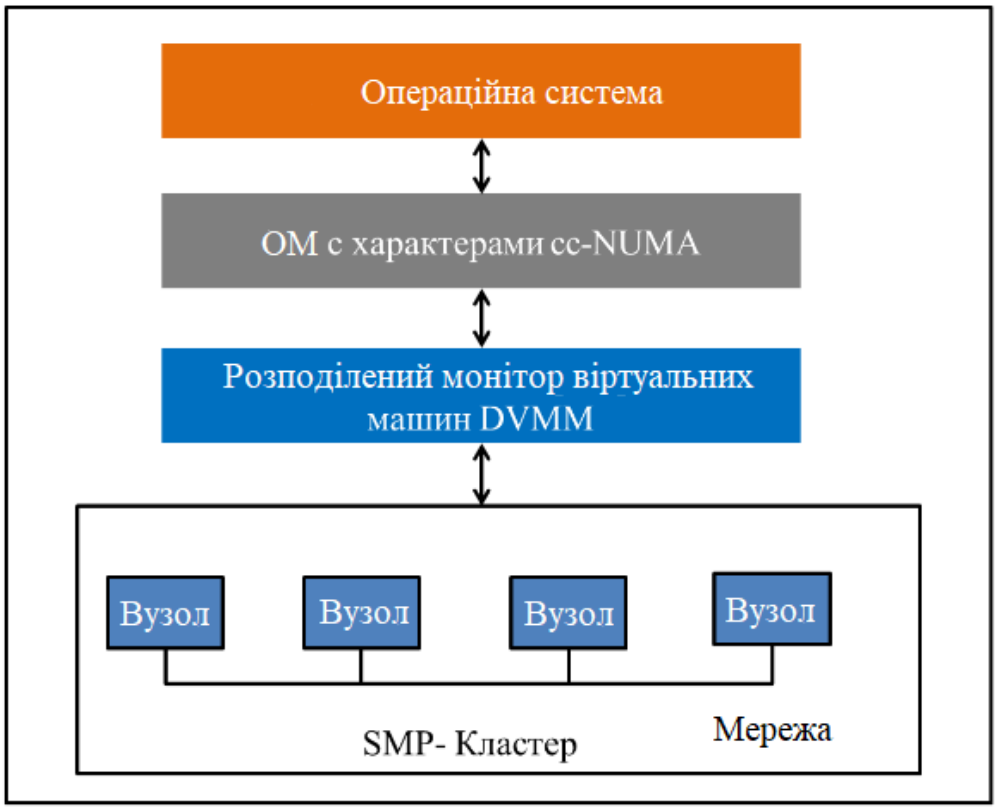


Рисунок 3.9 – Реалізація DVMM для інтеграції розподілених ресурсів віртуального операційного оточення

Ми створили DVM над кластером SMP на основі апаратної віртуалізації. DVM - це заміна всіх розподілених VMM у кожному вузлі. Усі VMM симетричні. Основною метою DVM є приховування розподілених апаратних атрибутів для надання SI в кластері SMP і підтримки єдиної операційної системи для прозорої роботи в кластері.

## 3.7 Розроблене комп'ютерне середовище

Консолідація та спільне використання послуг, розроблених як економічно ефективний інструмент, надають ці послуги через мережу, за потреби. Ця здатність залежить від прогресу в стандартизації, оптимізації, орієнтації на послуги та віртуалізації. Зокрема, віртуалізація підтримує динамічні пули ресурсів, такі як сервери та системи зберігання даних. Існує два типи консолідації: фізична та логічна.

При фізичній консолідації сервери фізично переміщуються в єдиний інформаційний центр, але їх представлення в комп'ютерній мережі може залишаються незмінними, що означає, що декілька мережевих послуг можуть бути призначені одному фізичному серверу.

Логічна консолідація - це інтеграція серверних обчислювальних ресурсів без фізичного переміщення їх до єдиного мережевого ресурсу. Зараз починають з’являтися необхідні рішення. Зокрема, вони дозволяють управляти мережею з декількох фізичних серверів як єдиний пристрій. Ми формулюємо набір вимог, яким повинна відповідати процедура консолідації, щоб її використання було економічно виправданим: доступність користувацьких програм після консолідації не повинна зміни; робота користувачів з додатками не повинна бути складною; інформаційна безпека не повинна негативно впливати; нам потрібно забезпечити значне зменшення загальних витрат на володіння інформаційною системою.

Найпростіший приклад застосування консолідації - об’єднання двох або більше компаній з різними інформаційними системами. У цьому випадку консолідація дозволяє заощадити гроші, усуваючи повторювані елементи та ефективно використовуючи наявні ресурси. Інший приклад - консолідація галузевих інформаційних систем.

Тут важлива логічна консолідація, яка дозволить центральному офісу контролювати роботу інформаційних систем філій. При міграції ресурсоємних додатків на централізовані сервери можна зменшити вартість

робочого місця, використовуючи дешевші комп’ютери, які працюють із суворо фіксованим набір програм - як правило, лише браузер. Поєднання серверів в одному центрі обробки даних дозволяє посилити їх захист і підвищити надійність розрахунків. Це стосується обох фізична безпека (організація доступу до апаратного забезпечення) та мережева безпека (віддалений доступ).

Те саме стосується надійності, оскільки легше забезпечити оптимальні умови навколишнього середовища, надійне живлення, постійне резервне копіювання та швидке планове обслуговування в централізованому обчислювальному центрі, ніж у розподіленій системі. У кожному випадку ви можете використовувати власні методи та стратегії для консолідації, що дозволило вам її оптимізувати. Однак існує універсальне рішення - віртуалізація, яке загалом дозволяє змінювати фізичну та логічну конфігурацію інформаційної системи. Щоб досягти ефективної консолідації, важливо зрозуміти, які ресурси потрібні для запуску програми з часом. Ці дані включають характеристики використання програми та пікові вимоги до ресурсів. Завдяки цій інформації віртуальні машини, які запускають програми, можуть запускатися за допомогою фізичних обчислень та ресурсів зберігання.

Неправильний запуск віртуальних машин може призвести до проблем із поточною продуктивністю, коли вимоги до ресурсів програми суттєво змінюються з часом.

### 3.7.1 Модель для запуску додатків у обчислювальному середовищі

Для того, щоб мати можливість віддалено запускати програми на розвиненому обчислювальному середовищі, на ньому слід виконати спеціальні процеси GRAM (Grid Resource Allocation Manager). За допомогою нього ви можете додавати, відстежувати та скасовувати завдання в системі. Користувацькі програми формують запити до GRAM спеціальною мовою

RSL (мова специфікації ресурсу). GRAM включає двох демонів:

- Globus-gatekeeper -daemon та бібліотеку GSI, що використовуються для захисту передачі даних та координації служб;

- Globus-job-manager -демон, який реалізує планування завдань та функціональність для передачі даних або файлів до управління ресурсами.


Рисунок 3.10 – Розроблена віртуальна обчислювальна система

Для запуску програми в системі були інтегровані UGE та GRAM, що реалізували інтерфейс програми DRMAA (API розподіленого управління ресурсами), що дозволяє запускати програму в системі за допомогою скриптів. GridFTP - це послуга для швидкої та ефективної передачі даних за протоколом gsiftp (особливо у великих обсягах). У системі UGE існує спеціальна структура даних - паралельне середовище, яке дозволяє враховувати особливості програм, що використовують паралельне

програмування технологій. Для вирішення паралельних завдань UGE та Mosix були інтегровані з додатковим пакетом скриптів та бібліотекою Mpich2.

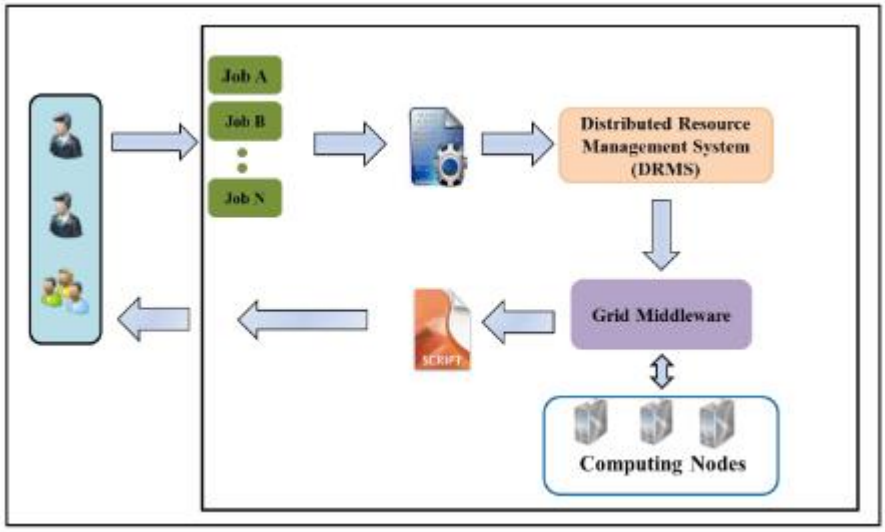


Рисунок 3.11 – Модель запуску завдань з інтерфейсом DRMAA

Для успішного виконання цих завдань ми проаналізували та розробили метод управління ресурсами в розподіленому обчислювальному середовищі для запуску конкретних ресурсомістких програм. Використовуються різні програмні продукти, такі як Univa Grid Engine, Mosix та Globus Toolkit. Таким чином, віртуальні машини можуть вільно спілкуватися між собою через віртуальну мережу та автоматично налаштовувати мережу. У цьому випадку віртуальна машина може розподіляти (навіть динамічно) між вузлами.

В даний час широко використовуються тестові програми, взяті з різних предметних областей, що представляють як модельні, так і реальні промислові програми. Ці тести дозволяють оцінити ефективність роботи комп'ютера в реальних завданнях і отримати найбільш повний аналіз того, як працює комп'ютер з конкретним додатком.

Найпоширенішими тестами, побудованими за цим принципом, є набір з

24 циклів Лівермора (ядра Лівермора Фортрана, LFK) та пакет паралельних контрольних тестів NAS (NPB), який включає дві групи тестів, що відображають різні аспекти реальних обчислювальних програм динаміки рідини. Тести NAS є альтернативою Linpack, оскільки вони відносно прості і водночас містять значно більше розрахунків, ніж, наприклад, Linpack або LFK.

Однак при всьому різноманітті тестові програми не можуть дати повного зображення того, як комп'ютер працює в різних режимах. Linpack - це програма, призначена для вирішення системи лінійних алгебраїчних керувань з щільною матрицею з вибором основного елемента за лінією. Для того, щоб оцінити ефективність обчислювальної системи щодо реальних завдань, був розроблений фіксований набір тестів.

Однак LINPACK має суттєвий недолік: програма паралелізована, тому неможливо оцінити ефективність комунікаційної складової суперкомп'ютера. LINPACK був обраний, оскільки він широко використовується, і номери запуску доступні майже для всіх відповідних систем. Тест складається з ряду простих синтетичних задач: ядра (орієнтири ядра) та псевдо-додатки (тести додатків), що імітують розрахунки реальних проблем (зокрема, в області обчислювальної динаміки рідини).

Для кожного ядра тести NAS визначають п'ять класів зростаючих навантажень, які називаються S, W, A, B та C. Клас S зручний для тестування і відповідає системам з максимум 4 процесорами класу W і підходить для робочих станцій. Клас W підходить для однопроцесорних систем, тоді як клас A підходить для систем з до 32 процесорами. Класи B і C зручні для багатопроцесорних систем. Клас B для систем з 32-128 процесорами та Клас C для систем з до 256 процесорами.

За термінологією NPB ядра та програми можуть виконувати обчислення за певними класами проблем: "Зразок коду", "Клас A", "Клас B", "Клас C", "Клас D". У NPB клас відноситься до розмірності основних масивів даних, що використовуються в тесті. Іншими словами, клас A - це

малі матриці, B - великий, C - дуже великий, D - величезний. Наприклад, для тесту на розкладання LU це буде розмірність вихідної матриці: $12^3$, $64^3$, $102^3$, $408^3$ для кожного з класів, перерахованих вище, відповідно.

Алгоритми додатків використовують ядра, описані вище в тій чи іншій формі, і, зрештою, зводиться до вирішення спеціальної форми SLAE (Система лінійних алгебраїчних рівнянь) (а також переважної більшості обчислювальних задач). Основна частина комп'ютерного часу в таких задачах витрачається саме на розв'язання SLAE (Система лінійних алгебраїчних рівнянь. Тому додатки можна охарактеризувати як ітераційні методи розв'язання SLAE. Існує три таких програми: LU, SP, BT. Тест порівняльних показників NAS набір для модельних завдань включає такі модулі: 1.LU (LU Solver).

Тест виконує обчислення, пов'язані з певним класом алгоритмів (INS3D-LU за класифікацією центру Еймса НАСА), в якому система рівнянь з вирішено рівномірно розріджений блок трикутної матриці 5x5. Цей тест вимірює затримку мережі та пропускну здатність кешу інструкцій. 2. SP (Скалярний Пентадіагональ). Тест вирішує кілька незалежних систем скалярних рівнянь - пендіагональні матриці з переважанням недіагональних членів.

Цей тест вимірює пропускну здатність пам'яті. 3. VT (Блок Тридіагональ). Рішенням для серії незалежних систем рівнянь є блок 5 × 5 тридіагональних матриць з переважним е недіагональних елементів. Цей тест вимірює пропускну здатність мережевих і кеш-команд. Алгоритми SP (від Scalar Pentadiagonal) та BT (від Block Tridiagonal) подібні: вирішення трьох не пов'язаних між собою систем рівнянь (у напрямках x, y та z) за допомогою мульти -метод схеми розділення.

Різниця між додатками полягає в структурі матриці: для SP це п'ятидіагональна матриця, а для BT це блокова тридіагональна матриця з розміром блоку 5x5.

Цей метод добре паралелізований і виконується для оцінки

продуктивності системи з плаваючою точкою та забезпечує оптимальне навантаження на мережу, але вимагає, щоб кількість вузлів у кластері було цілим квадратним числом. Тест SP більш чутливий до затримки мережі і проводиться для оцінки ефективності міжвузлових з'єднань. Додаток LU вирішує систему.

У нашій роботі ми проаналізували програмування MPI з MOSIX та без нього, щоб оцінити можливості паралельних обчислень у розвиненому середовищі. Ці тести проводились під контролем операційного середовища MOSIX, з використанням і без використання схеми міграції процесів. У нашому експерименті кластер MOSIX був створений у віртуальному середовищі. Там були встановлені програми MPI та MOSIX, і було запущено додаток для визначення часу затримки під час синхронізації процесів MPI. У цьому прикладі запускається тест зв'язку MPI (час затримки повідомлення). MPI-0 надсилає 1-байтове повідомлення MPI-1, витрачаючи час на очікування відповіді між ними.

Після цього для кожного повторення виконується синхронізація, а в кінці розраховується середній час очікування. Ці випробування проводились під контролем робочого середовища MOSIX, з використанням схеми проактивного переходу та без неї. Затримка мережі - це час, необхідний для того, щоб щось, надіслане від вихідного хоста, дійшло до хоста призначення. Час затримки туди-назад - це скільки часу потрібно для надсилання запиту від джерела до пункту призначення та для відповіді на повернення до початкового джерела. В принципі, затримка в кожному напрямку плюс час обробки [9]. На рисунку 3.12 показані результати тестування часу затримки в дорозі туди і назад.

Після всіх конфігурацій ми проаналізували ефективність нашого віртуального розподіленого середовища. У системі ми управляємо різнорідними ресурсами, створюємо віртуальні обчислювальні кластери під контролем гіпервізора та об’єднуємо обчислювальні ресурси в єдину обчислювальну систему.

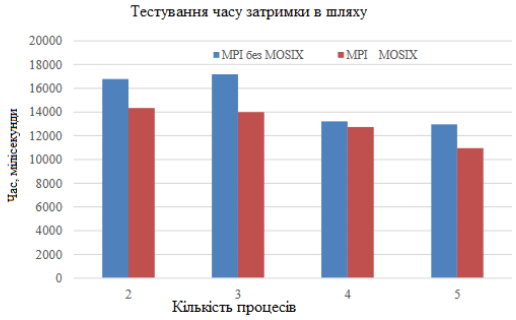


Рисунок 3.12 – Тестування часу затримки туди-назад із MOSIX та без нього

Наша система працює з реальними завданнями. Ми проаналізували тестові випадки, такі як тести LU з пакету NAS, розрахунки великого пакета OpenFOAM - відкритого пакету для чисельного моделювання задач у механіці континууму та наукового додатку CRYSTAL, який є програмним інструментом для досліджень у галузі хімії та фізики, щоб перевірити можливості розвиненого середовища.

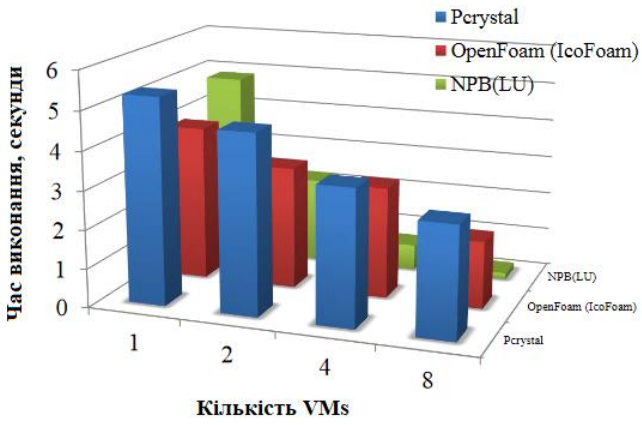


Рисунок 3.13 – Результати тесту продуктивності запуску програми нього

Перш ніж ми запустили ці програми, ми також проаналізували здатність нашої системи до паралельного програмування.

# ВИСНОВКИ

Був розроблений новий підхід до побудови віртуального оточення, що забезпечує збільшення обчислювальної потужності за рахунок динамічного балансування навантаження, управління ресурсами і міграцією процесів на вузлах обчислювального середовища.

Розроблен новий метод інтеграції різнорівневих програм на базі проміжного програмного забезпечення, що надається платформою для ефективного управління ресурсами обчислювальної системи.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Публікація

2. A. Itzkovitz and A. Schuster. Distributed shared memory: Bridging the granularity gap, in Proceedings of the First ACM Workshop on Software Distributed Shared Memory (WSDSM, 1999.)

3. A. Snavely and D. M. Tullsen. Symbiotic job scheduling for a simultaneous multithreading processor. In Proceedings of the Ninth International Conference on Architectural Support for ProgrammingLanguages and Operating Systems, pages 234–244, Nov. 2000.

4. Al-Kiswany, S., Subhraveti, D., Sarkar, P., and Ripeanu, M. VMFlock: Virtualmachine co-migration for the cloud. In HPDC'11 (San Jose, USA, June 2011).

5. Amnon Barak, Amnon Shiloh. MOSIX2 Cluster Multi-Cluster Cloud Management: User's and Administrator's Guides and Manuals. June 2009.

6. Amnon Barak, Avner Braverman, Ilia Gilderman, Oren Laden. Performance of PVM with the MOSIX Preemptive Process Migration Scheme. The Hebrew University of Jerusalem, Israel.

7. Amnon Barak, Oren La'adan. The MOSIX multicomputer operating system for high performance cluster computing. The Hebrew University of Jerusalem, Israel.

8. Andrew J. Younge, Robert Henschel, James T. Brown, Gregor von Laszewski, Judy Qiu, Geoffrey C. Fox, Pervasive Technology Institute, Indiana University. Analysis of Virtualization Technologies for High Performance Computing Environments.

9. Arun Babu Nagarajan, Frank Mueller, Christian Engelmann, Stephen L. Scott: Proactive fault tolerance for HPC with Xen virtualization. ICS 2007: 23-32.

10. Bog BOS: Sun Grid Engine [http://www.bog.pp.ru/work/sge.html]

11. Borko Furht and Armando J. Escalante. Handbook of Cloud Computing: Part1 Technologies and Systems, page: 3-19.

12. А.В.Богданов, Е.Н.Станкова: Распределенные Linuxкластеры как основы сетей науки и образования будущего. // Труды IX Всероссийской научно-методической конференции Телематика-2002.

13. John L. Hennessy, David A. Patterson: "Computer Architecture: A Quantitative Approach, 4thEdition", Morgan Kaufmann Publishers, ISBN 10: 0-12-370490-1, ISBN 13: 978-0-12-370490-0, 2007.

14. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing onlarge clusters. // OSDI’04: Sixth Symposium on Operating System Design and Implementation, December 2004.

15. А.В.Богданов, В.В.Корхов, В.В.Мареев, Е.Н.Станкова: Архитектуры и топологии многопроцессорных вычислительных систем. «Интернет-Университет ИнформационныхТехнологий»,2004.

16. А.В.Богданов, В.В.Корхов, В.В.Мареев, Е.Н.Станкова; Издательство: Интернет-университет информационных технологий - ИНТУИТ.ру, Архитектуры итопологии многопроцессорных вычислительных систем. (учебник), Серия: Основы информационных технологий , 2004.

17. А.В.Богданов, М.И.Павлова, Е.Н.Станкова, Л.С.Юденич: Высокопроизводительные вычислительные алгоритмы (учебное пособие)// URL: http://www.csa.ru/old/analitik/distant/q_start.html.

18. А.И. Аветисян, О.И. Самоваров, Д.А. Грушин: Архитектура и системное программное обеспечение вычислительных кластерных систем// Институт системного программирования Российской академии наук, Москва, 2005.

19. Антонов А. С: Параллельное программирование с использованием технологии MPI: Учебное пособие. -М: Изд-во МГУ, 2004. с-71.

20. Антонов А. С: Параллельное программирование с использованием технологии OpenMP: Учебное пособие. -М.: Изд-во МГУ, 2009. с-77.

21. Бажанов С.Е., Кутепов В.П., Шестаков Д.А: Разработка и

реализация системы функционального параллельного программирования на вычислительных сред.

22. Данилов В.В: Архитектура процессоров SunUltraSparcT1 и T2 (Niagara). // Московский инженерно-физический институт. 2007 г. стр. 8.

23. Деревянко А.С., Солощук М.Н. –Харьков: Технологии и средства консолидации информации. // Учебное пособие. НТУ "ХПИ", 2008.

24. Дмитрий Кузьмин, Федор Казаков, Денис Привалихин, Александр Легалов: На пути к переносимым параллельным программам. // URL: http://citforum.ru/programming/theory/parall_prog/

25. Е.В. Адуцкевич: “Организация обмена данными на параллельных компьютерах с распределенной памятью”, Институт математики НАН Беларуси, Минск (Беларусь), 2005