Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Метод об'єднання гетерогенних обчислювальних ресурсів у розподіленому віртуальному середовищі

(тема)

Виконав

студент II курсу, групи СПм-22-2

Мартинцов А.Ф.
(прізвище, ініціали)

Спеціальність

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

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

Освітня програма

Системне програмування
(повна назва освітньої програми)

Керівник: зав. каф. Коваленко А.А.
(посада, прізвище, ініціали)

Допускається до захисту

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

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

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

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

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Мартинцову Артему Федоровичу
(прізвище, ім’я, по батькові)

1. Тема роботи Метод об'єднання гетерогенних обчислювальних ресурсів у розподіленому віртуальному середовищі

затверджена наказом по університету від “ 06 ” листопада 2023 р. № 1299 Ст

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

3. Вхідні дані до роботи 1) Globus Toolkit; 2) операційна система –Windows 7, 8, 10 або 11 3) Parallel Virtual Machine; 4) система управління кластерами MOSIX

4. Перелік питань, що потрібно опрацювати у роботі

1) Аналіз особливостей розподілених обчислень
2) Розробка математичної моделі розподілених обчислювальних середовищ.
3) Реалізація розробленого гетерогенного обчислювального середовища
4) Тестування розробленого гетерогенного обчислювального середовища.
5) Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 14

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	Аналіз предметної області	07.11.23-13.11.23	
2	Вибір та обгрунтування методики дослідження	14.11.23-20.11.23	
3	Вибір інструментальних засобів	21.11.23-23.11.23	
4	Розробка математичної моделі	24.11.23-06.12.23	
5	Реалізація розробленого гетерогенного	07.12.23-23.12.23	
	обчислювального середовища		
6	Оформлення матеріалів кваліфікаційної роботи	26.12.23-02.01.24	
7	Подання кваліфікаційної роботи керівникові та	03.01.24-06.01.24	
	її попередній захист		
8	Подання кваліфікаційної роботи на рецензу-	09.01.24-12.01.24	
	вання		

Дата видачі завдання 06 листопада 2024 р.

Студент _____
(підпис)

Керівник роботи _____ зав. каф. Коваленко А.А.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 73 с., 19 рис., 2 табл., 1 дод., 16 джерел.

РОЗПОДІЛЕНІ ОБЧИСЛЕННЯ, ГРИД-ТЕХНОЛОГІЇ, ХМАРНІ ОБЧИСЛЕННЯ.

Метою кваліфікаційної роботи є збільшення обчислювальної потужності при об'єднанні гетерогенних обчислювальних ресурсів у єдиний обчислювальний комплекс для ефективного управління обчислювальними ресурсами та розробка методу запуску додатків у розподіленому віртуальному середовищі.

У ході виконання кваліфікаційної роботи було розглянуто Продуктивність обчислювальних середовищ визначається їх доступністю, переважно оцінюваної системами балансування навантаження, системами управління ресурсами, виконання робочої навантаження та обробки даних на вирішення величезної кількості завдань. Щоб задовольнити ці вимоги, нам потрібне потужне обчислювальне середовище з надійними системами балансування навантаження, системою загальної пам'яті і надійною системою обробки даних з величезним обсягом даних.

ABSTRACT

Master’s thesis: 73 pages, 19 figures, 2 tables, 2 appendices, 16 sources.

DISTRIBUTED COMPUTING, GRID TECHNOLOGIES, CLOUD COMPUTING.

The purpose of the work is to increase computing power by combining heterogeneous computing resources into a single computing complex for efficient management of computing resources and developing a method for launching applications in a distributed virtual environment.

The productivity of computing environments is determined by their availability, mainly assessed by load balancing systems, resource management systems, workload execution and data processing for solving a huge number of tasks. To meet these requirements, we need a powerful computing environment with reliable load balancing systems, a shared memory system, and a reliable data processing system with a huge amount of data.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ОС – операційно ї системи

ВМ – віртуальна машина

DVMM – монітор розподіленої віртуальної машини

FTP – протокол передачі файлів

IAAS – інфраструктура як послуга

MFS – MOSIX файлова система

MPI – інтерфейс передачі повідомлень

MPP – масова паралельна обробка даних

OPENCL – open computing language

OPENMP – open multi processing

PAAS – платформа як послуга

SMP – симетрична багатопроцесорна обробка

VFS – віртуальна файлова система

VO – віртуальна організація

ВСТУП

Продуктивність обчислювальних середовищ визначається їх доступністю, переважно оцінюваної системами балансування навантаження, системами управління ресурсами, виконання робочої навантаження та обробки даних на вирішення величезної кількості завдань. Щоб задовольнити ці вимоги, нам потрібне потужне обчислювальне середовище з надійними системами балансування навантаження, системою загальної пам'яті і надійною системою обробки даних з величезним обсягом даних. Так, кілька років тому використання віртуальних машин надало великі можливості для керування ресурсом за допомогою динамічної міграції віртуальних машин. За допомогою технологій віртуалізації фізичний сервер можна розділити на кілька ізольованих середовищ виконання, розгорнувши шар поверх апаратних ресурсів чи операційної системи (ОС). Серверні середовища виконання та віртуальні машини (ВМ) можуть працювати одна на одній, не перериваючи одна одну. Кожна віртуальна машина має свою власну ОС та програми.

На самому початку технології віртуалізації були широко використані з низки причин. У хмарних обчисленнях великі компанії можуть зібрати разом свої запасні апаратні ресурси і здавати їх в оренду клієнтам на платній основі, а користувачі можуть швидко почати працювати на віртуальній машині, не витрачаючи величезні витрати на купівлю та обслуговування обладнання. Оскільки всі користувачі вибирають хмарні центри обробки даних для зберігання своїх 7 програм, ефективне управління віртуальними машинами стало дуже актуальним у центрі обробки даних. Згідно з останніми експериментами, комбінація технології віртуалізації та SSI дозволяє нам отримувати кращі рішення та досягати гнучкості та простоти використання ресурсів. Віртуальна машина додає рівень гнучкого управління між обладнанням та додатками, у той час як SSI забезпечує абстракцію

розподілених ресурсів. Одночасне використання двох технологій дозволяє нам збільшити загальне використання ресурсів платформи, продуктивність та ефективність кластерних програм. Грід та хмарні обчислення – це сучасні обчислювальні технології, що складаються з різних ресурсів, що підтримуються різними організаціями. Управління такими системами надто складне. Тому дослідники використовують різні методології та інструменти для аналізу, оцінки та прогнозування запропонованих ними алгоритмів та методів. Більшість опублікованих досліджень було оцінено на основі реальних інфраструктур Грід-середовища або з використанням інструментів цих технологій. Ці розподілені системи мають велику обчислювальну потужність, що дозволяє вирішувати найскладніші завдання, такі як прогнозування погоди та моделювання землетрусів. Це надзвичайно складні системи територіально розподілених та незалежно керованих ресурсів. Деякі з багатьох аспектів, які слід розглянути, включають: управління ресурсами, гетерогенність, стійкість до відмов, продуктивність мережі, безпеку, адаптивність, масштабованість і прозорість.

Метою роботи є збільшення обчислювальної потужності при об'єднанні гетерогенних обчислювальних ресурсів у єдиний обчислювальний комплекс для ефективного управління обчислювальними ресурсами та розробка методу запуску додатків у розподіленому віртуальному середовищі.

Для досягнення поставленої мети необхідне вирішення таких завдань:

1) розробка нового методу побудови віртуального операційного оточення, що забезпечує підвищення обчислювальних потужностей;

2) дослідження системи управління розподіленими ресурсами у віртуальному обчислювальному середовищі;

3) інтеграція комплексу програм та технологій для оптимізації віртуальної гетерогенної системи та запуск спеціалізованих завдань для аналізу продуктивності обчислювальних середовищ.

1 ДОСЛІДЖЕННЯ ОСОБЛИВОСТЕЙ ГЕТЕРОГЕННИХ ОБЧИСЛЮВАЛЬНИХ РЕСУРСІВ

1.1 Особливості розподілених обчислень

Розподілені обчислення – це підхід до вирішення ресурсомістких завдань, що використовує кілька комп'ютерів, підключених до мережі. Розподілена обчислювальна система – це сукупність незалежних комп'ютерів, з'єднаних каналами зв'язку, які з погляду користувача деякого програмного забезпечення виглядають єдиним цілим. Вони є окремим випадком паралельних обчислень. Паралельна програма включає кілька процесів, що працюють разом для виконання конкретної задачі. Спільна робота кількох процесів здійснюється за допомогою їхньої взаємодії, при цьому потрібна синхронізація (наприклад, взаємний виняток або умовна синхронізація). Сфера застосування паралельних обчислень постійно розширюється, охоплюючи нові сфери. Застосування розподілених систем стає практично повсюдним як для наукових розрахунків, а й у повсякденних завдань бізнесу (досить пригадати високонавантажені web-сервера) [6]. Для запуску паралельних додатків використовуються багатопроцесорні системи, окремим випадком яких і є розподілені системи. Багатопроцесорні системи існують у різних конфігураціях. Найбільш поширеними типами таких систем є:

- системи високопродуктивних обчислень – системи, призначені для паралельних обчислень, багатопроцесорні обчислювальні системи (БОС) для високопродуктивних обчислень зазвичай збираються з безлічі комп'ютерів, тому розробка таких систем являє собою складний процес, що вимагає постійної координації таких питань, як одночасне управління великою кількістю комп'ютерів, загальний доступ до ресурсів тощо;
- системи високої надійності – це комплекс програмних та технічних засобів, призначених для забезпечення безперервної безвідмовної роботи

систем;

- багатопотокові 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], оптимізація конфігурації та підтримка багатопроцесорної системи – досить складні завдання.

Багато інструментів було створено для паралельного програмування у вигляді спеціальних мов програмування та API:

- POSIX Threads – стандарт POSIX реалізації потоків (ниток) виконання, що визначає API для створення та управління ними;

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

- PVM (Parallel Virtual Machine) – це загальнодоступний програмний пакет, що дозволяє об'єднувати різнорідний набір комп'ютерів у загальний обчислювальний ресурс;

- OpenMP (Open Multi-Processing) – відкритий стандарт для розпаралелювання програм мовами C, C++ та Фортран;

- OpenCL (Open Computing Language) – стандарт для написання паралельних програм, що дозволяє використовувати не лише CPU, а й GPU, а також спеціалізовані прискорювачі. Крім того, існують спеціалізовані системи, що дозволяють об'єднувати комп'ютери в єдиний обчислювальний комплекс та здійснювати балансування завдань:

- MOSIX – це система управління кластерами, яка об'єднує їх у єдину систему (Single-System). Це операційна система, що дозволяє керувати кластером як єдиною системою;

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

Почнемо з того, що буває кілька видів паралелізму. Найпростіший вид – це паралелізм лише на рівні бітів. Ця форма паралелізму полягає в

збільшенні розміру машинного слова. Як відомо, 4-бітні процесори були замінені 8-бітними, а ці 16-, 32-, і, нарешті, 64-бітними. Наступний вид паралелізму має на увазі наявність конвеєра команд і має назву паралелізму на рівні інструкцій. Сучасні процесори мають багатоступінчастий конвеєр команд. Кожному щаблі конвеєра відповідає певна дія, яку виконує процесор на цьому етапі. Класичний приклад процесора з конвеєром – це RISC процесор з 5-ма ступенями: вибірка інструкції з пам'яті, декодування інструкції, виконання інструкції, доступ до пам'яті, запис результату в регістри.

Паралелізм даних передбачає виконання однієї й тієї операції над масивом даних. Різні фрагменти такого масиву обробляються на векторному процесорі чи різних процесорах паралельної машини. Розподіл даних між процесорами, задіяними у програмі. У цьому випадку векторизація або розпаралелювання найчастіше виконується вже на стадії компіляції-переведення вихідного коду програми машинні команди. Роль програміста в цьому випадку зазвичай обмежується налаштуванням параметрів векторної або паралельної оптимізації компілятора, директив паралельної компіляції та використанням спеціалізованих мов для паралельних обчислень. Паралелізм завдань передбачає виконання окремого завдання кожному з обчислювальних вузлів [3].

1.2 Архітектури обчислювальних систем

У 1966 році Майкл Флінн запропонував дуже зручну класифікацію обчислювальних систем. Він був заснований на концепції потоку, який сприймається як послідовність елементів, команд чи даних, оброблюваних процесором [3, 8]. Відповідна система класифікації заснована на підрахунку кількості потоків інструкцій та даних та описує чотири архітектурні класи.

SISD (Single Instruction Stream / Single Data Stream) – єдиний потік інструкцій та єдиний потік даних. Цей клас включає послідовні комп'ютерні

системи, що мають один центральний процесор, здатний обробляти тільки один потік інструкцій, що послідовно виконуються. В даний час практично всі високопродуктивні системи мають більше одного центрального процесора, але кожен із них виконує незв'язані потоки команд, що робить такі системи комплексами систем SISD, що працюють на різних просторах даних [4, 8]. Конвеєризація може бути використана для збільшення швидкості обробки команд та швидкості виконання арифметичних операцій.

MISD (multiple instruction stream/single data stream) – кілька потоків інструкцій та один потік даних. Теоретично, у цьому типі машин набір інструкцій має виконуватися по одному потоку даних. Але досі не було створено жодної реальної машини, яка належить до цього класу. Як аналог функціонування такої системи є можливим розглянути діяльність банку. З кожного терміналу відправляються різні команди, але вони мають справу з однією і тією самою загальною базою даних.

KSIMD (Single instruction stream / multiple data stream) – один потік інструкцій та кілька потоків даних. Ці системи зазвичай мають велику кількість процесорів, від 1024 до 16384, які можуть виконувати одну і ту ж інструкцію для різних даних жорсткої конфігурації. Одна команда виконується паралельно з багатьма елементами даних. Ще одним підкласом систем SIMD є векторні комп'ютери. Вони маніпулюють масивами вхідних даних за допомогою спеціально розроблених векторних процесорів. Слід зазначити, що технологія GPGPU належить саме до цієї категорії.

MIMD (multiple instruction stream / multiple data stream) – множинний потік інструкцій та множинний потік даних. Цей тип машини виконує кілька потоків команд на різних потоках даних паралельно. На відміну від багатопроцесорних SISD-машин, згаданих вище, команди та дані в цьому випадку пов'язані, оскільки вони є різними частинами однієї і тієї ж задачі. Системи MIMD можуть виконувати безліч підзавдань паралельно, щоб скоротити час, необхідний виконання основного завдання. Велика різноманітність систем, які у цей клас, робить класифікацію Флінна

недостатньо адекватної. Дійсно, дана класифікація недостатньо детальна, тому необхідно вводити докладнішу класифікацію, пов'язану з різними архітектурами та обладнанням. Розглянемо кілька конкретних архітектур обчислювальних систем.

SMP (Symmetric Multiprocessing) – симетрична багатопроцесорна архітектура. Головною особливістю систем з архітектурою SMP є наявність загальної фізичної пам'яті, що поділяється всіма процесорами. Пам'ять використовується, зокрема, передачі повідомлень між процесорами, у своїй всі обчислювальні пристрої за зверненні до неї мають рівні правничий та однакову адресацію всім осередків пам'яті. Ось чому архітектура SMP називається симетричною. Остання обставина дозволяє дуже ефективно обмінюватись даними з іншими обчислювальними пристроями.

До переваг систем з такою архітектурою потрібно насамперед віднести легкість їх програмування. Дійсно, архітектура SMP не накладає обмежень на модель програмування, що використовується під час створення програми: зазвичай використовується модель паралельної гілки, де всі процесори працюють незалежно друг від друга. Проте можна реалізувати моделі, які використовують міжпроцесорний обмін. Використання спільної пам'яті збільшує швидкість обміну, і програма також має доступ до всього обсягу пам'яті відразу. Слід зазначити, що для таких систем є досить ефективні засоби автоматичного розпаралелювання. Також однією з переваг SMP-систем є їхня простота в експлуатації. Головний недолік – погана масштабованість. Причиною поганої масштабованості є той факт, що шина в даний час здатна обробляти тільки одну транзакцію, що викликає проблеми вирішення конфліктів, коли кілька процесорів звертаються до тих самих областей загальної фізичної пам'яті одночасно.

Обчислювальні елементи починають зміщуватися один від одного. Коли виникає такий конфлікт, залежить від швидкості зв'язку та кількості обчислювальних елементів [2, 9].

MPP (Massive Parallel Processing) – у цьому випадку система будується

з окремих модулів, що містять процесор, локальну оперативну пам'ять, мережеві адаптери, інколи жорсткі диски та/або інші пристрої введення. Головна особливість цієї архітектури у тому, що пам'ять фізично розділена. Насправді, такі модулі є повнофункціональними комп'ютерами. Головною перевагою таких систем є масштабованість. До недоліків можна віднести знижену швидкість міжпроцесорного обміну (оскільки в цьому випадку відсутня загальна оперативна пам'ять, в результаті-потрібна спеціальна методика програмування для реалізації обміну повідомленнями між процесорами), обмежений обсяг оперативної пам'яті (оскільки тепер кожен процесор має свій власний банк оперативної пам'яті, зазвичай не дуже великий), а також високу ціну програмного забезпечення для таких систем (потрібно докласти зусиль, щоб досягти максимальної ефективності використання ресурсів) [9].

NUMA (Non-Uniform Memory Access) – архітектура з нерівномірним доступом до пам'яті. Гібридна архітектура поєднує в собі переваги систем із загальною пам'яттю та відносну дешевизну окремих систем пам'яті. Суть цієї архітектури полягає в особливій організації пам'яті, а саме: пам'ять фізично розподілена по різних частинах системи, але логічно вона є загальною, тому користувач бачить єдиний адресний простір. Система побудована з однорідних базових модулів (плат), які з невеликої кількості процесорів і блоку пам'яті. Модулі поєднані за допомогою високошвидкісного комутатора. Підтримується єдиний адресний простір, апаратний доступ до віддаленої пам'яті, тобто до пам'яті інших модулів. По суті, архітектура NUMA є MPP архітектурою, де як окремі обчислювальні елементи беруться SMP вузли. Стів Воллох запропонував ідею цієї архітектури.

CC-NUMA (Cache Coherent Non-Uniform Memory Access) – архітектура з неоднорідним доступом до пам'яті із забезпеченням когерентності кешів. Поліпшення NUMA-систем, запропоноване Сеймуром Креєм. Поняття когерентності кешів визначає той факт, що всі центральні процесори набувають однакових значень тих самих змінних у будь-який момент години.

Справді, оскільки кеш-пам'ять належить окремому комп'ютеру, а чи не всієї багатопроцесорної системи в цілому, дані, які у кеш одного комп'ютера, можуть бути недоступні іншому. Щоб цього уникнути, слід провести синхронізацію інформації, що зберігається у кеш-пам'яті процесорів.

PVP (Parallel Vector Processing) – паралельна архітектура із векторними процесорами. Ключовою особливістю таких систем є векторно-конвеєрні процесори, що забезпечують команди для однотипної обробки векторів незалежних даних, які ефективно виконуються на конвеєрних функціональних пристроях. Зазвичай кілька таких процесорів працюють із загальною оперативною пам'яттю на одному вузлі. Декілька вузлів можуть бути об'єднані за допомогою комутатора. Створення програм для таких систем передбачає векторизацію циклів та їх розпаралелювання [1].

Кластерна архітектура – набір робочих станцій загального призначення, об'єднаних у мережу. Такі системи є найдешевшими, тому що можуть збиратися з наявних, у тому числі і застарілих комп'ютерів (нагадаємо про різноманітні Beowulf-системи). Зазвичай розрізняють такі види кластерів: відмовостійкі кластери (відмінна риса – надмірна кількість вузлів, що гарантують надання сервісу в разі відмови одного або декількох серверів), кластери з балансуванням навантаження (у даному випадку один або кілька вхідних вузлів розподіляють запити між кількома вузлами, які обробляють ці запити), обчислювальні кластери (використовуються ресурсоємних, як правило, наукових обчислень) [5].

Грід-системи – по суті, один із видів кластерних систем, що має на увазі географічно розподілену архітектуру.

1.3 Особливості Грид -технології та хмарні обчислення

Грід-система – це "віртуальний суперкомп'ютер", що складається з кластерів і слабо пов'язаних різнорідних комп'ютерів, що працюють разом для виконання величезної кількості завдань. З точки зору мережевої

організації, Грід – це узгоджене, відкрите і стандартизоване середовище, що забезпечує гнучке, безпечне і скоординоване поділ обчислювальних та запам'ятовувальних ресурсів, що є частиною цього середовища в рамках однієї віртуальної організації. Відмінними рисами є Грід-безпека та географічно розподілені вузли, які можуть бути розташовані в будь-якому місці та зазвичай підключені через Інтернет [3]. Наприклад, комп'ютери, розташовані в різних частинах світу, можуть працювати разом над одним і тим самим завданням. Існує кілька типів Грід-систем: добровільна грід (заснована на використанні вільно наданого безкоштовного ресурсу персональних комп'ютерів), наукова Грід і комерційна Грід. Багато хто вважає, що головною відмінністю хмарних обчислень від Грід-технологій є віртуалізація. У той час як Грід-системи забезпечують високе використання обчислювальних ресурсів шляхом розподілу одного складного завдання на кілька обчислювальних вузлів, хмарні обчислення йдуть шляхом виконання декількох завдань на одному сервері як віртуальні машини. Віртуальне середовище дозволяє розділити такі ресурси, як центральний процесор (CPU), пам'ять, введення-виведення та мережа з однієї хост-системи на кілька середовищ. Обчислювальні ресурси Грід-системи забезпечують користувача обчислювальною потужністю. Обчислювальні ресурси можуть бути як кластерами, так і окремими робочими станціями. За всього різноманіття архітектур будь-яка комп'ютерна система може розглядатися як потенційний обчислювальний ресурс Грід-системи. Необхідною умовою для цього є наявність проміжного програмного забезпечення, що реалізує стандартний зовнішній інтерфейс із ресурсом та дозволяє зробити ресурс доступним для Грід-системи [5]. Основними загальними завданнями Грід є:

- створення великомасштабних розподілених обчислювальних систем та систем для обробки, комплексного аналізу та моніторингу даних із серійно обладнання, джерела яких також можуть бути (глобально) розподілені;
- підвищення ефективності обчислювальної техніки шляхом надання в грід ресурсів, що тимчасово простоюють.

Пріоритет тієї чи іншої спільної задачі, що вирішується за допомогою Грид, визначається типом Грид та характером прикладних областей, в яких він використовується [1, 7].

В результаті досягнення наукових чи інженерних цілей може бути утруднено, дуже дорого, а іноді й зовсім неможливо. 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 і які належать різним людям та установам протягом виконання завдання або проекту, то проблема може бути вирішена. Така можливість забезпечується Грід-середовищем для розподілу окремих частин великого завдання з географічно віддалених ресурсів (якщо сама природа завдання дозволяє розділити її на частини) [3, 5].

Грід-технології – це розвиток та узагальнення мета-комп'ютерних ідей. Як процесорних потужностей розглядаються як суперкомп'ютери, а й будь-які комп'ютери загалом. Загальні ресурси: комунікації, дані, програмне забезпечення та процесорний час. Грід – це відкрите та стандартизоване середовище, що забезпечує гнучке, безпечне та скоординоване спільне використання ресурсів у віртуальній організації (рисунок 1.1).

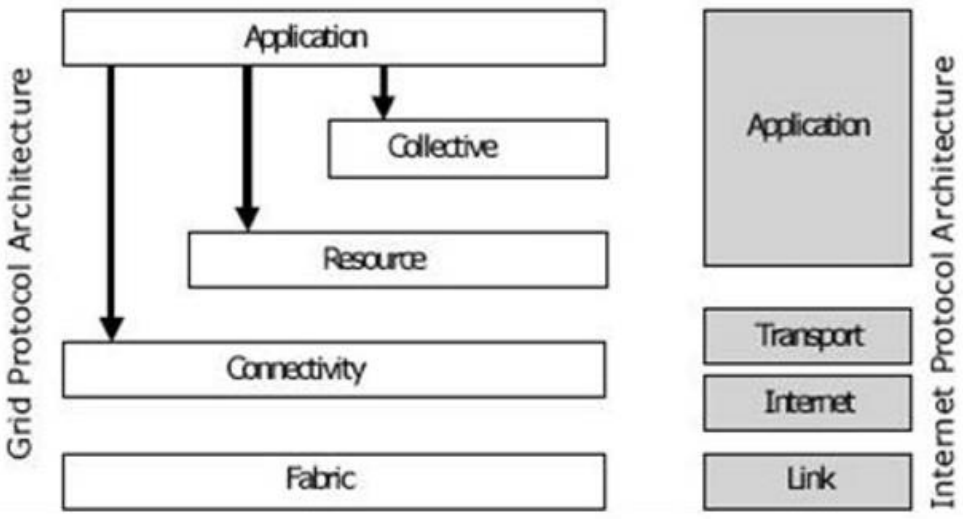

Рисунок 1.1 – Рівні Грід архітектури

Грід характеризується відсутністю центру управління обчислювальними ресурсами, використанням відкритих стандартів та нетривіальним рівнем сервісу [13]. Обчислювальні вузли Grid зазвичай розташовані далеко один від одного, слабо пов'язані між собою через інтернет-канали, і доступність того чи іншого з них у довільний час не гарантована. Це накладає додаткові вимоги управління ресурсами [8].

Сервіс-орієнтована архітектура є основою для побудови надійних розподілених систем, що базуються на послугах зі стандартизованими інтерфейсами та на принципі слабкого зв'язку між взаємодіючими службами (рисунок 1.2).

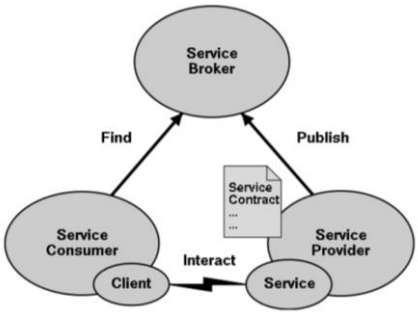

Рисунок 1.2 – Сервісно-орієнтована архітектура

Сервіс-орієнтована архітектура (SOA) має такі характеристики:

- архітектура розподілена, функціональні модулі програми (системи) можуть бути розподілені між кількома обчислювальними системами та взаємодіяти з використанням локальних чи глобальних мереж;
- інтерфейс функціональних модулів такий, що використання модулів не залежить від технології або платформи, в рамках якої вони реалізовані;
- можливий динамічний пошук та підключення потрібних функціональних модулів.

Архітектура виходить з загальноприйнятих галузевих стандартах.

Технології та системи хмарних обчислень пов'язані з темою високопродуктивних обчислень. Хмарні обчислення – це розподілена технологія обробки даних, в якій комп'ютерні ресурси та потужність

надаються користувачеві як інтернет-сервіс. Суть хмарних обчислень полягає у наданні кінцевим користувачам віддаленого динамічного доступу до обчислювальних ресурсів, сервісів та програм (включаючи інфраструктуру та операційні системи) через Інтернет. Таким чином, хмарні обчислення – це апаратне та програмне забезпечення, доступне користувачеві через Інтернет (або локальну мережу) як сервіс, що дозволяє використовувати зручний веб-інтерфейс для віддаленого доступу до виділених ресурсів (обчислювальних ресурсів, програм та даних). Комп'ютер користувача діє як стандартний термінал, підключений до мережі. Комп'ютери, що виконують хмарні обчислення, називаються обчислювальними хмарами. У цьому випадку навантаження між комп'ютерами, що входять до складу обчислювальної хмари, розподіляється автоматично [8].

Розглянемо моделі служб хмарних обчислень. Програмне забезпечення як послуга (SaaS) – можливості надаються споживачеві при використанні програм постачальника, що працюють на хмарній інфраструктурі. Програми доступні для різних клієнтських пристроїв через інтерфейс клієнта, наприклад веб-браузер. Споживач не керує та не контролює базову хмарну інфраструктуру, включаючи мережу, сервери, операційні системи, сховище або навіть окремі функції програми, за можливим виключенням обмежених конкретних параметрів конфігурації програми [7].

Платформа як послуга (PaaS). Споживачу надається можливість розгортати створені споживачем або отримані ним програми в хмарній інфраструктурі, створені за допомогою мов програмування та інструментів, які підтримує постачальник. Споживач не керує та не контролює базову хмарну інфраструктуру, включаючи мережу, сервери, операційні системи або сховище, але керує розгорнутими програмами та, можливо, програмою, яка змінює конфігурацію середовища.

Інфраструктура як послуга (IaaS). Можливості надаються споживачеві у вигляді обробки, зберігання даних, мережевих та інших базових обчислювальних ресурсів, де споживач має можливість розгортати та

виконувати користувальницьке програмне забезпечення, яке може включати операційні системи та додатки. Споживач не керує і не контролює базову хмарну інфраструктуру, а керує операційними системами, сховищем, розгорнутими програмами та мережевими компонентами (такими як брандмауери хостів) [10].

Гетерогенні обчислення. Використання графічних процесорів для загальних обчислень – новий перспективний напрямок. Гетерогенні обчислення відкривають нові перспективи на вирішення ресурсоємних завдань. Нині ця область активно розвивається: так, вже створено окремий стандарт. OpenCL – відкритий стандарт паралельного програмування для гетерогенних обчислень. Він визначає Сподібна мова для написання обчислювальних ядер та API. OpenCL дозволяє використовувати у обчисленнях і CPU, і GPU. Його ключовою особливістю є переносимість коду. Відомо, що код для GPGPU завжди був платформозалежним. Різні вендори пропонували свої власні API та мови програмування. Так, перехід з однієї платформи на іншу, досконалішу, міг вилитися у величезну проблему. Вирішенням цієї проблеми і є OpenCL. Його реалізації є у найбільших виробників CPU та GPU [4].

CUDA (Compute Unified Device Architecture) – розроблена компанією NVIDIA, яка дозволяє виконувати обчислення з використанням графічних процесорів NVIDIA, що підтримують технологію GPGPU.

OpenCL (Open Computing Language) – це стандарт Хроноса, фреймворк для написання комп'ютерних програм, пов'язаних із паралельними обчисленнями на різних платформах. Основна ідея OpenCL у тому, щоб надати програмісту універсальний інструмент використання всіх обчислювальних потужностей сучасних систем.

Віртуалізація – це рівень програмного забезпечення, що розташований між апаратним забезпеченням та основною операційною системою і дозволяє працювати паралельно з кількома операційними системами на одній фізичній машині. Багато виробників працюють у цій галузі, серед яких Xen і VMWare

добре відомі приклади. Віртуалізація забезпечує безпечне та ізольоване операційне середовище для всіх типів традиційного програмного забезпечення [10].

Віртуалізація серверних ресурсів одна із основних програм віртуалізації. Такий підхід значно знижує витрати на придбання та обслуговування серверів: вам не потрібно купувати та обслуговувати багато серверів, що працюють за принципом "одна програма-один сервер", достатньо купити потужний сервер, який вважатиме багато додатків і підтримуватиме його один. Ключовими моментами тут є безпека і стійкість до відмови: серверні ресурси розподіляються між додатками таким чином, що кожен додаток використовує лише ресурси, які йому безпосередньо виділяються. Це усуває масу проблем: програми більше не конфліктують через необхідність використовувати одні й ті ж ресурси (завдання з розподілу ресурсів призначаються відповідним інструментам), а "зависання" одного з додатків ніяк не впливає на роботу інших; адже кожен з них працює у своїй. ізольованого середовища.

Віртуалізація серверів існує у таких формах [10]: фізичне (апаратне), використання віртуальних машин і віртуалізація лише на рівні операційної системи та у формі управління ресурсами сервера. Для фізичної віртуалізації використовуються динамічні Домени-апаратні розділи, засновані на кількох системних платах та з'єднані високошвидкісним комутатором. Кожен із них працює як окремий комп'ютер, будучи при цьому електрично ізольованим, що дозволяє здійснювати безперервну модернізацію та технічне обслуговування. Такий підхід дозволяє динамічно перерозподіляти ресурси сервера та обслуговувати окремі домени, не зупиняючи роботу сервера загалом. Таким чином, обслуговування клієнтів не зупиняється ні на хвилину, і користувачі навіть не помітять збою в одному домені. Підтримка динамічних доменів була вперше реалізована компанією Sun Microsystems (нещодавно придбаною корпорацією Oracle) понад десять років тому, і вона залишається головним розробником інструментів, які використовують цю

технологію і досі. Приклад рішення, що використовує цю технологію, є сервер Sparc Enterprise M9000, який підтримує до 24 динамічних доменів. У той час цей сервер був визнаний одним з найшвидших, показавши 2,023 TFlops на тесті LINPACK, використовуваному визначення швидкості суперкомп'ютерів.

Ідея віртуалізації програм не нова і дуже популярна в корпоративному секторі. Це дозволило вам запускати застарілі програми у сучасному операційному середовищі. Віртуалізація дала поштовх подальшому розвитку цієї ідеї та дозволила її значно вдосконалити. Віртуалізована програма виконується в окремому контейнері, який також містить необхідні параметри середовища, змінні середовища, бібліотеки та ресурси. Це дозволяє зменшити взаємний вплив запущених програм між собою та операційної системою і практично дозволяє запускати застарілі і несумісні докладання, різні версії однієї й тієї програми одночасно без урахування сумісності [7].

Віртуалізація пам'яті відноситься до інтеграції оперативної пам'яті до пулу віртуальної пам'яті. При цьому пам'ять окремого вузла перестає бути прив'язаною до нього. Тепер будь-який комп'ютер у кластері має доступ до всього пулу пам'яті, тобто до пам'яті всіх машин разом. Віртуалізація пам'яті дозволяє обійти фізичні обмеження обсягу пам'яті [1]. Це підвищує загальну продуктивність системи, підвищує ефективність пам'яті. Віртуалізація пам'яті буває двох видів. Перший пропонує інтеграцію лише на рівні додатків, де окремі програми отримують доступом до спільної пам'яті безпосередньо через програмні інтерфейси. Другий - це інтеграція лише на рівні ОС. У цьому випадку ОС встановлює з'єднання з пулом пам'яті і робить пам'ять пула доступною для всіх програм. Необхідно розрізняти віртуалізацію пам'яті із загальною пам'яттю (Shared Memory) [5].

Загальна пам'ять не дозволяє абстрагуватись від ресурсів. Віртуалізація пам'яті усуває зв'язок між логічними та фізичними ресурсами та призводить до ситуації, коли ресурси розподіляються в міру необхідності. Технології NUMA і SMP використовують загальну пам'ять, але у межах лише однієї

багатопроцесорної системи, тоді як віртуалізація пам'яті передбачає об'єднання ресурсів кількох комп'ютерів у мережі.

Віртуалізація зберігання даних передбачає абстракцію зберігання інформації, коли дані зберігаються на декількох пристроях, об'єднаних у загальну систему зберігання. Він забезпечує надійне управління та консолідацію, поєднання ресурсів зберігання в пул. Він забезпечує прозорий доступ до даних у рамках багаторівневої системи. Водночас, системи зберігання, об'єднані в загальний віртуальний пул, використовуються більш ефективно, оптимізується навантаження на окремі пристрої, усувається можливість виникнення вузьких місць, значно спрощується управління системою зберігання, що знижує витрати на адміністрування, а також скорочує час резервного копіювання та відновлення даних. Підвищується гнучкість та керованість системи, знижується загальна вартість зберігання даних [2, 7].

Віртуалізація мереж торкається області, яка стала однією з найважливіших в галузі обчислювальної техніки, комп'ютерних мереж. Фізичні ресурси мережі діляться кілька логічних, віртуальних мереж. На базі фізичної мережі створюється кілька логічних мереж, що використовують одне й те обладнання. Можна створити кілька віртуальних локальних мереж на базі одного підприємства для потреб окремих користувачів та груп, а можна створити віртуальну мережу поверх глобальної мережі, що найчастіше відбувається. Віртуальні локальні мережі (VLAN) використовуються для поділу мережі між кількома підприємствами в одному будинку або підрозділами всередині одного підприємства, використовуючи одні й ті самі фізичні ресурси для організації мережі. Віртуальні мережі поверх іншої мережі дуже зручні для віддаленого доступу до конфіденційних даних.

2 АСПЕКТИ РОЗРОБКИ МАТЕМАТИЧНОЇ МОДЕЛІ РОЗПОДІЛЕНИХ ОБЧИСЛЮВАЛЬНИХ СЕРЕДОВИЩ

2.1 Оптимізація прискорення обчислювальних середовищ

Проблема масштабованості було питанням останніх років і навіть на сьогоднішній день і відіграє велику роль в галузі інформаційних технологій та дослідницьких цілей. Потенційне прискорення, яке можна отримати від паралельної обробки, відтоді було проведено велику кількість досліджень у різних апаратних архітектурах. Оцінка прискорення для критичних за часом додатків є складним завданням, що включає безліч факторів і точок зору [5], наприклад, під час роботи з науковими додатками, що включають велике моделювання та екстремальні обмеження швидкості. Хоча велика кількість розпаралелених додатків вже добре працюють між клієнт-серверними платформами. Однак ці програми дивовижно паралельні, працюють на різних платформах, кожна з яких має свою власну операційну систему (ОС). Це дозволить нам краще оцінити виклики прискорення у галузі сучасних інформаційних технологій. Насправді прискорення також залежить від різних факторів, включаючи притаманний самій системі паралелізм, апаратні архітектури машин та ОС з гнучкими об'єктами для виділення та призначення процесорів та ресурсів пам'яті [6].

2.1.1 Узагальненість Закону Амдалю

Закон Амдалю заслуговує на особливу увагу! коли організація має комп'ютер із вісьмома процесорами, у якому успішно виконується паралельна програма. Організація отримує кошти на розробку і вирішує купити більш продуктивний комп'ютер із 32 процесорами, розраховуючи, що програма прискориться вчетверо швидше. Але після купівлі нового комп'ютера та запуску завдання на ньому, виявляється, що програма почала

працювати всього в 2 рази швидше. Причину буде розкрито законом Амдала. Джин Майрон Амдал-інженер, який працював в IBM і засновник корпорації Amdahl Corporation. Він сформулював

2.2 Технології програмних продуктів для побудови гетерогенного середовища

Для побудови ефективного гетерогенного середовища, нами були обрані різні програмні продукти та технології такі як, Univa Grid Enigne для управління ресурсами, MOSIX платформа для розробки ефективного комплексу, що забезпечує кращу високопродуктивність за допомогою балансування навантаження, Globus Toolkit для організації системи доступу, Грід протоколи, що забезпечують системи безпеки 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 загальними обчислювальними ресурсами. Такі системи забезпечують відносно низькі обчислювальні витрати, мають високу масштабованість, високий рівень надійності і мають перевірені інструменти для проектування, налагодження та аналізу паралельних програм Ефективність обчислювальних середовищах визначається їх наявності, в основному оцінюються за їх системами балансування навантаження, системи управління ресурсами, виконання робочого навантаження та обробки даних для вирішення величезної кількості. Щоб виконати ці види вимог, ми потребуємо потужного обчислювального середовища з надійних систем балансування навантаження, систем із загальною пам'яттю та надійна система обробки даних з величезною кількістю.

2.2.1 Організація системи доступу користувачів до розподіленого обчислювального середовища за допомогою Globus Toolkit

Globus Toolkit – це високопродуктивні програмні інструменти з відкритим вихідним кодом для Гриду, що включають програмні сервіси та бібліотеки для моніторингу ресурсів, забезпечення безпеки та управління ними. У цій роботі досліджено методи організації системи доступу користувачів до розподіленого обчислювального середовища та методи авторизації користувачів за принципом єдиного вікна. Для забезпечення зручного та безпечного доступу користувачів ми вивчили надійні методи автентифікації та авторизації програмного продукту Globus Toolkit.

Globus Toolkit є одним з основних інструментів, що використовуються для побудови Грід-систем та їх додатків. Він був розроблений у 1996 році для підтримки розвитку сервіс-орієнтованих розподілених обчислень. Проект "Глобус" виріс завдяки стратегії з відкритим кодом. Це провокує більш широке та швидке розповсюдження та призводить до великих технічних інновацій, оскільки спільнота користувачів постійно покращує якість продукту [16]. Мета його створення-дозволити додаткам працювати з розподіленими гетерогенними обчислювальними ресурсами у вигляді єдиної віртуальної машини. Основна увага у цьому проекті приділяється обчислювальним сіточним системам. Обчислювальна Грід-система - це інфраструктура апаратних та програмних ресурсів, що забезпечує надійний та повномасштабний доступ до високопродуктивних обчислювальних систем незалежно від географічного розташування користувачів або ресурсів [16]. Globus надає набір стандартних веб-служб для забезпечення безпеки, управління даними, інформаційних служб та виконання процесів, а також набір утиліт та бібліотек, які дозволяють цим службам бути незалежними від платформи, функціонувати на низькому рівні стеку веб-служб (WSRF, WSN) та дозволяють реалізовувати додаткові веб-служби з використанням різних мов програмування, таких як Java, C та Python. Це дозволяє використовувати

GT4 для побудови як розподілених об'єктних систем, так і сервіс-орієнтованих архітектур (рисунок 2.1).

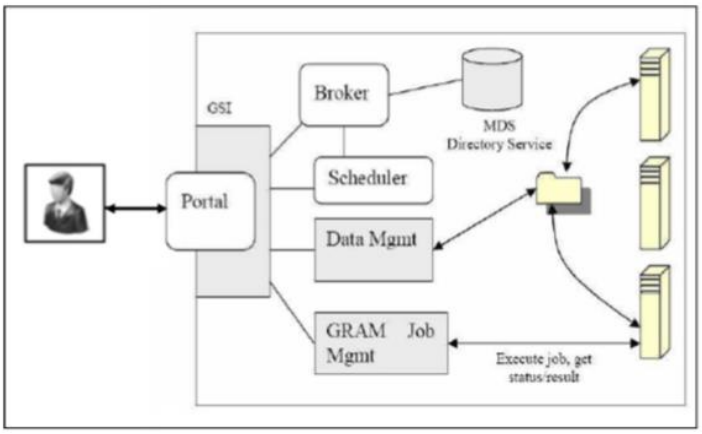

Рисунок 2.1 – Загальна схема взаємодії компонентів Globus Toolkit

Базовим елементом системи Globus є інструментарій Globus Toolkit, який описує основні сервіси та можливості, необхідні для створення обчислювальних Грід-систем [3,4]. Система Globus надає високорівневим програмам доступ до сервісів, кожен з яких додаток або розробник може використовувати для досягнення власних цілей. Цей спосіб роботи може бути реалізований лише за наявності високого ступеня ізоляції окремих сервісів та чітко визначеного програмного інтерфейсу для кожної послуги.

2.3 Єдиний образ операційної системи

Єдиний образ операційної системи означає, що всі розподілені ресурси організовані до єдиного пристрою для користувачів, користувачам не потрібно керувати кожними вузлами обчислювального середовища. SSI включає деякі атрибути, такі як єдине простір пам'яті, єдина система вводу/виводу, єдина файлова система, система управління окремими навантаженнями, тощо. Ключові атрибути SSI є єдиним простором пам'яті і єдиним простором процесу. SSI кластери можуть бути реалізовані на апаратному рівні, рівень проміжного та на рівні додатків. У принципі,

архітектури для багатопроцесорних систем можна розділити на дві групи: ті, з когерентним розподіленої пам'яті, таких як SMP і cc-NUMA систем, а ті, без, таких, як мережі робочих станцій товарних з'єднаних Ethernet. Загальні системи пам'яті забезпечують прості моделі програмування, сумісні з великою базою існуючих програм і операційних систем. Більшість існуючих операційних систем підтримують такі архітектури; це відносно просте завдання підтримувати їх, навіть якщо не з оптимальною продуктивністю. З іншого боку, вони, природно, надати себе єдиний образ операційної системи (SSI), де є один екземпляр операційної системи з одного простору ресурсів [12]. Поява єдиної системи робить простий для користувачів та адміністраторів. Єдиний ресурс означає, що обчислення можуть прозоро мігрувати між процесорами, щоб збалансувати навантаження.

2.3.1 Дослідження MOSIX

У нашому експерименті MOSIX кластер був створений у віртуальному тестовому полігоні ПМ-ПУ для аналізу продуктивності, тому побудували власне ефективне обчислювальне середовище. MOSIX: набір розширень для ядра Linux, що дозволяє розподіляти процеси між вузлами кластера. MOSIX можна логічно поділити на дві частини. Це міграція процесів та набір алгоритмів для ефективного динамічного розподілу ресурсів. Обидві частини реалізовані лише на рівні ядра операційної системи та прозорі додатків. Міграція здійснюється в такий спосіб. Процес ділиться на два контексти: контекст користувача і системний контекст. Контекст користувача містить сегмент коду, стек, сегмент даних та вміст регістру. Системний контекст містить інформацію про використовувані ресурси та стек ядра. Інтерфейс між користувальницькою та системною частинами встановлюється на мережному рівні. Користувацька частина може мігрувати, а системна частина жорстко пов'язана з вузлом, на якому було запущено процес [12]. У нашому MOSIX кластері ми протестували продуктивність наукового застосування Кристал,

який працює ефективно і паралельно з усіма розподіленими обчислювальними вузлами. Кристал – це програмний інструмент для хімії та фізики. Програма "Кристал" була розроблена групою теоретичної хімії та групою обчислювального матеріалознавства Туринського університету. Він був розроблений в основному для розрахунків кристалів (тривимірних), пластин (двовимірних) та полімерів (одномірних) з використанням вимірювань трансляційної симетрії, але також може бути використаний для розрахунків окремих молекул. Результати випробувань наведено на рисунку 2.2.

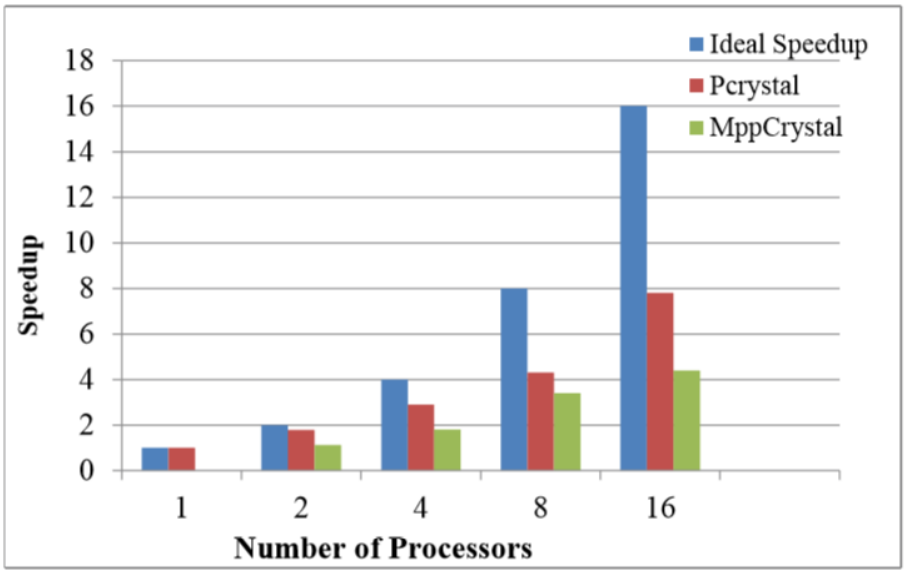

Рисунок 2.2 – Аналіз продуктивності кластеру MOSIX

У цьому експерименті результати тестування прискорення нашого обчислювального середовища MOSIX показують, що прискорення обчислень близьке до ідеалу.

2.4 Розподілене динамічне балансування навантаження Грід

Нещодавно, розподілених обчислень є одним із привабливих архітектур для високопродуктивних обчислень. Грід обчислювальна система

є Інтернет-масштабу розподілених обчислень система для спільного використання розподілених ресурсів по традиційній організації кордону. У Грід-систем, найбільш важливі питання включають як інтегрувати динамічно гетерогенних розподілених ресурсів, і як підвищити ефективність використання цих інтегрованих ресурсів. Хоча ці різні проекти Грід спрямовані на обмін розподілених ресурсів з різних віртуальних організацій (VO), це, як і раніше, важко поділитися розподілених ресурсів у зв'язку з різними цілями у будівництві інше віртуальні організації [9]. Є багато проміжних програм (наприклад, Globus Toolkit, Unicore, GLite і т.д.), які були розроблені для систем грід. Більшість із них зосереджені на забезпеченні основних послуг проміжного підтримки функціональних можливостях розробки додатків високого рівня. Проте, вони, зазвичай, залежить від спеціалізованих серверів підтримки розподілену інформацію про ресурси [8].

2.4.1 Дослідження системи управління ресурсами з Univa Grid Engine

Управління ресурсами є одним із найважливіших завдань з технологічного проміжного. Ресурси включають доступну обчислювальну потужність (тобто ЦП), пам'ять та допоміжні системи зберігання даних. Стратегії, що реалізуються проміжного принципово визначити, наскільки рано робота може закінчити своє виконання та забезпечити бажані результати обчислень [7]. Univa Grid Engine є розподілене управління ресурсом системи галузі (DRM) використовується сотнями компаній у всьому світі, щоб побудувати великі інфраструктури, обчислювальні кластери для обробки великих обсягів навантаження. Висока масштабованість і надійна система DRM, Grid Engine дозволяє компаніям виробляти більш якісні продукти, скоротити час виходу на ринок, і впорядкувати та спростити обчислювальне середовище. У цій роботі ми використовували Univa Grid Engine 8.0, початкове вивільнення Grid Engine і проходить контроль якості та повністю підтримується Univa. Цей реліз

покращує Grid Engine з відкритим вихідним кодом ядра платформи з розширеними можливостями та інтеграції в корпоративні готові системи, а також рішень для управління хмарою, які дозволяють організаціям впроваджувати найбільш масштабоване та найвище розподілених обчисленнях рішення на ринку.Univa Grid Engine дозволяє легко створити кластер тисяч машин за рахунок використання комбінованого обчислювальну потужність настільних комп'ютерів, серверів та хмар у простий, простий в адмініструванні середовища.

Політики планування можуть бути застосовані до всіх робіт, що подаються в кластер, забезпечуючи першочергові завдання буде завершено вчасно, одночасно підтримуючи максимальне використання всіх касетних машин. З Grid Engine, будь-який ресурс або ліцензійне програмне забезпечення можна контролювати, відстежувати і планується забезпечити програми автоматично узгоджені з відповідними ліцензіями та машин у кластері. Перерахуємо особливості цієї версії. Grid Engine забезпечує кілька політик планування для узгодження навантаження кластері до бізнесу та організаційних цілей, таких як максимізація використання на всіх машинах, скорочення часу обороту на роботу в кластері, або пріоритетності навантаження відповідно до групи, відділу або компанії власності. Grid Engine безперервно збирає метрики від усіх вузлах кластера, а потім використовує планування стратегії, налаштовані адміністратором оцінити всіх переданих навантаження і відповідати певним вимогам робочих місць ресурсів, що вимиються. Усі ресурси в центрі обробки даних або кластера повинні бути ефективно розділяє. Grid Engine забезпечує потужний та гнучкий контроль потужності ресурс для всіх загальних ресурсів. Grid Engine можна масштабувати до кластера 64000 ядер і більше в одному керованому середовищі.

У цій роботі програмний продукт Univa Grid Engine було проаналізовано та встановлено у двох багатопроцесорних серверах ULTRA SPARC T1. Системи UltraSPARC T1 забезпечують високу продуктивність за

досить помірної тактової частоти (до 200 МГц) з допомогою оптимізації середньої кількості команд, виконуваних за такт. Однак за такого підходу закономірно виникають питання ефективного управління конвеєром команд та ієрархією системної пам'яті. Для підвищення продуктивності необхідно максимально скоротити середній час доступу до пам'яті і збільшити середню кількість команд, що видаються для виконання в кожному такті, не перевищуючи розумного рівня складності процесора [8].

Архітектура процесора UltraSPARC T1 включає вісім ядер на основі специфікації SPARC V9 і 90-нанометрову дев'ятирівневу "мідну" технологію CMOS, що забезпечує паралельну обробку 32 потоків даних (4 потоки на ядро процесора).

Процесор UltraSPARC T1 можна як систему, що складається з 32 однопоточних логічних процесорів. Вісім ядер зібрані у загальній серверній SMP-архітектурі [9]. Кожне з його ядер здатне до апаратної багатопоточності (Hardware Multithreading), тобто воно може розділити свій процесорний час і ресурси між чотирма паралельними командними потоками.

Grid Engine дозволяє об'єднати кілька серверів або робочих станцій в один обчислювальний ресурс, який може використовуватися як для пакетних завдань, так і високопродуктивних пакетних обчислень. Адміністратор комп'ютерної мережі може отримувати дані моніторингу та статистики та використовувати їх для оптимізації використання ресурсів. Адміністративний інтерфейс дозволяє задавати різні параметри для обчислювальних завдань, такі як пріоритети, необхідні апаратні ресурси, ліцензії на програмне забезпечення, тимчасове вікно та права користувача доступу до певних ресурсів. На рисунку показані компоненти кластеру Univa Grid Engine (рисунок 2.3).

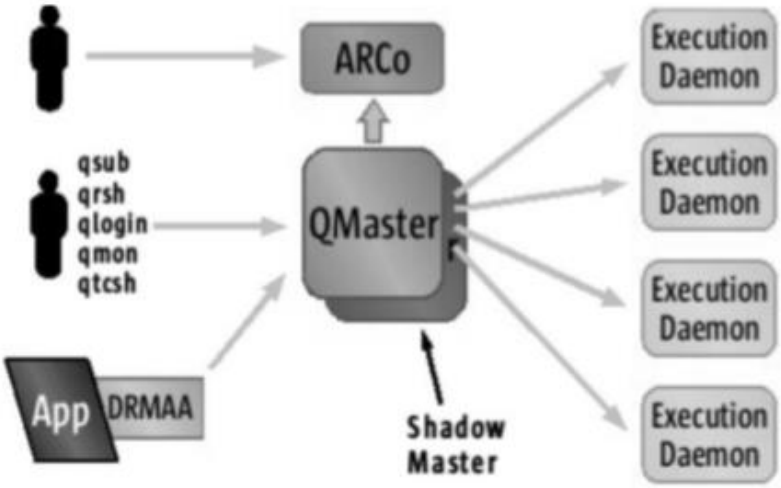

Рисунок 2.3 – Компоненти Univa Grid Engine

У центрі діаграми знаходиться qmaster. Цей центральний компонент Univa Grid Engine керує кластером, приймаючи вхідні завдання від користувачів, призначаючи завдання ресурсів, відстежуючи стан кластера і обробляючи команди управління. qmaster-це багатопотоковий демон, що працює на одному хості в обчислювальному кластері. Щоб скоротити незапланований час простою кластера, один або кілька тіньових майстрів можуть виконувати завдання на додаткових вузлах кластера. Якщо qmaster або хосту не вдається виконати завдання, завдання передається новому qmaster, запускаючи новий демон qmaster. Кожен хост у кластері, якому необхідно виконати завдання, повинен запустити відповідний демон. Демон отримує завдання від qmaster і виконує їх у певному місці на своєму хості. Програмне забезпечення Univa Grid Engine не встановлює обмежень на кількість завдань, які демон може розповсюджувати, але в більшості випадків кількість завдань визначається кількістю ядер процесора, доступних на хості. Коли завдання завершено, демон повідомляє qmaster, що може запланувати нове завдання. У стандартному режимі кожен демон надсилає повідомлення про статус qmaster. Якщо qmaster не впорається з одним із завдань, отримавши кілька послідовних повідомлень від демона, qmaster не зареєструє

цей хост і всі його ресурси як доступні і видалити його зі списку планувальника як доступні.

Завдання відправляється в qmaster у різний спосіб. DRMAA надає програмний інтерфейс для програми, що дозволяє запускати завдання та керувати ними. Програмне забезпечення Univa Grid Engine працює з C та Java, що дозволяє використовувати DRMAA для широкого спектру програм. qmon це графічний інтерфейс для Univa Grid Engine. За допомогою qmon користувачі та адміністратори можуть запускати, контролювати та керувати завданнями, а також керувати всіма функціями кластера. Qsub-це утиліта командного рядка для запуску черг, пакетів та паралельних завдань. Останній компонент, показаний на діаграмі, Arco (Accounting and Reporting Console), інтернет-додаток, що надає доступ до двигуна Univa Grid Engine для обліку інформації, що зберігається в базі даних. За допомогою ARCo кінцеві користувачі та адміністратори можуть створювати та виконувати запити кластерного обліку (рисунок 2.4).

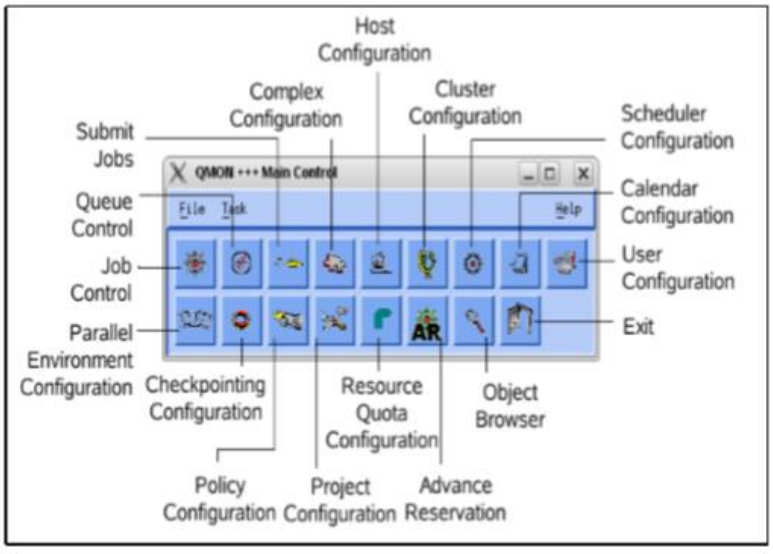

Рисунок 2.4 – Графічний інтерфейс користувача – qmon

Служби та інші об'єкти, що існують та взаємодіють у сітці, реалізуються як демони UNX. Крім того, UGE пропонує величезний набір інструментів командного рядка для планування завдань, моніторингу та

загального управління з можливостями резервного копіювання та зручним інтерфейсом.

Такий підхід відкриває великі можливості для роботи зі скриптами в операційному середовищі, наприклад, у термінальному режимі. UGE використовує в основному 4 типи хостів: Master host, Execution host, Administration host and Submit host. Кожен хост може бути членом більш ніж однієї категорії одночасно і управлятиме відповідними демонами UGE. Єдине обмеження до цього підрозділу хостів полягає в тому, що може існувати лише один Master-хост на верхньому рівні UGE grid (названий Cell) [16].

У роботі було використовано систему управління розподіленими ресурсами UGE у двох багатопроцесорних серверах Ultra Sparc T1. І розглянули виконання програми PI для розрахунків у розподіленому обчислювальному середовищі. Як видно, ми маємо дуже добрі результати: прискорення розрахунків близьке до ідеального (рисунок 2.5).

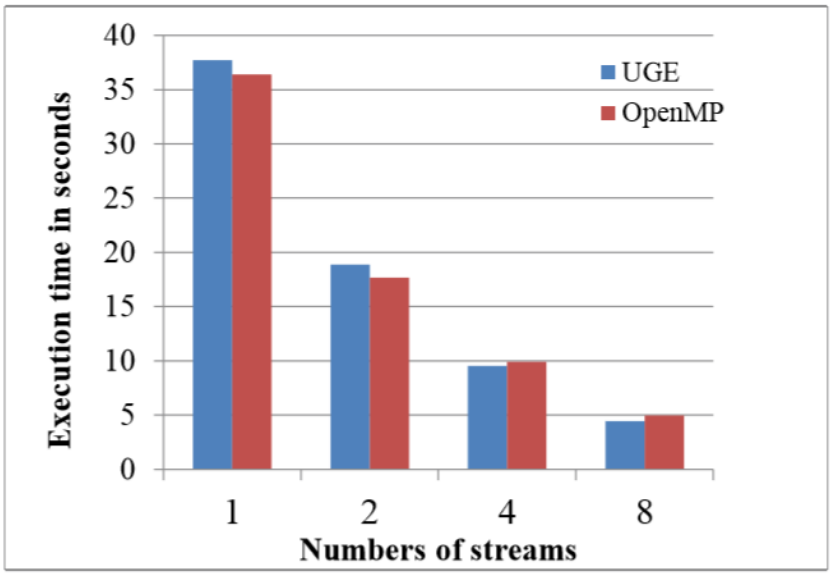

Рисунок 2.5 – Результати часу виконання програми PI

Univa Grid Engine призначений для кластерної Grid-класу мережі та доступний для вільного. У системі UGE існує спеціальна структура даних

паралельне середовище, що дозволяє враховувати особливості додатків, які використовують технології паралельного програмування. Для вирішення паралельних завдань ми інтегрували UGE та Openmp з додатковим пакетом скриптів. Двигун сітки Univa був вибраний для розподіленої системи керування ресурсами.

2.5 Розробка системи управління ресурсами обчислювального середовища

Центральний компонент Univa Grid Engine управляє розподіленою системою та кластером, приймаючи вхідні завдання від користувачів, призначаючи завдання ресурсів, відстежуючи поточний стан кластера та обробляючи команди управління. Користувачі можуть надсилати завдання головному демону за допомогою команди (qsub) та перевіряти їх стан (за допомогою команди qstat). Існує графічний інтерфейс (QMON) та інтерфейс програмування драми (Distributed Resource Management Application API), який дозволяє працювати з будь-якою іншою програмою або писати скрипти. Це інтерфейс, який ми використовували для інтеграції інструментів Globus та Linux. Користувачі Globus можуть запустити завдання за допомогою команди globusrun-ws через DRMAA [5].

3 РЕАЛІЗАЦІЯ РОЗРОБЛЕНОГО ГЕТЕРОГЕННОГО ОБЧИСЛЮВАЛЬНОГО СЕРЕДОВИЩА

3.1 Інтеграція системи віртуалізації та єдиного образу операційної системи

Система реалізована для спрощення складності управління та програмування кластерів. Ресурс у обчислювальному середовищі інтегрується з використанням комбінації віртуалізації та розподіленої у спільній пам'яті (DSM), а для гостьової операційної системи в результаті створення єдиного образу системи створюється абстрактна єдина фізична машина. Використовуючи розподілений монітор віртуальних машин (KVM), заснований на апаратних технологіях віртуалізації, KVM містить деякі симетричні та спільні VMM, розподілені на кількох вузлах.

Таким чином, DVM може підтримувати незмінену застарілу операційну систему для прозорої роботи на розподілених вузлах кластера. Наше рішення значно перевершує інші рішення SSI і має більшу прозорість, високу продуктивність і простоту реалізації [2, 5]. Віртуалізація дозволяє консолідувати та об'єднувати ресурси. Віртуалізація забезпечує логічну абстракцію фізичних обчислювальних ресурсів та створює обчислювальні середовища, які не обмежені фізичною конфігурацією чи реалізацією. Віртуалізація дуже важлива для Грид-обчислень, оскільки надання послуг спрощується за рахунок надання платформи для оптимізації різних ресурсів, що масштабується, що робить її більш ефективною.

Гіпервізор відіграє у апаратної віртуалізації. Це частина програмного забезпечення, що забезпечує віртуалізоване апаратне середовище для підтримки одночасного запуску кількох операційних систем з використанням одного фізичного сервера [4,7]. Віртуалізація нині широко використовується для підприємства.

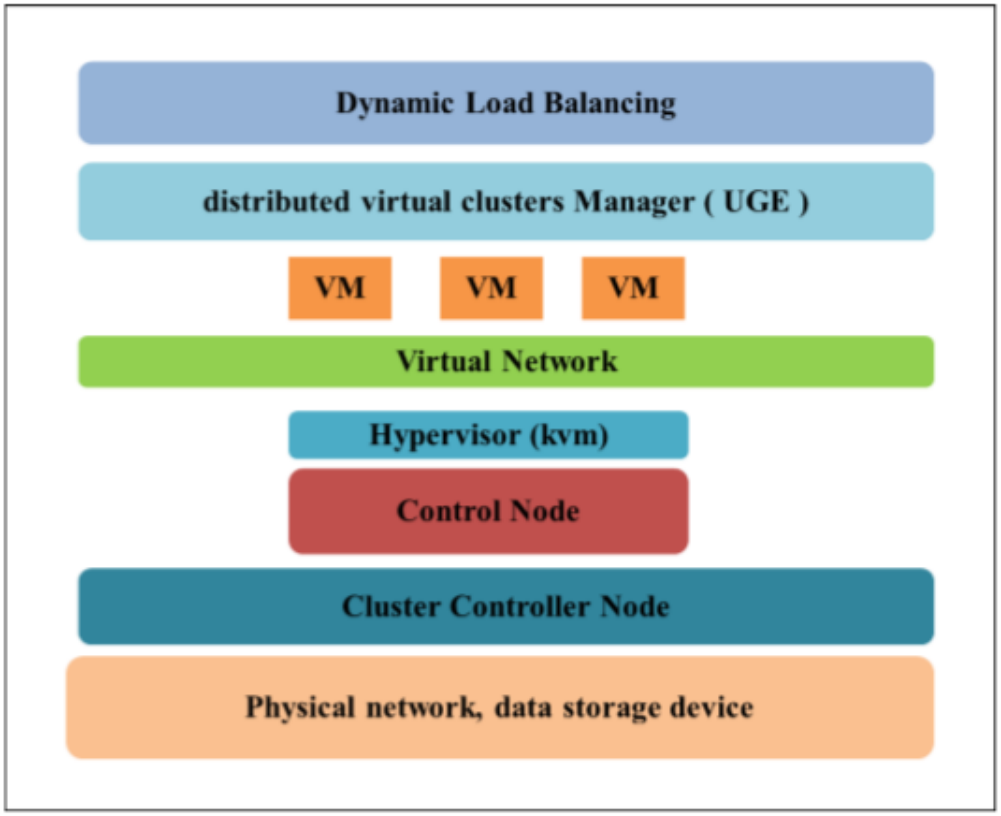

Рисунок 3.1 – Інтеграція віртуалізації та єдиного образу ОС

Віртуалізація має й інші важливі переваги у високопродуктивних обчисленнях, такі як міграція, масштабованість гіпервізорів та багато іншого [8].

3.1.1 Інтеграції у віртуалізації

Віртуалізація була активною темою досліджень з 1970-х років, і новітні технології віртуалізації надають деякі обчислювальні можливості (в останні роки нові машини з багатоядерними процесорами можуть конкурувати з кількома серверами). Ці додаткові функції можна використовувати для запуску віртуальних машин над фізичними машинами.

Поняття ВМ, вона включає п'ять основних функцій:

1) ізоляція (ступінь ізоляції між голими металевими віртуальними машинами і додатками, що працюють на різних віртуальних машинах),

2) консолідація серверів (можливість перемикання на використання ресурсів, виділених для конкретної віртуальної машини),

3) перенесення додатків (можливість запуску незміненої програми),

4) перенесення віртуальних машин (можливість міграції віртуальних середовищ через різні апаратні архітектури),

5) призупинення/перезапуск (можливість зупинки/відновлення віртуальних машин).

Операційна система на основі віртуалізації відображена на рисунку 3.2. Віртуалізація включена до операційної системи, яка підтримує кілька ізольованих та віртуалізованих гостьових ОС на одному фізичному сервері з характеристикою, що всі вони знаходяться в одному ядрі операційної системи з винятковим контролем апаратної інфраструктури. Як операційна система, хост може переглядати віртуальні машини та керувати ними [10].

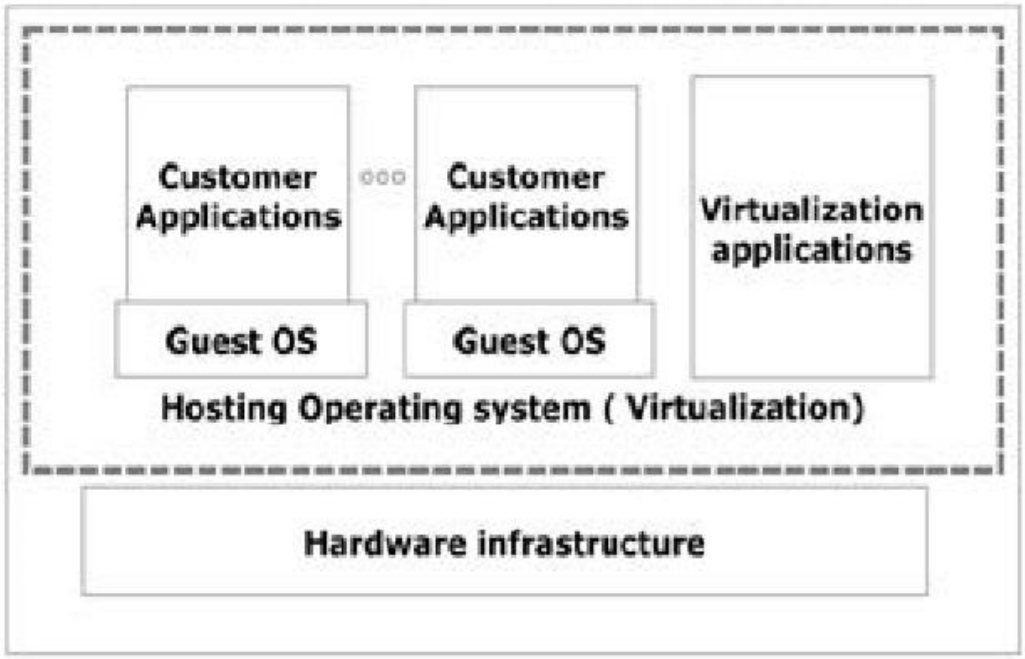

Рисунок 3.2 – ОС на основі віртуалізації

Віртуалізація на основі програми відображена на рисунку 3.3. Додаток на основі віртуалізації розміщується на верхній частині розміщеної операційної системи. Ця програма віртуалізації, яка емулює всі VM містить

свій гостьовий операційної системи та відповідні програми [5].

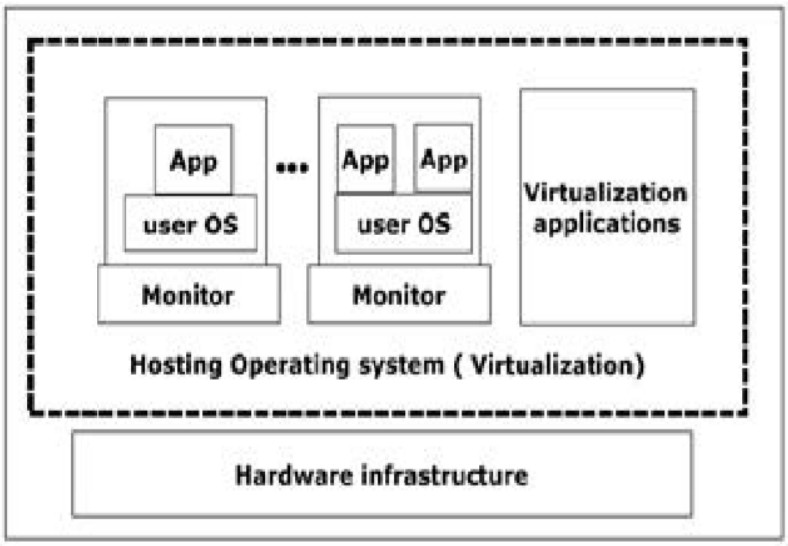

Рисунок 3.3 – Віртуалізація на основі програми

Розглянемо віртуалізацію на основі гіпервізора (рисунок 3.4). Гіпервізор доступний під час завантаження машини для керування загальним доступом до системних ресурсів між кількома віртуальними машинами. Деякі з цих віртуальних машин є привілейованими розділами, які управляють платформою віртуалізації та розміщеними віртуальними машинами.

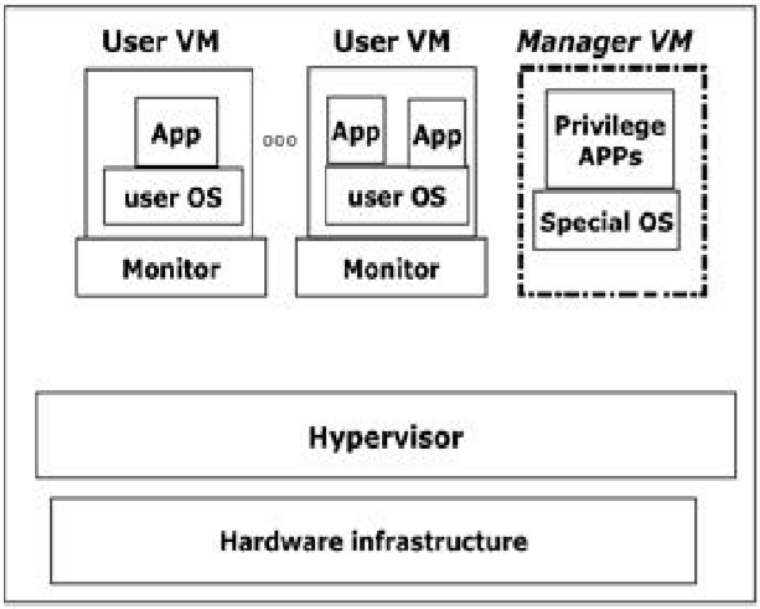

Рисунок 3.4 – Віртуалізація на основі гіпервізора

У цій архітектурі привілейовані розділи переглядають віртуальні машини та керують ними. Цей підхід встановлює найбільш контрольоване середовище та може використовувати додаткові засоби безпеки, такі як системи виявлення вторгнень [4]. У кваліфікаційній роботі реалізувано метод віртуалізації на основі гіпервізора для консолідації та об'єднання ресурсів. Цей метод віртуалізації має деякі інші великі переваги у високопродуктивних обчисленнях, а також процес міграції, масштабованість гіпервізора та багато іншого [15].

3.1.2 Розробка гіпервізора для контролю віртуального обчислювального оточення

Гіпервізор, також відомий як монітор віртуальних машин (VMM), є програмною платформою віртуалізації. Гіпервізор дозволяє кільком операційним системам працювати на одному апаратному вузлі. Кожна операційна система має власний процесор, пам'ять і ресурси собі [9]. Гіпервізор управляє розподіленими процесорами та ресурсами, які потрібні для кожної операційної системи, у свою чергу, та гарантує, що гостьові операційні системи або віртуальні машини не можуть порушувати одна одну.

У віртуальному середовищі монітор віртуальних машин (VMM) є основною програмою управління з максимальним рівнем привілеїв і VMM керує однією або декількома операційними системами. Гіпервізори діляться на два типи: Власні гіпервізори: програмні системи, що працюють безпосередньо на хост-програмному забезпеченні як апаратне управління та монітор гостьової операційної системи.

Таким чином, гостьова операційна система працює на іншому рівні над гіпервізором. Це традиційна реалізація архітектур віртуальних машин. Гіпервізори хоста: програмні програми, що працюють у нормальному операційному середовищі. Враховуючи рівень гіпервізора, окремий програмний рівень, гостьові операційні системи, таким чином, працюють на

третьому рівні вище за апаратне забезпечення.

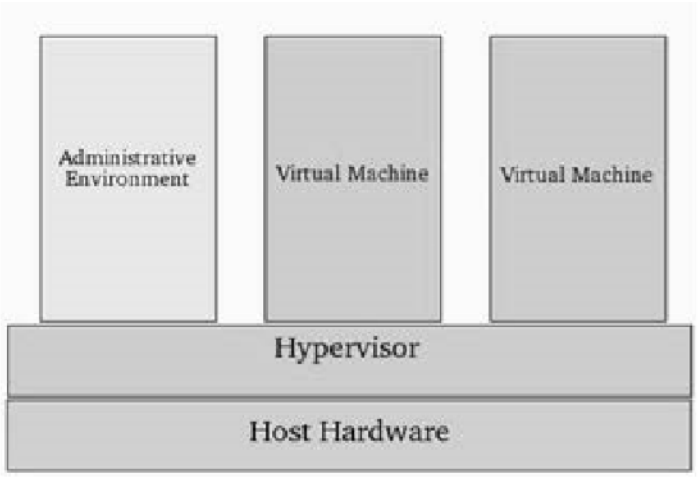

Рисунок 3.5 – Архітектура гіпервізора (VMM)

Було розроблено та впроваджено монітор віртуальних машин (VMM), який дозволяє віртуалізувати загальну пам'ять багатопроцесорної кластерної машини [2]. Ця функція значно полегшує використання кластера. Наприклад, він дозволяє паралельним програмам із загальною пам'яттю для багатопроцесорних систем працювати на кластерах без зміни додатків. Крім того, операційні системи, що підтримують мультипроцесори (наприклад, Linux) можуть бути встановлені у віртуальних машинах з невеликими змінами. Гіпервізор (VMM) забезпечує повний контроль над апаратною машиною і створює віртуальні машини, кожна з яких поводиться як повна фізична машина, яка може працювати під керуванням своєї власної операційної системи (рисунок 3.6).

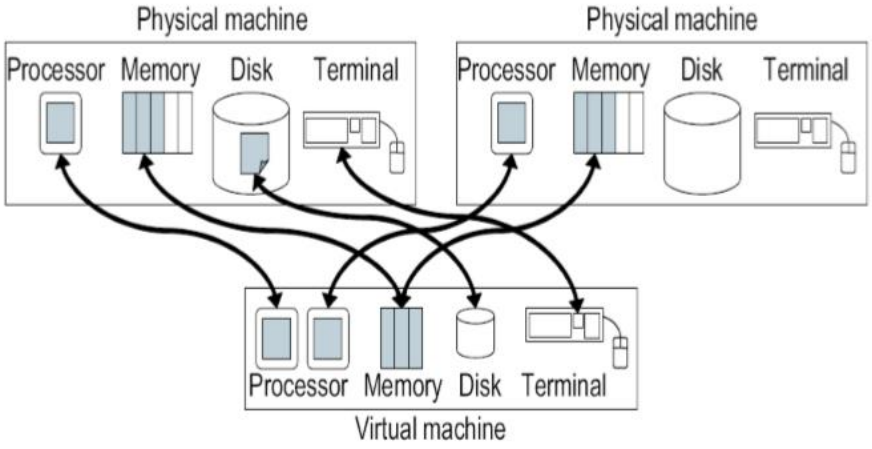

Рисунок 3.6 – Відображення між ВМ та ФМ

Більшість нинішнього інтересу до віртуалізації обертається навколо віртуальних серверів, частково тому, що віртуалізація серверів може призвести до значної економії коштів. Вираз VM відноситься до програмного забезпечення комп'ютера, яке, як і фізичний комп'ютер, запускає операційну систему та програми. Операційна система на віртуальній машині називається гостьовою операційною системою. Крім того, існує рівень управління, званий Virtual machine monitor або Manager (VMM), який створює та керує всіма віртуальними машинами у віртуальному середовищі [6, 9, 7].

3.1.3 Реалізація DVMM для інтеграції розподілених ресурсів віртуального обчислювального оточення

Популярні монітори віртуальних машин, такі як Xen і VMware, призначені в основному для віртуалізації на одному фізичному вузлі. У нашій роботі ми реалізували монітор віртуальної машини для розподіленого кросвузла та для інтеграції ресурсів обчислювальної системи.

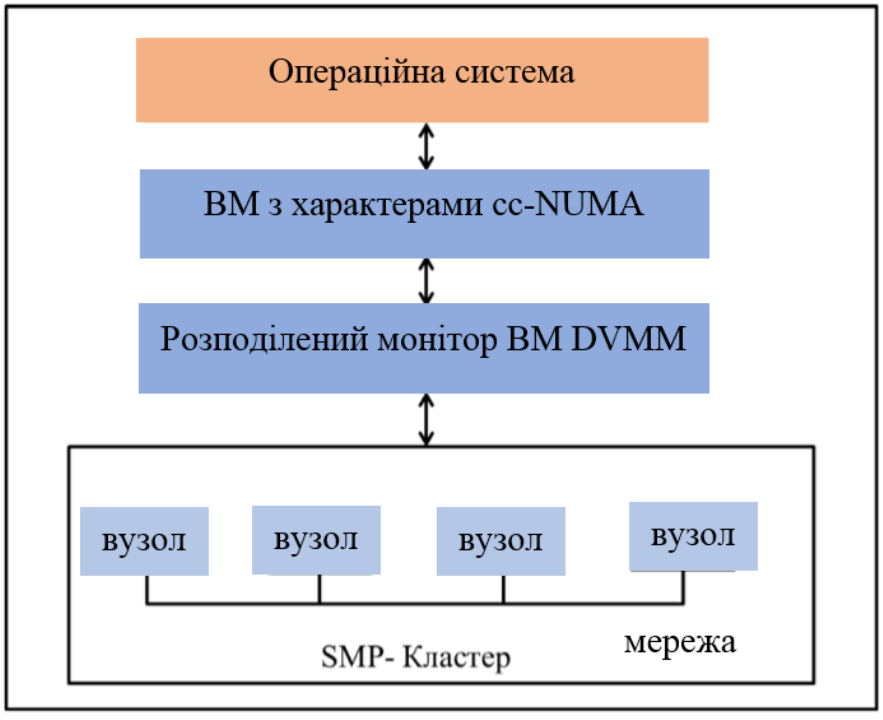

Рисунок 3.7 – Системна архітектура DVMM

Було створено DVM над кластером SMP, що базується на апаратній віртуалізації. DVM є заміною для всіх розподілених VMM у кожному вузлі. Усі VMM є симетричними. Основна мета DVM-приховати розподілені апаратні атрибути для забезпечення SSI в кластері SMP та підтримки єдиної операційної системи для прозорої роботи в кластері [15].

3.2 Інтеграція та консолідація програмних продуктів

Консолідація та спільне використання послуг, розроблених як економічно ефективний інструмент, забезпечують надання цих послуг через мережу в міру необхідності. Ця можливість залежить від досягнень у галузі стандартизації, оптимізації, сервісної орієнтації та віртуалізації. Зокрема, віртуалізація підтримує динамічні пули ресурсів, такі як сервери та системи зберігання даних. Існує два основних типи консолідації: фізична та логічна. Під час виконання фізичної консолідації сервери фізично переміщаються у єдиний інформаційний центр, та їх подання у комп'ютерної мережі може бути незмінним, що означає, що одному фізичному серверу може бути призначено кілька мережевих служб. Логічна консолідація – це інтеграція серверних обчислювальних ресурсів без фізичного переміщення в єдиний мережевий ресурс. Наразі починають з'являтися необхідні рішення. Зокрема, вони дозволяють керувати мережею з кількох фізичних серверів як єдиним пристроєм.

Сформулюємо комплекс вимог, яким має відповідати процедура консолідації, щоб її використання було економічно виправдане:

- доступність додатків користувача після консолідації не повинна змінюватися;
- робота користувачів із додатками не повинна ускладнюватись;
- інформаційна безпека має негативно позначатися;
- необхідно забезпечити значне зниження загальної вартості володіння інформаційною системою.

Найпростіший приклад застосування консолідації – це об'єднання двох чи більше компаній із різними інформаційними системами. У цьому випадку консолідація дозволяє заощадити кошти за рахунок усунення елементів, що дублюють, та ефективного використання наявних ресурсів. Інший приклад — консолідація галузевих інформаційних систем. Тут важлива логічна консолідація, яка дозволить центральному офісу контролювати роботу інформаційних систем філій.

При перенесенні ресурсомістких додатків на централізовані сервери можна знизити вартість робочого місця за рахунок використання більш дешевих комп'ютерів, на яких виконується суворо фіксований набір програм, в ідеалі просто браузер. Об'єднання серверів у єдиний дата-центр дозволяє посилити їх захист та підвищити надійність розрахунків. Це стосується як фізичної безпеки (організація доступу до обладнання), так і мережевої безпеки (віддалений доступ). Те саме стосується і надійності, оскільки в централізованому обчислювальному центрі легше забезпечити оптимальні умови навколишнього середовища, надійне енергопостачання, постійне резервне копіювання та швидке планове технічне обслуговування, ніж у розподіленій системі.

У кожному конкретному випадку ви можете використовувати власні методи та стратегії консолідації, що дозволило вам оптимізувати її. Проте існує універсальне рішення-віртуалізація, яка дозволяє змінювати фізичну і логічну конфігурацію інформаційної системи.

Для досягнення ефективної консолідації важливо розуміти, які ресурси необхідні для запуску програми з часом. Ці дані включають характеристики використання додатків та пікові потреби у ресурсах. Використовуючи цю інформацію, віртуальні машини, на яких запущено програми, можуть бути запущені з використанням фізичних обчислювальних ресурсів та ресурсів зберігання. Неправильне виконання віртуальних машин може призвести до проблем з поточною продуктивністю, коли вимоги до ресурсів програми значно змінюються з часом [7]. В даний час багато підприємств

використовують різні стратегії для вирішення цієї проблеми: Консолідація для скорочення кількості серверів або систем зберігання даних, необхідних підтримки IT-сервісів; І віртуалізація, яка допомагає відокремити ці послуги від конкретних реалізацій і представити їх у більш гнучкій формі. У цій роботі розглядаються методи інтеграції та консолідації програмних продуктів у розподіленому обчислювальному середовищі. Також був розроблений комплекс програм для створення операційного середовища, що дозволив підвищити загальну продуктивність гетерогенних програмно-апаратних комплексів у середньому на порядок за рахунок адаптації архітектури кожної окремої ВМ до конкретного додатку [6, 9].

3.3 Імітаційне моделювання у розробленому обчислювальному середовищі

Для віддаленого запуску програми у розробленому обчислювальному середовищі на ньому повинні виконуватись спеціальні процеси GRAM (Grid Resource Allocation Manager). Ви можете використовувати його для додавання, відстеження та скасування завдань у системі. Користувальницькі програми формують запити на втиснення в спеціальний RSL (Resource Specification Language) [6,7]. GRAM включає два демони: Globus-gatekeeper-daemon і бібліотека GSI, що використовуються для захисту передачі даних і координації обслуговування Globus-job-manager-демон, що реалізує планування завдань і функціональність для передачі даних або файлів в управління ресурсами (рисунок 3.8).

Для запуску програми UGE та GRAM були інтегровані в систему, реалізувавши програмний інтерфейс DRAMA (Distributed Resource Management Application API), який дозволяє запускати програму в системі за допомогою скриптів. GridFTP – це сервіс для швидкої та ефективної передачі даних за протоколом gsiftp (особливо у великих обсягах). У величезній системі існує особлива структура даних – паралельне середовище, що

дозволяє враховувати особливості додатків, що використовують технології паралельного програмування.

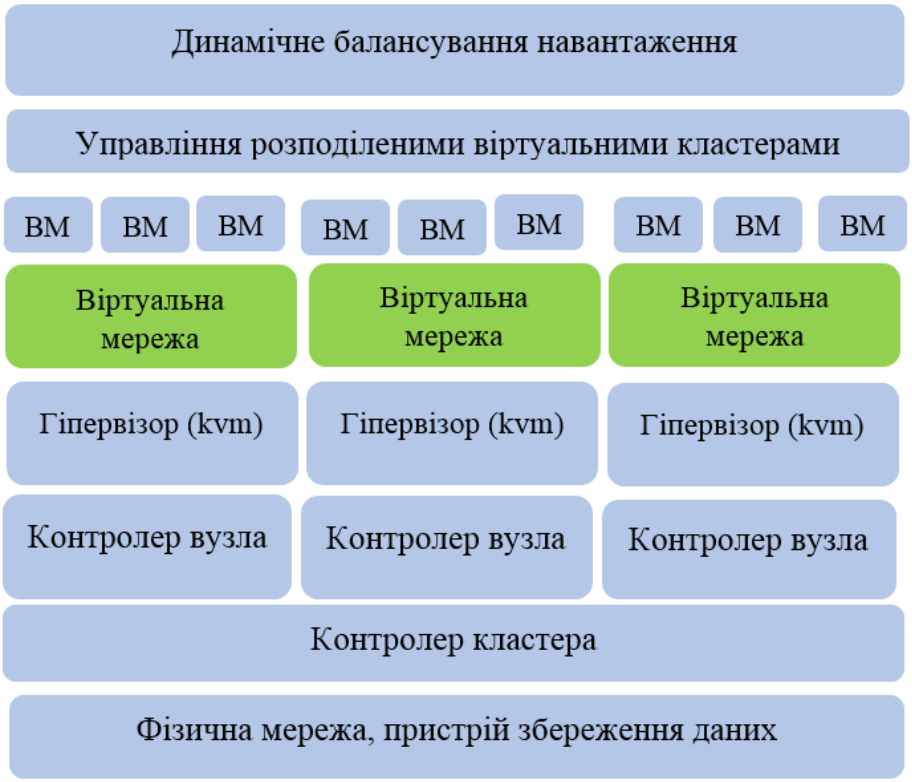

Рисунок 3.8 – Розроблена віртуальна обчислювальна система

Для вирішення паралельних завдань UGE та Mosix були інтегровані з додатковим пакетом сценаріїв та бібліотекою Mpich2 (рисунок 3.8).

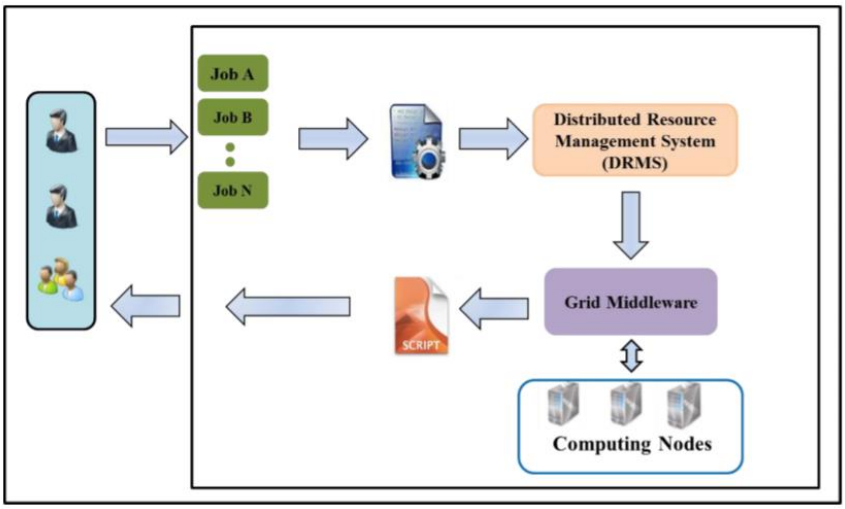

Рисунок 3.9 – Модель запуску завдань з інтерфейсом DRMAA

Для успішного виконання цих завдань ми проаналізували та розробили методику управління ресурсами у розподіленому обчислювальному середовищі для запуску конкретних ресурсомістких додатків. Використовуються різні програмні продукти, такі як Univa Grid Engine, Mosix та Globus Toolkit. Таким чином, віртуальні машини можуть вільно взаємодіяти один з одним через віртуальну мережу та автоматично налаштовувати мережу. І тут віртуальна машина може розподілятися (навіть динамічно) між вузлами.

3.4 Стандартні тести продуктивності та їх застосування

В даний час широко використовуються тестові програми, взяті з різних предметних областей і являють собою або модельні, або реальні промислові програми. Ці тести дозволяють оцінити продуктивність комп'ютера в реальних завданнях і отримати найповніший аналіз того, як комп'ютер працює з конкретним додатком [5]. Найбільш поширеними тестами, заснованими на цьому принципі, є: набір з 24 циклів Лівермору (Livermore Fortran Core, LFK) і пакет паралельних NAS Benchmark (NPB), що включає дві групи тестів, що відображають різні аспекти реальних обчислювальних програм гідродинаміки. Тести NAS є альтернативою Linpack, оскільки вони відносно прості і водночас містять значно більше обчислень, ніж Linpack або LFK.

Однак при всій різноманітності тестових програм неможливо дати повну картину того, як працює комп'ютер у різних режимах. Linpack - це програма, призначена для вирішення системи лінійних керування алгебри з щільною матрицею з виділенням основного елемента по рядку. Було розроблено фіксований набір тестів з метою оцінки продуктивності комп'ютерної системи на реальних задачах. Найвідоміший з них-LINPACK. Однак у LINPACK є суттєвий недолік: програма розпаралелена, тому неможливо оцінити ефективність комунікаційної складової суперкомп'ютера.

LINPACK був обраний тому, що він широко використовується і номери виконання доступні практично для всіх відповідних систем. Тест складається з низки простих синтетичних завдань: ядер (kernel Benchmark) та псевдододатків (application Benchmark), що емулюють обчислення за реальними завданнями (зокрема, в галузі обчислювальної гідродинаміки). Для кожного ядра еталонні тести NAS визначають п'ять класів зростаючих робочих навантажень, названих S, W, A, B і C. клас S зручний для тестування і відповідає системам з максимальною кількістю процесорів 4 клас W зручний для робочих станцій настільних комп'ютерів. Клас W підходить для систем із одним процесором, Клас A для систем із 32 процесорами. Класи і зручні для багатопроцесорних систем. Клас для систем з 32-128 процесорами і клас для систем до 256 процесорів. У термінології NPB ядра і програми можуть виконувати обчислення в певних класах завдань): "приклад коду", "клас A", "клас B", "Клас C", "клас D". У NPB клас визначається як розмірність основних наборів даних, використовуються в тесті. Іншими словами, клас A-це маленькі матриці, Bвеликі, C-дуже великі, D-величезні.Наприклад, для тесту LU-декомпозиції це буде розмірність вихідної матриці: 12^3, 64^3, 102^3, 408^3 для кожного з перерахованих вище класів відповідно [6].

Крім ядер, пакет NPB пропонує низку псевдо-додатків, які емулюють роботу реальних програм з обчислювальної гідродинаміки. Відмовитися від використання реальних програм на користь псевдо-додатків вирішено з кількох причин:

- збереження цього вихідного коду в таємниці;
- полегшення роботи з вихідним кодом щодо портування на інші архітектури;
- легкість додавання нових компонентів;
- легкість масштабування коду для великих розмірів.

Прикладні алгоритми використовують описані вище ядра в тій чи іншій формі і, зрештою, зводяться до вирішення особливого типу СЛАР (системи

лінійних рівнянь алгебри) (втім, як і переважна більшість обчислювальних завдань).

Основна частина обчислювального часу у таких завданнях витрачається рішення СЛАР. Тому застосування може бути описано як ітераційні методи розв'язання систем лінійних рівнянь.

Існує три такі програми: LU, SP, BT. Розглянемо набір тестів NAS Benchmarks для модельних завдань.

LU (Lu Solver). Тест виконує обчислення, що відносяться до певного класу алгоритмів (INS3D-LU за класифікацією центру Еймса НАСА), в яких вирішується система рівнянь із рівномірно розрідженою трикутною блочною матрицею 5x5. Цей тест вимірює затримку мережі та пропускну здатність кешу команд.

SP (скалярна пента-діагональ). Тест вирішує кілька незалежних систем скалярних рівнянь-пента-діагональних матриць з переважанням недіагональних членів. Цей тест вимірює пропускну спроможність пам'яті.

БТ (блоково трьохдіагональних). Рішенням низки незалежних систем рівнянь є блочні тридіагональні матриці 5x5 з переважанням недіагональних елементів.

Цей тест вимірює пропускну здатність мережі та команди кешу. Алгоритми SP (від скалярного п'ятидіагонального) та БТ (від блочного трьохдіагонального) аналогічні: розв'язання трьох незв'язаних систем рівнянь (в напрямках x, y та z) методом багатосекційної схеми. Різниця між програмами полягає в структурі матриці: для SP це п'ятидіагональна матриця, а для BT-блочна тридіагональна матриця з розміром блоку 5x5. Цей метод добре розпаралельний і використовується для оцінки продуктивності системи з плаваючою комою і забезпечує оптимальне навантаження на мережу, але вимагає, щоб кількість вузлів у кластері була квадратом цілого числа. Тест SP більш чутливий до затримки мережі та використовується для оцінки продуктивності інтерстиціальних з'єднань.

Додаток LU вирішує систему рівнянь з рівномірною розрядженою

блоковою структурою (5x5) методом симетричної послідовної надрелаксації (symmetric successive over-relaxation – SSOR), до якої наводять тривимірні рівняння Нав'є-Стокса. Для розподілу даних цьому додатку потрібна кількість вузлів, кратних ступеня двійки. Особливістю цього тесту є його критичність на час передачі дуже малих обсягів даних між вузлами (розмір повідомлення, що передається в цьому тесті становить 40 байт). Приналежність до певного класу кожного із додатків визначає розмірність системи рівнянь. Нижче представлена зведена таблиця належності ядер та додатків до певного класу:

Таблиця 3.1 – Належності ядер

Тест	Клас A	Клас B	Клас C	Клас D
EP	2^{28}	2^{30}	2^{32}	2^{36}
MG	256^3	256^3	512^3	1024^3
CG	14000	75000	$1.5x10^5$	$1.5x10^6$
FT	256^2x128	256^2x512	512^3	1024^2x2048
IS	2^{23}	2^{25}	2^{27}	
LU	64^3	102^3	162^3	408^3
SP	64^3	102^3	162^3	408^3

Особливістю реалізації тесту є необхідність компіляції програми кожного класу завдань. Це спричинено тим, що стандарт мови fortran-77 не підтримує динамічний розподіл пам'яті. Результати NPB виходять у мільйонах дій на секунду. Завдання було обрано після оцінки безлічі великих прикладних програм обчислювальної гідродинаміки, що вирішуються в NASA.

Завдання пакета NPB містять значно більше обчислень, ніж ті тести, що використовувалися раніше, наприклад такі, як Livermore Loops або LINPACK, тому вони більш прийнятні для оцінки паралельних машин. З іншого боку, ці завдання щодо прості, що дозволяє ставити ці завдання нових

обчислювальних системах без значних зусиль і затримок [8]. У цьому розділі будуть розглянуті проблеми встановлення та налаштування ПЗ для розрахунків у розподіленому обчислювальному середовищі. І буде проведено тестування обчислювального середовища на різних завданнях [4].

3.4.1 NAS Parallel Benchmarks

Насамперед ми хотіли б перевірити ефективність розробленого обчислювального середовища, оцінити перспективи його використання для реальних завдань. Тести паралельних бенчмарків NAS найкраще підходять для цього завдання. NAS Parallel Benchmarks – безліч тестів продуктивності для перевірки можливостей високопаралельних обчислювальних систем. Вони були розроблені в NASA Advanced Supercomputing (NAS) Division.

У цьому наборі є такі тести як:

- LU – розкладання матриць;
- EP – генерація незалежних нормально розподілених випадкових величин;
- FT – швидке перетворення Фур'є;
- IS – сортування малих цілих чисел;
- DT – тести націлені оцінку швидкості передачі.

Тести розраховані на обчислювальні комплекси різного масштабу. Існує кілька класів завдань для тестів:

- S – невеликі завдання для тестових цілей;
- W – завдання окремих робочих станцій;
- A, B, C – стандартні задачі;
- D, E, F – великі завдання.

3.4.2 CRYSTAL

CRYSTAL – це програма, призначена в основному для розрахунків трьохвимірних, двохвимірних та одновимірних фігур з використанням трансляційної симетрії, але вона також може бути використана для одиночних об’єктів. Програма складається з двох модулів: crystal та properties. Програма crystal призначена для виконання SCF-розрахунків, оптимізації геометричних та частотних розрахунків для структур, заданих у вхідних даних. Пакет CRYSTAL містить наступні паралельні двійкові файли: Pcrystal, Pproperties, MPPcrystal, Pcrystal (parallel crystal) і Ppproperties (parallel properties) є реплікованими версіями даних і властивостей, в той час як MPPcrystal (massively parallel crystal) розподіляє дані та завдання по ядрам більш ефективно, ніж Pcrystal, і особливо підходить для великих випадків елементарних величезними вимогами до пам'яті. Pcrystal та MPPcrystal істотно відрізняються тим, як вони обробляють дані у взаємному просторі, а також алгоритмами, що використовуються для діагоналізації матриці Фока та обробки власних векторів.

З іншого боку, дані в реальному просторі, такі як обчислення одно- та двоелектронних інтегралів, обробляються Pcrystal та MPPcrystal з використанням одних і тих самих алгоритмів та стратегії розпаралелювання. Паралелізм у кристалі ґрунтується на бібліотеках MPI. Pcrystal та Ppproperties не пов'язують жодні інші бібліотеки. MPPкристал покладається на використання масивних паралельних бібліотек. Як Pcrystal, і Ppproperties очікують зчитування вхідних даних із файлу під назвою INPUT у каталозі тих дисків, де програми зберігають тимчасові дані (Fortran units). Функція Pcrystal була протестована в нашому обчислювальному середовищі та підходить для малих та середніх систем.

Програма CRYSTAL буде встановлена на диску з програмним забезпеченням, доступним всім користувачам кластера в нашому розподіленому обчислювальному середовищі. Отже, щоб встановити

CRYSTAL, виконайте такі дії:

1) Завантажити виконувані файли: повна ліцензія (послідовна та паралельна версії) (Макс 10000 атомів / комірка) / демо-ліцензія (послідовна версія) (Макс 10 атомів / комірка).

2) Завантажити скрипти для запуску програм.

3) Завантажте графічний пакет Крім того, використовуйте CRYSPLOT новий веб-графічний інтерфейс для побудови обчислюваних властивостей.

Для тестування кристала у нашій системі здійснюється наступними кроками.

Крок 1. Послідовне виконання з використанням скриптів, для тестування програми з вхідними даними тестових випадків поставляється, зробіть з $CRY2K17_ROOT тест каталогу: mkdir test_first

Крок 2. Для того, щоб кристал і властивості зчитували вхідний файл з тестового набору даних, ви повинні встановити ці дві змінні; setenv CRY2K17_INP "$CRY2K17_ROOT/test_cases/inputs" setenv CRY2K17_PROP "$CRY2K17_ROOT/test_cases/inputs" або (bash) export CRY2K17_INP=$CRY2K17_ROOT/test 7_ROOT/test_cases/inputs

Крок 3. Щоб запустити тест runcry17 test

Крок 4 Паралельне виконання з використанням скриптів set MPIDIR = /opt/openmpi-1.6.3/bin set MPIBIN = mpirun

3.4.3 OpenFoam

OpenFOAM (Open Source Field Operation and Manipulation) – це безкоштовний пакет для чисельного моделювання завдань механіки суцільних середовищ [9]. Він використовується багатьма різними організаціями в усьому світі, як комерційними, і некомерційними. Набір бібліотек надає інструменти для вирішення систем рівнянь у приватних похідних як у просторі, так і в часі. Написано мовою Сі ++. OpenFOAM.

Результати виконання завдання на OpenFoam (icoFoam) у віртуальній обчислювальній системі показані на рисунку 3.10. Цей пакет буде встановлений на диск із програмним забезпеченням, доступним для всіх користувачів кластера в нашому розподіленому обчислювальному середовищі. Він монтується у папці /opt/. Отже, щоб встановити OpenFOAM, виконайте такі дії:

1) у вікні терміналу, додати OpenFOAM до списку розташування в сховищі для apt, щоб шукати;

2) оновити список пакетів APT для врахування нового розташування сховища sudo apt-get update;

3) встановлення OpenFOAM (211 у назву відноситься до версії 2.1.1): sudo apt-get install openfoam211 4. Встановити Paraview sudo apt-get install para view open foam 3120.

Для того, щоб використовувати встановлений пакет Open FOAM, потрібно виконати наступні кроки:

а) Відкрити Bashrc файл у домашньому каталозі користувача gedit ~/.bashrc

б) У нижній частині цього файлу додати рядки source /opt/openfoam211/etc/bashrc.

в) відтестувати програми icoFoam з пакета OpenFOAM, icoFoam -help "Usage" має з'явитися повідомлення, що установки та конфігурації користувача завершені.

Для запуску послідовних розрахунків необхідно виконати три кроки.

Крок 1. Створити директорію проекту в $HOME/OpenFOAM каталозі з ім'ям <користувач> -2.1.1 і створити каталог з ім'ям, запустивши в ньому, mkdir -p $FOAM_RUN.

Крок 2. Скопіювати папку приклади в OpenFOAM каталог. Якщо змінні середовища OpenFOAM встановлені правильно, то наступна команда приведе до результату: cp -r $FOAM_TUTORIALS $FOAM_RUN

Для прикладу розглянемо перший випадок стисканого ламінарного

потоку в порожнині: cd $FOAM_RUN/tutorials/incompressible/icoFoam/cavity /decomposeParDict Задає параметри для розпаралелювання задачі:

1 параметр number of Subdomains повинен відповідати числу паралельних процесів;

2 параметр метод задає метод для розбиття області;

3 параметр (блок) simpleCoeffs використовується у разі методу «simple» («n(a b c)» вказує, як саме розбивати область (a * b * c = number of Subdomains)).

Директорія '0' містить початкові умови завдання, фізичні властивості змінних вказані в constant/transportProperties. Ми запустимо розрахунки з Δt=10-5.

Перед запуском розрахунків необхідно підготувати дані. Для цього переходимо в папку із завданням та виконуємо blockMesh. Дана команда створить сітку, для контролю можна виконати checkMesh. Тепер можна запускати власне розрахунки. Для розрахунків використовується програма (solver) pisoFoam із пакету OpenFOAM piso Foam. Наші обчислення завершилися без помилок. Після цього можна запускати програму візуалізації ParaView за допомогою скрипта paraFoam (результат показаний на малюнку).

Для запуску паралельних розрахунків необхідно виконати наступні кроки.

Крок 1. Спочатку виконуємо дії, що й для послідовних розрахунків: blockMesh checkMesh

Крок 2. Тепер потрібно провести декомпозицію завдання. Для цього нам потрібно створити файл decomposeParDict у папці системи нашого прикладу, в якому будуть вказані параметри розпаралелювання завдання. Його зразковий вміст: numberOfSubdomains 2; метод simple; simpleCoeffs { n (2 1 1); delta 0,001; } Параметр «numberOfSubdomains» повинен відповідати числу паралельних потоків, які ми запускатимемо (він вказує число підзавдань, на яке ми розбиваємо наше завдання). Параметр «method»

визначає метод, з допомогою якого буде розбито завдання. Наприкінці ми визначаємо, як саме буде поділено наше завдання (на 2 області по осі x у даному випадку). Тепер запускаємо програму decomposePar, яка і проведе розбиття: decomposePar

Тепер можна запускати розрахунки qsub <скрипт>

Вміст скрипту для запуску: #!/bin/bash ~/ openfoam.sh <директорія_з_даними> pisoFoam

Після виконання розрахунків запускаємо програму reconstrucPar: reconstructPar

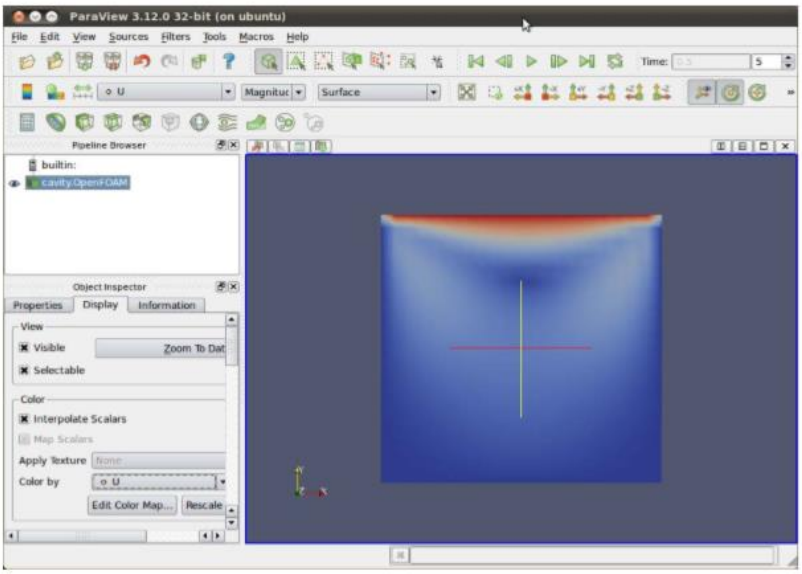

Рисунок 3.10 – Перегляд запуску програми у ParaView

Тепер ми можемо переглянути результати за допомогою ParaView, paraFoam, що готує дані для ParaView і запускає цю програму (рисунок 3.10).

3.5 Дослідження систем паралельного програмування у розробленому обчислювальному середовищі

Паралельна обробка – це метод, при якому багато дрібних завдань вирішують одну велику проблему, яка стала ключовою технологією, що

дозволяє використовувати її в сучасних обчисленнях. В останні кілька років спостерігається дедалі ширше визнання та впровадження паралельної обробки даних. Цьому сприяли дві великі події: масово-паралельні процесори (MPPs) та широке використання розподілених обчислень. Машини типу MPPs, мабуть, є найпотужнішими комп'ютерами у світі. Ці машини об'єднують від кількох сотень до кількох тисяч процесорів в одній великій шафі, підключеній до великого обсягу пам'яті, і пропонують величезну обчислювальну потужність. Але вартість таких машин дуже висока та дорога. Друга велика розробка, що впливає рішення наукових завдань, – це розподілені обчислення. Розподілені обчислення – це процес, в якому набір комп'ютерів, з'єднаних мережею, використовується колективно для вирішення однієї великої проблеми.

Ідея використання таких кластерів або мереж робочих станцій для вирішення паралельного завдання стала дуже популярною, оскільки такі кластери дозволяють людям використовувати існуючі та в основному простоюючі робочі станції та комп'ютери, дозволяючи їм виконувати паралельну обробку без необхідності купувати дорогий суперкомп'ютер. Оскільки дедалі більше організацій мають високошвидкісні локальні мережі, які з'єднують кілька робочих станцій загального призначення, об'єднані обчислювальні ресурси можуть перевищувати ємність одного високопродуктивного комп'ютера. Спільним між розподіленими обчисленнями та MPP є концепція передачі повідомлень. Для будь-якої паралельної обробки дані повинні обмінюватись між взаємодіючими завданнями. Бібліотеки передачі повідомлень дозволили переносити паралельний алгоритм на платформу паралельних обчислень. PVM та MPI були найуспішнішими з цих бібліотек. В даний час PVM і MPI є інструментами, що найчастіше використовуються для паралельного програмування. У нашій роботі ми проаналізували MPI програмування з MOSIX і без нього, щоб оцінити можливості паралельних обчислень у розробленому середовищі. Ці випробування проводилися 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, а також запущено програму для визначення часу затримки при синхронізації процесів MPI. У цьому прикладі виконується тест зв'язку MPI (час затримки повідомлення). MPI-0 відправляє 1байтове повідомлення MPI-1, витрачаючи час очікування відповіді з-поміж них. Після цього виконується синхронізація кожного повторення, а наприкінці обчислюється середній час очікування. Ці випробування проводилися під контролем операційного середовища MOSIX, з використанням і без використання схеми запобігання міграції процесів. Мережева затримка – це час, який потрібен для того, щоб щось, відправлене з вихідного хоста, досягло кінцевого хоста. На рисунку 3.11 наведено результати тестування часу затримки в дорозі туди та назад.

Рисунок 3.11 – Тестування часу затримки в дорозі туди і назад з і без MOSIX

Час затримки в дорозі туди і назад – це те, скільки часу потрібно для запиту, відправленого з джерела до пункту призначення, і для відповіді, щоб повернутися до джерела. У принципі, затримка у кожному напрямі плюс час обробки [9].

3.6 Аналіз продуктивності розробленої системи

Після всіх конфігурацій ми проаналізували продуктивність нашого віртуального розподіленого середовища (таблиця 3.2).

Таблиця 3.2 – Результати тестування

Кількість VMs	Час виконання		
	PCrystal	OpenFOAM	NPB
1	5,32	4,01	4,86
2	4,63	3,16	2,22
4	3,51	2,87	0,67
8	2,87	1,74	0,17

У системі ми керуємо різноманітними ресурсами, створюємо віртуальні обчислювальні кластери під управлінням гіпервізора та об'єднуємо обчислювальні ресурси в єдину обчислювальну систему. Наша система працює із реальними завданнями (рисунок 3.12).

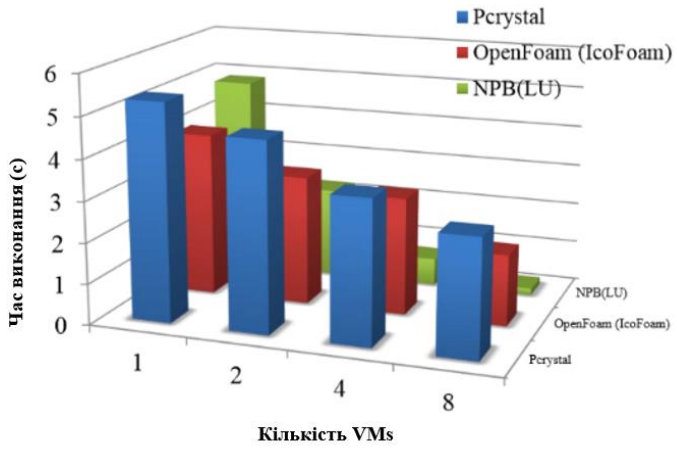

Рисунок 3.12 – Результати тесту продуктивності запуску додатків

Ми проаналізували тестові приклади, такі як LU-тести з NAS, розрахунки великого пакету OpenFOAM-відкритого пакета для чисельного моделювання завдань механіки.

ВИСНОВКИ

У кваліфікаційній роботі представлені основні поняття та особливості Grid-обчислень, хмарних обчислень та технологій віртуалізації. Grid-системи забезпечують високе навантаження обчислювальних ресурсів, розподіляючи одне складне завдання між кількома обчислювальними вузлами, а хмарні обчислення йдуть шляхом виконання кількох завдань одному сервері як віртуальних машин.

Описано функції та основні причини використання Grid- та хмарних обчислень. Тоді як Grid переважно використовується для вирішення завдань на певний (обмежений) проміжок часу, а хмарні обчислення переважно орієнтовані на надання послуг. Grid та хмарні обчислення доповнюють один одного. Grid-інтерфейси та протоколи можуть забезпечувати взаємодію між хмарними ресурсами або інтеграцію хмарних платформ.

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

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Мартинцов А.Ф., Коваленко А.А., Ситник О.В. Особливості гетерогенних обчислювальних ресурсів. Проблеми інформатизації. Тези доповідей 11 міжнародної науково-технічної конференції, Т.1, Баку – Бельсько-Бяла – Харків, 15-16 листопада 2023.– С.71.

2. Kovalenko A..,MiroshnychenkoR.,Martyntsov A. Distributed computing systems based on the use of Grid technologies. Системи управління, навігації та зв'язку, 2021, № 1(71) С. 101-104.

3. D.A. Bacigalupo, J. van Hemert, et al, "Managing dynamic enterprise and urgent workloads on clouds using layered queuing and historical performance models," Simulation Modelling Practice and Theory. Vol 19, 2021, pp. 1479-1495.

4. D.J. Magenheimer., W.Thomas, C.V.Blades. Optimized paravirtualization for the Itanium processor family. In Proceedings of the 3rd Virtual Machine Research and Technology Symposium, pages 73–82, 2023. The Grid2023 Project. The Grid2023 Production Grid: Principles and Practice. iVDGL: Technical Report. – 2023. – 42 p.

5. Ian Foster. What Is The GRID? A Three Point Checklist. GRID Today, July 22, 2002: Vol. 1 No. 6, http://www.gridtoday.com/02/0722/100136.html. http://www.gridclub.ru/library/publication.2004-11-29.5830756248

6. SDK доповноеної реальності [Електронний ресурс] – Режим доступу до ресурсу: https://evergreens.com.ua/ru/articles/web-ar-tools-overview.html

7. Огляд Vuforia SDK [Електронний ресурс] – Режим доступу до ресурсу: https://docs.unity3d.com/2017.2/Documentation/Manual/vuforia-sdk-overview.html

8. Object & Scene Tracking Augmented Reality [Електронний ресурс] – Режим доступу до ресурсу: https://www.wikitude.com/augmented-reality-object-scene-recognition/

9. Що таке Colaboratory [Електронний ресурс] – Режим доступу до ресурсу: https://colab.research.google.com/notebooks/welcome.ipynb?hl=ru#scrollTo=5fCEDCU_qrC0

10. Google Colab - ваш робочий простір на Python в хмарному середовищі [Електронний ресурс] – Режим доступу до ресурсу: https://www.machinelearningmastery.ru/google-colab-your-python-workspace-on-cloud-c3aed424de0d/

11. Просте введення в Pytorch для нейронних мереж [Електронний ресурс] – Режим доступу до ресурсу: https://www.machinelearningmastery.ru/an-easy-introduction-to-pytorch-for-neural-networks-3ea08516bff2/

12. Початок роботи з набором даних COCO [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/getting-started-with-coco-dataset-82def99fa0b8

13. 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.)

14. 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 Programming Languages and Operating Systems, pages 234–244, Nov. 2020.

15. A.V. Bogdanov, M. Dmitriev, Ye Myint Naing, Eucalyptus Open-source Private Cloud Infrastructure, GRID 2010, Proceedings of the 4th International Conference Dubna, June 28- July 3, 2010. Page: 57-63.

16. A.V.Bogdanov, A.A. Lazarev, La Min Htut, Myo Tun Tun, Building User Access System in Grid Environment, Distributed Computing and Grid-Technologies in Science and Education: Proceedings of the 4th Intern. Conference, Dubna, 2019, Pages: 63-69.