ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: **«ДОСЛІДЖЕННЯ МЕТОДІВ ОПТИМІЗАЦІЇ ВИКОРИСТАННЯ РЕСУРСІВ У МЕРЕЖАХ ЦЕНТРІВ ОБРОБКИ ДАНИХ (DATA CENTERS) З ВИКОРИСТАННЯМ ВІРТУАЛІЗАЦІЇ ТА КОНТЕЙНЕРИЗАЦІЇ»**

на здобуття освітнього ступеня магістр
за спеціальності 123 Комп'ютерна інженерія
(код,найменування спеціальності)
освітньо-професійної програми Комп'ютерні системи та мережі
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Андрій ГОЛОСУН
(підпис) *(ім'я, ПРІЗВИЩЕ здобувача)*

Виконав: здобувач вищої освіти гр.КСДМ-61
Андрій ГОЛОСУН
(ім'я, ПРІЗВИЩЕ)
Керівник: Наталія ЛАЩЕВСЬКА
к.т.н., доцент *(ім'я, ПРІЗВИЩЕ)*
Рецензент: _____
науковий ступінь, вчене звання *(ім'я, ПРІЗВИЩЕ)*

Київ 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерної інженерії
Ступінь вищої освіти «Магістр»

Спеціальність 123 Комп'ютерна інженерія
Освітньо-професійна прорама Комп'ютерні системи та мережі

ЗАТВЕРДЖУЮ
Завідувач кафедрою Комп'ютерної інженерії
Наталія ЛАЩЕВСЬКА
(ім'я, ПРІЗВИЩЕ)
"____" _____2023 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Голосуну Андрію Ігоровичу
(прізвище, ім'я, по батькові здобувача)

1.Тема кваліфікаційної роботи: Дослідження методів оптимізації використання ресурсів у мережах центрів обробки даних (Data Centers) з використанням віртуалізації та контейнеризації
керівник роботи Наталія ЛАЩЕВСЬКА к.т.н., доцент
(ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від "19" 10 2023 р. №145
2. Строк подання кваліфікаційної роботи
3. Вихідні дані кваліфікаційної роботи:
3.1. Інтернет ресурси стосовно центрів обробки даних.
3.2. Інтернет ресурси стосовно методів віртуалізації та контейнеризаціїю
3.3. Науково-технічна література.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
4.1. Концепція віртуалізації серверів у центрах обробки даних.
4.2. Міграція віртуальних машин у контейнери для консолідації в центрах обробки даних.
4.3. Оцінка продуктивності фреймворку SEaMLESS.

5.Перелік ілюстраційного матеріалу: *презентація*

6. Дата видачі завдання "19" жовтня 2023р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	**Примітка**
1.	Підбір технічної літератури	20.10.2023р. 27.10.2023р.	Виконано
2.	Концепція віртуалізації серверів у центрах обробки даних	27.10.2023р. 11.11.2023р.	Виконано
3.	Міграція віртуальних машин у контейнери для консолідації в центрах обробки даних	11.11.2023р. 25.11.2023р.	Виконано
4.	Оцінка продуктивності фреймворку SEaMLESS	25.11.2023р. 09.12.2023р.	Виконано
5.	Оформлення роботи, висновки	09.12.2023р. 18.12.2023р.	Виконано
6.	Розробка демонстраційного матеріалу, доповідь	18.12.2023р. 25.12.2023р.	Виконано

Здобувач вищої освіти _____ Андрій ГОЛОСУН
(підпис) *(ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Наталія ЛАЩЕВСЬКА
(підпис) *(ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття ступеня магістр: 78 стор., 9 рис., 2 табл., 25 джерел.

Мета роботи – Забезпечення ефективного використання ресурсів та підвищення продуктивності мереж центрів обробки даних.

Об'єкт дослідження – Методи оптимізації використання ресурсів у мережах центрів обробки даних.

Предмет дослідження – Мережі центрів обробки даних (Data Centers).

Короткий зміст роботи: У роботі представлено концепцію віртуалізації серверів, а також розглянуто два основні типи віртуалізації: віртуалізація на основі гіпервізора та віртуалізація на рівні операційної системи. Представлено підходи до міграції віртуальних екземплярів, що є однією з ключових переваг віртуалізації; загальну інформацію про управління ресурсами в центрі обробки даних, а також підходи до технічного обслуговування в центрі обробки даних, з особливим акцентом на тему оновлення гіпервізорів.

В роботі представлено SEaMLESS - рішення проблеми простою віртуальних машин у центрах обробки даних, фреймворк для трансформації та заміни неактивних віртуальних машин на легкі віртуальні мережеві функції, що дозволяє відключати такі віртуальні машини кількома способами. В рамках SEaMLESS було розроблено нову технологію, яка називається suspend-to-swap, для відключення ВМ і повернення виділеної пам'яті, зберігаючи при цьому швидкий час відновлення.

КЛЮЧОВІ СЛОВА: ЦЕНТРИ ОБРОБКИ ДАНИХ, ОПТИМІЗАЦІЯ, ВІРТУАЛІЗАЦІЯ СЕРВЕРІВ, КОНТЕЙНЕРИЗАЦІЯ, ГІПЕРВІЗОРИ, ХМАРНІ ОБЧИСЛЕННЯ, МІГРАЦІЯ ВІРТУАЛЬНИХ МАШИН, НАДМІРНЕ ВИКОРИСТАННЯ ПАМ'ЯТІ, ВІРТУАЛЬНІ МАШИНИ

ABSTRACT

The text part of the qualification work for obtaining a master's degree: 78 pages, 2 table, 7 figures, 25 sources.

The purpose of the work is ensuring efficient use of resources and increasing the performance of data center networks.

The object of research is Methods of optimizing the use of resources in data center networks.

The subject of research is Networks of data centers.

Summary of the work: The paper presents the concept of server virtualization, and also considers two main types of virtualization: hypervisor-based virtualization and operating system-level virtualization. Approaches to migration of virtual instances are presented, which is one of the key advantages of virtualization; general information on resource management in the data center, as well as maintenance approaches in the data center, with a special emphasis on the topic of upgrading hypervisors.

The work presents SEaMLESS - a solution to the problem of downtime of virtual machines in data centers, a framework for transforming and replacing inactive virtual machines with light virtual network functions, which allows you to disable such virtual machines in several ways. Within SEaMLESS, a new technology called suspend-to-swap has been developed to shut down a VM and reclaim allocated memory while maintaining fast recovery times.

KEY WORDS: DATA CENTERS, OPTIMIZATION, SERVER VIRTUALIZATION, CONTAINERIZATION, HYPERVISORS, CLOUD COMPUTING, VIRTUAL MACHINE MIGRATION, EXCESSIVE MEMORY USAGE, VIRTUAL MACHINES

ЗМІСТ

ВСТУП

Замість безлічі незалежно керованих серверів галузь зробила вибір на користь концентрації більшої частини IT-інфраструктури в одному добре керованому об'єкті - центрі обробки даних (ЦОД). Віртуалізація серверів інкапсулює різнорідні обчислення, сервіси та додатки у віртуальні екземпляри (віртуальні машини або контейнери), консолідуючи в мирній ізоляції більшу частину IT-навантаження на одному фізичному обладнанні. Компанії будь-якого розміру та типу довіряють ефективності та результативності віртуалізації серверів настільки, що приймають ідею запуску своїх основних сервісів у сторонніх багатокористувацьких центрах обробки даних, каталізуючи впровадження хмарних обчислень.

Віртуалізація серверів - це технологія, яка має першочергове значення в сучасних дата-центрах. Віртуалізація надає два ключові механізми, віртуальні екземпляри та міграція, які дозволяють максимально ефективно використовувати ресурси та зменшити капітальні витрати в дата-центрі. У цій кваліфікаційній роботі було визначено та вивчено контекст, в якому традиційна міграція віртуальних екземплярів не забезпечує оптимального використання ресурсів кластера, а саме простої віртуальних машин.

Віртуальні машини, що простоюють, постійно блокують ресурси, які їм призначені, лише для очікування вхідних запитів користувачів. Дійсно, поки вони більшу частину часу простоюють, їх не можна вимкнути, що дозволило б вивільнити ресурси для більш вимогливих сервісів. Для вирішення цієї проблеми було запропоновано SEaMLESS - рішення, яке використовує нову міграцію віртуальних машин у контейнери, що перетворює непрацюючі віртуальні машини Linux на проксі-сервери, які не потребують ресурсів.

SEaMLESS перехоплює нові запити користувачів, коли віртуальні машини вимкнені, прозоро відновлюючи їх виконання при появі нових ознак активності. Крім того, в роботі було запропоновано просту у використанні технологію

вимкнення віртуальних машин на основі радиційної підкачки пам'яті гіпервізора. Завдяки новій технології suspend-to-swap можна звільнити більшу частину пам'яті та процесора, захоплених непрацюючими екземплярами, забезпечуючи при цьому швидке відновлення роботи.

Тобто, дослідження методів оптимізації використання ресурсів у мережах центрів обробки даних є актуальним, оскільки зростає значення цих мереж у сучасному інформаційному суспільстві. Використання віртуалізації та контейнеризації може допомогти ефективніше використовувати ресурси та забезпечити високу продуктивність мереж центрів обробки даних.

1 КОНЦЕПЦІЯ ВІРТУАЛІЗАЦІЇ СЕРВЕРІВ У ЦЕНТРАХ ОБРОБКИ ДАНИХ

1.1 Постановка проблеми

Замість безлічі незалежно керованих серверів галузь зробила вибір на користь концентрації більшої частини IT-інфраструктури в одному добре керованому об'єкті - центрі обробки даних (ЦОД), приклад інфраструктури яких зображено на рисунку 1.1.

Віртуалізація серверів інкапсулює різнорідні обчислення, сервіси та додатки у віртуальні екземпляри (віртуальні машини або контейнери), консолідуючи в мирній ізоляції більшу частину IT-навантаження на одному фізичному обладнанні. Компанії будь-якого розміру та типу довіряють ефективності та результативності віртуалізації серверів настільки, що приймають ідею запуску своїх основних сервісів у сторонніх багатокористувацьких центрах обробки даних, каталізуючи впровадження хмарних обчислень.

Частково економія коштів, пов'язана з центрами обробки даних і віртуалізацією серверів, досягається завдяки більш простому управлінню віртуальними екземплярами порівняно з фізичними машинами, що підвищує продуктивність і ефективність роботи персоналу. Ще одна можливість економії коштів, яка сама по собі породила цілий ряд нових напрямків досліджень, пов'язана з концепцією максимального використання ресурсів в центрах обробки даних. Неповне використання ресурсів є симптомом надмірної закупівлі IT-обладнання, якої можна було б уникнути, а також пов'язане з марнотратством фізичного простору об'єкту - стійок, кімнат, землі - та електроенергії для живлення й охолодження непотрібних приладів. Прикладом останнього є енергетична неефективність звичайних комп'ютерів x86-64, що переважають у сучасних дата-центрах, які пропорційно споживають набагато більше електроенергії, коли працюють з низьким коефіцієнтом використання.

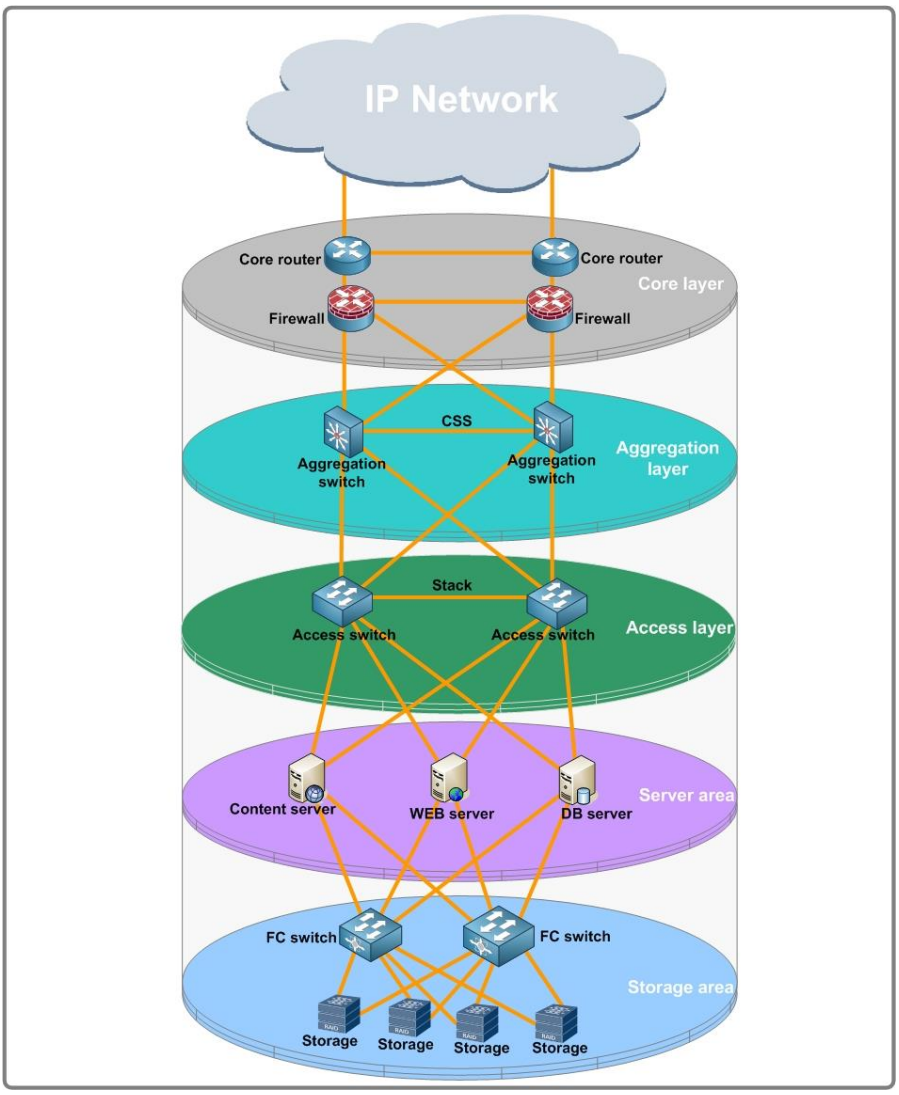


Рисунок 1.1 - Інфраструктура ЦОД

У центрах обробки даних використовується складне програмне забезпечення, так звані менеджери ресурсів, які відповідають за планування розміщення віртуальних машин і контейнерів на фізичних машинах. Менеджер ресурсів жонглює кількома об'єктивними показниками, які спрямовані на зниження операційних витрат центру обробки даних, задовольняючи при цьому такі вимоги користувачів. Крім того, оскільки центри обробки даних динамічні, менеджер ресурсів повинен постійно вдосконалювати розміщення віртуальних екземплярів, щоб відповідати еволюції використання ресурсів, а також враховувати такі події, як технічне обслуговування, збої, виведення з експлуатації фізичних комп'ютерів. Саме тут в гру вступає міграція. Дійсно, ключовою особливістю віртуалізації серверів є можливість міграції віртуальних екземплярів з однієї фізичної машини

на іншу. Незалежність між віртуальними екземплярами та базовим обладнанням (або операційним середовищем) дозволяє, в принципі, розміщувати віртуальну машину або контейнер на кожній фізичній машині. Таким чином, вже запущений екземпляр може бути знищений і знову інстальований для запуску на інших машинах. Як відомо, віртуальні машини здатні зберігати свій стан виконання під час міграції, прозоро відновлюючи роботу на іншій фізичній машині. Складні алгоритми можуть передавати основну частину даних про стан в бекграунд, в той час як екземпляри продовжують працювати, зменшуючи переривання обслуговування (міграція в реальному часі). Міграції покращують використання ресурсів дата-центрів шляхом коригування розміщення віртуальних машин (динамічна повторна консолідація), щоб перемістити екземпляри в безпечне місце перед збоями хоста та подіями технічного обслуговування.

В роботі був визначений сценарій, в якому традиційні міграції, що лежать в основі як динамічної ре-консолідації, так і евакуації вузлів (для обслуговування вузлів), є або неефективними, або шкідливими як для зменшення втрат ресурсів, так і для покращення їх використання, а саме простій віртуальних машин у центрах обробки даних з блокуванням їхньої фізичної пам'яті, незважаючи на їхню бездіяльність або дуже низьку активність.

1.1.1 Непрацюючі віртуальні машини в дата-центрах

Нещодавні дослідження показують, що простої віртуальних машин є поширеною проблемою в центрах обробки даних. 80% віртуальних машин, розгорнутих у ЦОД, простоюють, не проявляючи жодних ознак активності користувачів. Непрацюючі екземпляри часто зустрічаються, коли приватні компанії розгортають власні DNS або поштові сервери, або коли розробники програмного забезпечення використовують ВМ для розробки та тестування нових додатків, рідко вимикаючи їх у неробочий час. Навіть якщо вони не

використовуються активно, ці ВМ блокують призначені їм ресурси, незалежно від того, як міграція їх консолідує.

Операторам дата-центрів не залишається іншого вибору, окрім як надмірно резервувати ресурси для подальшого розміщення нових екземплярів. На відміну від CPU, оператори дата-центрів уникають агресивного надмірного резервування пам'яті, оскільки наслідки нестачі пам'яті непередбачувано впливають на продуктивність усіх активних розміщених віртуальних машин (swap-in/out, thrashing). Цей сценарій представляє складну проблему: неактивні ВМ не можуть бути просто вимкнені, оскільки на них зазвичай розміщуються мережеві сервіси, які є важливими для кінцевих користувачів. Крім того, існуючі рішення покладаються або на реінжиніринг платформи (наприклад, використання контейнерів), або на спеціальні проксі-сервери на рівні додатків, які перехоплюють запити кінцевих користувачів в той час, як екземпляри закриваються. Отже, ці рішення не пропонують загальної або легко реалізованої методології, що залишає простір для вдосконалення та нового підходу.

1.1.2 Масштабне оновлення гіпервізорів

Гіпервізори є критично важливими компонентами в центрах обробки даних. Будь-яка несправність гіпервізора може вплинути на продуктивність, швидкість та доступність віртуальних машин, що виконують критично важливі для бізнесу робочі навантаження. Тому оновлення повинні швидко застосовуватися до всього парку хостів. Однак складні оновлення на рівні ядра неминуче вимагають перезавантаження гіпервізора, що призводить до зупинки запущених віртуальних машин.

Міграція в реальному часі є ефективним методом евакуації хостів, які потребують перезавантаження. Віртуальні машини зберігаються шляхом прозорого переміщення їх виконання на здорові хости, що призводить до

мінімального часу простою та обмеженого погіршення продуктивності. Однак міграцію в реальному часі важко масштабувати. Вона споживає ресурси центру обробки даних, такі як пропускна здатність мережі, щоб підтримувати передачу стану віртуальних машин в реальному часі. Крім того, вона вимагає достатньої кількості вільної пам'яті на хості для розміщення всіх переміщених екземплярів. Таким чином, існують сценарії, коли міграція в реальному часі є небажаною, наприклад, в центрах обробки даних з обмеженими ресурсами, або коли мінімальний час простою є зайвим (наприклад, для відмовостійких екземплярів, таких як пакетні завдання або репліковані сервіси). Ці фактори пропонують цікавий підхід: оновлення на місці. У цьому підході гіпервізор повністю оновлюється без необхідності міграції віртуальних машин в інше місце. Основна проблема полягає в тому, щоб виконати оновлення на місці, не спричиняючи непідйомних простоїв ВМ, пропонуючи дієву альтернативу класичному підходу до міграції в реальному часі.

У цій кваліфікційній роботі запропоновано спеціальну методику міграції, яка долає внутрішні обмеження традиційних міграцій, надаючи додатковий інструмент для оптимізації та спрощення управління ресурсами центрів обробки даних.

1.1.3 Міграція віртуальних машин у контейнери для консолідації в ЦОД

ВМ, що простоюють, призводять до марнування ресурсів у центрах обробки даних, зокрема пам'яті. Непрацюючі ВМ не можна вимкнути, щоб звільнити пам'ять, оскільки це призведе до порушення роботи сервісів. Ключовим спостереженням є те, що ВМ містять те, що ми називаємо шлюзовими процесами. Ці шлюзові процеси - це набір процесів у користувацькому просторі, які очікують на вхідні з'єднання (наприклад, веб-сервер або SSH-сервер), що є точкою входу до ВМ. В даній роботі пропонується SEaMLESS - рішення, яке мігрує весь набір процесів шлюзу з простоюючої ВМ до безресурсного контейнера, який надає

інтерфейс сервісу назовні. Таким чином, непрацюючі віртуальні машини можна відключити, щоб звільнити їхні ресурси. SEaMLESS реалізує виявлення активності користувачів механізм, який гарантує коректне та безпечне виконання пересаджених шлюзових процесів, працюючи в безресурсному середовищі, яке може підтримувати лише обмежену кількість обчислень.

При появі нової активності віртуальні машини відновлюють роботу, а їхні шлюзові процеси повертаються на своє місце, щоб прозоро відповідати на запити користувачів. Крім того, було розроблено гібридний метод призупинення роботи віртуальних машин, який називається suspend-to-swap і базується на традиційному системному просторі підкачки. Цей метод звільняє більшу частину пам'яті, виділеної і використовуваної віртуальною машиною, забезпечуючи при цьому швидке відновлення роботи завдяки лінивому завантаженню сторінок пам'яті зі свопу. SEaMLESS призначений для роботи на рівні орендарів. Він може збільшити використання ресурсів без співпраці з центром обробки даних або хмарним оператором.

1.2 Віртуалізація на основі гіпервізора

Віртуалізація серверів - це практика розбиття і розподілу фізичних машин, як правило, висококласних, але все ж таки звичайних машин для центрів обробки даних, на безпечні та ізольовані корпуси, які забезпечують звичне середовище для IT-персоналу для розробки та розгортання сервісів. Основною перевагою віртуалізації є зниження капітальних та операційних витрат на придбання та управління IT-обладнанням, що досягається за рахунок консолідації різних робочих навантажень на одному обладнанні. Існує два підходи до реалізації віртуалізації серверів, які працюють на найнижчому рівні абстракції, пропонуючи найбільшу гнучкість: віртуалізація на основі гіпервізора та віртуалізація на рівні операційної системи

Віртуальні машини - це програмні об'єкти, які реалізують віртуальну версію всього апаратного забезпечення комп'ютера, включаючи процесори, пам'ять, пристрої вводу/виводу та шини, які об'єднують всі компоненти. Таким чином, застаріла ОС може працювати всередині ВМ (так звана гостьова ОС), а незмінені додатки розміщуються зверху, пропонуючи DevOps середовище, ідентичне реальному аналогу. Існує кілька підходів до реалізації віртуальних машин, принаймні для використання в центрах обробки даних, є віртуалізація на основі гіпервізора.

Гіпервізор - це система, що відповідає за запуск віртуальних машин, гарантуючи, що програмне забезпечення, яке виконується у віртуальних машинах, поводиться так само, як якщо б воно виконувалося на фізичній версії тієї самої машини. Це завдання виконується на непрямому рівні, який використовує реальне апаратне забезпечення для реалізації ефектів, які гостьове програмне забезпечення має на віртуальне апаратне забезпечення. Оскільки це пов'язано з накладними витратами на виконання, ключовим завданням гіпервізора є досягнення найкращої ефективності.

Пряме виконання - це метод, який використовується для мінімізації таких накладних витрат, дозволяючи програмному забезпеченню у віртуальних машинах виконуватися на реальному обладнанні кожного разу, коли це безпечно. Звідси випливає, що гіпервізори зазвичай обмежують тип віртуального обладнання, яке підтримується віртуальними машинами. Наприклад, віртуальні процесори повинні мати однакову архітектуру набору команд (Instruction Set Architecture, ISA) фізична машина, що лежить в основі гіпервізора.

1.2.1 Гіпервізори типу 1 і 2

Гіпервізори традиційно поділяються на тип 1 і тип 2. Перш ніж визначити кожну категорію, треба представити роль диспетчера віртуальних машин (VMM).

Хоча VMM і гіпервізор іноді використовуються як взаємозамінні, в цій дипомній роботі ми називаємо VMM підсистемою, яка віртуалізує процесор і пам'ять ВМ. Іншими словами, вона контролює стан віртуальних процесорів та віртуальної пам'яті, оновлюючи його по мірі виконання ВМ. У гіпервізорах першого типу ВММ працює так званим "голим" способом на фізичній машині і безпосередньо отримує фізичні ресурси, процесор, пам'ять та пристрої вводу/виводу, які забезпечують віртуалізацію. Відомими зразками цієї категорії є VMware ESXi, Xen та Microsoft Hyper-V. У гіпервізорах 2-го типу ВММ є клієнтом ОС (так званої хостової ОС). Хостова ОС працює на "голому металі", тоді як ВММ запитує ресурси для запуску віртуальних машин. Наприклад, ВМ може бути процесом користувацького простору, який запускається хостом ОС. Такий процес час від часу планується для виконання інструкцій ВМ на фізичному процесорі. ВМ може виділяти пам'ять за допомогою системного виклику mmap (наприклад, у Linux) та виконувати запис на диск за допомогою системного виклику write у файлі. Яскравими прикладами цієї категорії є VMware Workstation та багато гіпервізорів на основі Linux та його інфраструктури віртуальних машин на основі ядра (KVM), таких як QEMU-KVM, AHV від Nutanix та AWS Nitro. На рисунку 1.2 зображено можливу компоновку шарів, які складають гіпервізор типу 1 та типу 2.

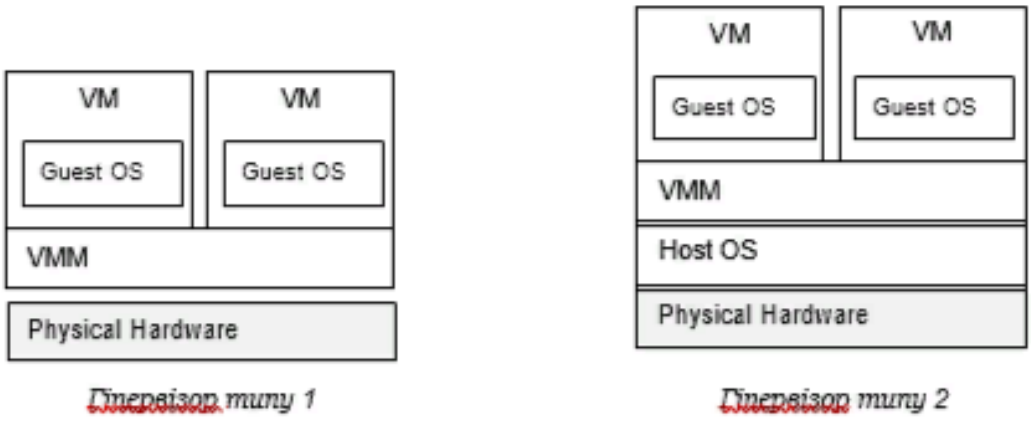


Рисунок 1.2 - Архітектури гіпервізорів типу 1 та типу 2

Ця категоризація важлива в контексті оновлення гіпервізорів, що є темою, яка цікавить цю дипломну роботу. Оновлення ОС або гіпервізора вимагає

перезапуску всього компонента, що означає перезавантаження хоста. Гіпервізори типу 1 можуть бути повністю монолітними, тому оновлення на ВММ призводить до перезавантаження. Гіпервізори типу 2, завдяки своїй модульності, можуть мати незалежні компоненти, такі як ВММ, які можна замінити без перезавантаження всієї машини. Однак, хостова ОС у гіпервізорах типу 2 відіграє фундаментальну роль у стабільності та ефективності гіпервізора, тому очікується часте оновлення на цьому рівні.

1.2.2 Віртуалізація процесора

Існує три стилі віртуалізації ресурсів, таких як процесор, пам'ять та пристрої вводу/виводу: повна віртуалізація, пара-віртуалізація та апаратна віртуалізація.

Для x86(-64) апаратна віртуалізація була впроваджена нещодавно. Однак, починаючи з 70-х років, цей підхід найбільш придатний для побудови ефективних, безпечних та надійних гіпервізорів. Попек і Голдберг вперше виклали цю концепцію у вигляді теореми. Вона звучить наступним чином. Якщо множина чутливих інструкцій (з побічними ефектами та/або залежностями від стану обладнання) є підмножиною привілейованих інструкцій (виконуваних лише в режимі супервізора/ядра), то центральний процесор можна віртуалізувати за допомогою прямого виконання та пасток і емуляції. Таким чином, Гіпервізор дозволяє виконання віртуальної машини безпосередньо на фізичному процесорі в найменш привілейованому режимі, будучи впевненим, що будь-яка чутлива інструкція, яка незаконно змінює стан обладнання фізичної машини, буде перехоплена, оскільки не виконується в режимі супервізора. Гіпервізор знову отримує контроль і емулює вплив чутливої інструкції на стан віртуального процесора, обманюючи віртуальну машину в тому, що виконання відбулося коректно. Однак x86 не відповідає вимогам Попека та Голдберга. Так було до 2005/2006 року, коли Intel та AMD представили свої розширення віртуалізації VT-x та AMD-V для процесорів x86.

На прикладі VT-x додано два нових режими виконання: кореневий та некореневий. Чутливі інструкції x86, які виконуються в режимі non-root, тепер спричиняють пастку, яка повертає керування гіпервізору для подальшої емуляції. Новий набір інструкцій керує входом у non-root режим (vmlaunch/vmresume). Крім того, структура в пам'яті, яка називається VMCS, зберігає дамп стану регістрів процесора при кожному переході з некореневого режиму, дозволяючи віртуальній машині маніпулювати своїм приватним контекстом процесора. У Linux KVM є компонентом, відповідальним за експорт можливості запуску програм у режимі, відмінному від кореневого. На рисунку 1.3 зображено взаємодію між режимами та захисними кільцями x86.

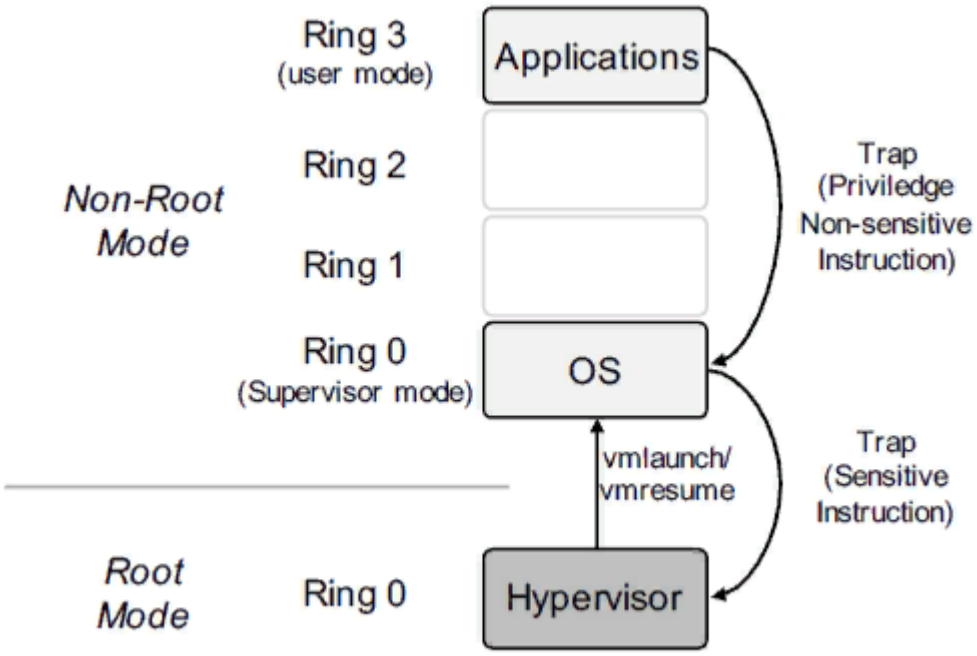


Рисунок 1.3 - VT-x та AMD-V в кореневому та некореневому режимах по відношенню до застарілих захисних кілець x86

Повна віртуалізація та паравіртуалізація - це попередні підходи до повторної оренди VT-x/AMD-V. Якщо коротко, повна віртуалізація використовує техніку, яка називається динамічною двійковою трансляцією. Блоки інструкцій, які віртуальна машина збирається виконати, аналізуються для виявлення x86-чутливих інструкцій, які перекомпілюються "на льоту" з емульованими нешкідливими

інструкціями. При паравиртуалізації замість перекомпіляції на льоту код гостьової ОС повністю виправляється перед встановленням у ВМ, щоб замінити чутливі інструкції на гіпервиклики. Гіпервиклики - це нешкідливі інструкції, які спричиняють пастку для ВМ, передачі керування гіпервізору для виконання емуляції. Динамічна двійкова трансляція має великі накладні витрати, але дозволяє запускати немодифіковані гостьові ОС. З іншого боку, паравіртуалізація має кращі показники продуктивності, але вимагає виправлення застарілого коду. Обидва підходи все ще відіграють важливу роль у віртуалізації пристроїв вводу/виводу.

1.2.3 Віртуалізація пам'яті

Пам'ять - це ще один ресурс, який виграє від розширення віртуалізації x86, що надається VT-x та AMD-V. Віртуальна машина має MMU і віртуальну пам'ять через підкачку, точно так само, як і реальні x86-машини. Однак, віртуальні машини також мають простір фізичної пам'яті, який відображається у діапазон віртуальної пам'яті в пам'яті VMM. Для прикладу розглянемо гіпервізор QEMU-KVM, де ВММ - це процес у просторі користувача, QEMU, який використовує API KVM для керування функціями апаратної віртуалізації. QEMU виділяє пам'ять для ВМ як неперервний масив байт, зіставлений з її віртуальним робочим простором. Отже, існує два виміри віртуально-фізичного відображення. По-перше, віртуальні адреси в гостьовій пам'яті - віртуальні адреси гостя (GVA) - відображаються у фізичний адресний простір гостя, який має однозначну відповідність масиву байт, виділеному ВМ. У свою чергу, масив байт у віртуальній пам'яті ВМ - віртуальні адреси хоста (HVA) - відображається у фізичну пам'ять реального комп'ютера - фізичні адреси хоста (HPA).

У традиційній x86 віртуальна пам'ять відображається, сторінка за сторінкою, на еквівалентні фрагменти фізичної пам'яті, відомі як фрейми. Структура, яка містить це відображення, називається таблицею сторінок. Таблиця сторінок - це

ієрархічна структура з 4 рівнями (у x86-64) підпорядкованих таблиць сторінок, з'єднаних у дерево. Трансляція адреси прозоро виконується MMU, автоматично спускаючись по дереву (page-table walk), щоб знайти запис, який містить відображення. Оскільки існує декілька віртуальних адресних просторів (тобто кожен процес має свій власний), вибір відповідальної таблиці сторінок відбувається через регістр процесора cr3, завантажений фізичною адресою кореня таблиці сторінок. Система кешування, яка називається Translation Look-aside Buffer (TLB), встановлюється і керується MMU, щоб уникнути надмірних обходів таблиць сторінок для повторюваних віртуальних адрес.

Апаратна віртуалізація, що постачається з VT-x та AMD-V, представляє відповідно розширену таблицю сторінок (EPT) та вкладені таблиці сторінок (NPT). MMU віртуалізовано у віртуальній машині, тому гостьова ОС може прозоро керувати віртуально-фізичними відображеннями за допомогою таблиць сторінок, встановлених у віртуальному регістрі cr3. Гіпервізор у кореневому режимі також встановлює власне віртуально-фізичне відображення, транслюючи байтовий масив, який підтримує пам'ять ВМ, до HPA. EPT поєднує в апаратному забезпеченні два виміри відображення. GVA перетворюються через таблиці гостьових сторінок у масив байт у HVA, який, нарешті, перетворюється на HPA за допомогою гіпервізора page-table. На рисунку 1.4 зображено взаємозв'язок між двовимірними адресними просторами в апаратній віртуалізації.

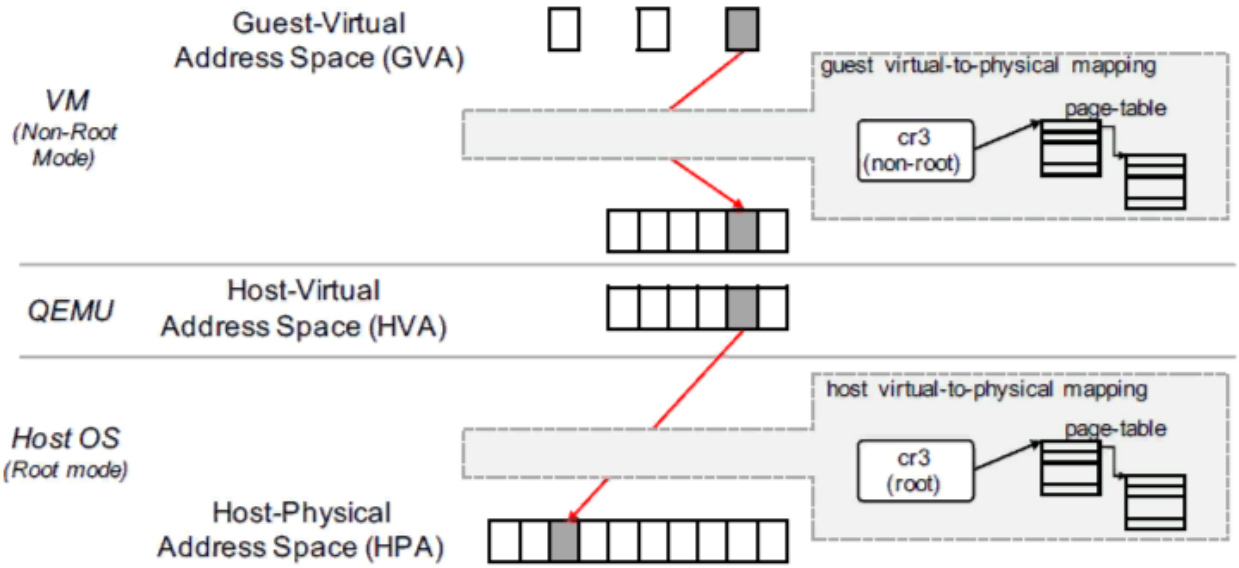


Рисунок 1.4 - Двовимірний пейджинг у VT-x/AMD-v з EPT/NPT

Прогулянка по EPT у найгіршому випадку призводить до збільшення кількості звернень до пам'яті у квадраті порівняно з одиночним виміром. TLB відіграє більш важливу роль, і багато покращень спрямовано на максимізацію кешування, наприклад, зменшення кількості записів у таблиці сторінок за допомогою hug-сторінок.

1.2.4 Віртуалізація вводу/виводу

Введення/виведення в сучасних гіпервізорах все ще представляє велику різноманітність підходів. Ми почнемо з повної віртуалізації, в цьому контексті також відомої як емуляція вводу/виводу. У такому випадку гостьова ОС запускає незмінені драйвери для реально існуючих пристроїв, хоча і віртуалізованих. Усі взаємодії між гостьовою ОС та пристроями вводу/виводу, а саме Memory Mapped I/O (MMIO) та Port Mapped I/O (PIO), налаштовані на пастку. Ефекти емулюються гіпервізором, оновлюючи стан віртуального пристрою, щоб ввести гостьову ОС в

оману і змусити її думати, що взаємодія дійсно відбулася. Необхідні операції вводу/виводу, які повинен виконувати віртуальний пристрій, перетворюються на робоче навантаження вводу/виводу, так зване back-end, яке гіпервізор виконує на апаратному забезпеченні фізичної машини. Цей підхід вважається найменш ефективним через емуляцію всіх апаратних деталей існуючого пристрою, необхідну для забезпечення повної сумісності зі старими драйверами гостьової ОС. Однак це хороший компроміс для віртуалізації застарілих пристроїв з низькою пропускною здатністю, таких як послідовний порт і периферійні пристрої введення (наприклад, клавіатури, миші).

Паравіртуалізація має той самий компроміс, що і віртуалізація процесора, приносячи в жертву сумісність заради продуктивності. Серія спеціальних віртуальних пристроїв, які називаються паравіртуалізованими пристроями, працюють у парі зі спеціальними драйверами, розгорнутими в гостьової ОС, обидві розроблені, щоб запропонувати віртуальний ввід/вивід, який мінімізує необхідну емуляцію. Паравіртуалізовані пристрої є простими, реалізованими у вигляді пар кільцевих буферів, тому їх дуже легко емулювати. Хоча це швидше, ніж повна віртуалізація, драйвери для паравіртуалізованих пристроїв потрібно портувати до гостьової ОС.

Проілюстровані підходи не використовують апаратну віртуалізацію. Для досягнення найкращої продуктивності гостьовій ОС у віртуальній машині краще мати можливість взаємодіяти з реальним пристроєм без втручання гіпервізора. Це називається прямим призначенням пристрою, або PCI passthrough, оскільки воно розроблено спеціально для класу пристроїв, які забезпечують найвищу продуктивність, а саме для PCI та PCI Express (PCIe). Ми вже представили список взаємодій між програмним та апаратним забезпеченням: PIO та MMIO. PIO використовує спеціальні інструкції x86 для доступу до регістрів пристроїв, тоді як MMIO відображає регістри на фізичні адреси, щоб їх можна було читати і записувати за допомогою операцій з пам'яттю. Таким чином, за допомогою MMIO-відображення регістрів реального пристрою у фізичний адресний простір віртуальної машини, гостьова ОС може безпосередньо взаємодіяти з пристроєм.

Ускладнення виникає, коли гостьова ОС налаштовує реальний пристрій на прямий доступ до пам'яті (DMA) до пам'яті віртуальної машини. Фізичні адреси, які бачить гостьова ОС, не відповідають реальним фізичним адресам, які бачать пристрої. Intel вирішує цю проблему в рамках своєї технології віртуалізації для прямого вводу/виводу (VT-d), представляючи компонент під назвою I/O Memory Management Unit (IOMMU). Треба звернути увагу, що аналог цієї технології від AMD називається AMD-Vi. IOMMU відповідає за перетворення фізичних адрес у транзакціях прямого доступу до пам'яті (DMA), що виконуються пристроєм. Фізичні адреси гостьових пристроїв транслюються в HPA, де фактично відображається оперативна пам'ять. Відображення задається за допомогою таблиць сторінок, подібно до аналога, який обслуговує MMU.

Пряме призначення пристроїв надає одній віртуальній машині повний контроль над усім пристроєм PCI/PCIe, який більше не може бути спільним для інших віртуальних машин. Щоб збільшити масштабованість прямого призначення пристроїв, виробники обладнання почали вбудовувати в свої карти можливості віртуалізації: Single Root I/O Vitalization (SR-IOV). Пристрої PCIe, які підтримують SR-IOV, виконують апаратне мультиплексування команд вводу/виводу від різних віртуальних машин. Ця функція відображається гіпервізору так, ніби один пристрій складається з безлічі незалежних кінцевих точок PCIe (віртуальних функцій, або VF, за термінологією SR-IOV). Кожна VF може бути безпосередньо призначена віртуальній машині, а одна карта SR-IOV може підтримувати тисячі VF.

1.3 Віртуалізація на рівні операційної системи

Віртуальні машини забезпечують надійну ізоляцію, водночас пропонуючи звичне середовище для розгортання додатків. Перше досягається застарілою гостьовою ОС всередині віртуальних машин, яка пропонує набір типових абстракцій, таких як системні виклики, файлова система, бібліотеки, об'єкти

міжпроцесного зв'язку (IPC) тощо. Тим не менш, значна частина накладних витрат, пов'язаних з віртуалізацією, пов'язана з виконанням застарілої гостьової ОС як непривілейованого об'єкта. Запуск великої кількості різноманітних ОС ніколи не був метою віртуалізації серверів в центрах обробки даних (кілька основних ОС легше підтримувати), натомість основна увага приділяється ізоляції та зручному для розгортання середовищу. Таким чином, спільнота віртуалізації працювала над альтернативним підходом: віртуалізація на рівні операційної системи.

Віртуалізація на рівні ОС ізолює групи процесів, включно з джерелами ОС, які вони бачать і до яких звертаються, утворюючи так звані контейнери. Сучасні ОС вже забезпечують ізоляцію процесора та пам'яті для процесів. Однак, процеси все ще поділяють багато просторів імен, тобто вони поділяють такі елементи, як ідентифікатори процесів (PID), їх положення у глобальному дереві процесів, змонтовані файлові системи, мережевий стек (IP-адреси, транспортні порти, таблиця маршрутизації тощо), об'єкти IPC, ім'я хоста та системних користувачів. Ізоляція вищезгаданих просторів імен (namespace isolation) є особливістю, яку використовують для створення контейнерів такі рушії, як Docker та LXC. Групи керування (cgroup) у Linux або об'єкти завдань у Windows обмежують та обліковують використання ресурсів (процесор, пам'ять, дисковий ввід/вивід, мережа тощо) для групи процесів, доповнюючи ізоляцію контейнера. Подальша безпека досягається за допомогою таких функцій, як seccomp у Linux, для фільтрації системних викликів, що викликаються з контейнера, зменшуючи можливості для атак на підвищення привілеїв.

Контейнери зазвичай вважаються швидшими за віртуальні машини, хоча продуктивність процесора та пам'яті близька до апаратної віртуалізації ВМ. Швидкість вводу/виводу в контейнерах вища, ніж у повністю віртуалізованих або частково віртуалізованих пристроях вводу/виводу віртуальних машин, але подібна до швидкості вводу/виводу з безпосередньо призначеними пристроями. Контейнер загалом займає менше пам'яті, ніж віртуальна машина, завдяки повноцінній гостьовій ОС, що працює всередині контейнера. З іншого боку, спільне використання одного ядра не ізолює спільно розміщені контейнери від збоїв ОС.

Крім того, більша поверхня атаки, з точки зору кількості інтерфейсів, які забезпечують ізоляцію, робить контейнери менш захищеними в багатокористувацьких дата-центрах.

1.4 Міграція віртуальних екземплярів

Віртуалізація серверів відокремлює віртуальні екземпляри від базового сервера, тобто апаратного забезпечення для віртуальних машин і середовища операційної системи для контейнерів. Таким чином, віртуалізація серверів дозволяє повторно створювати той самий віртуальний екземпляр на кожному іншому сервері в дата-центрі. Цей принцип лежить в основі міграції віртуальних екземплярів, що визначається як дії з переміщення виконання віртуального екземпляра з однієї машини на іншу. У цьому розділі надається огляд міграції віртуальних машин, класичної функції віртуалізації на основі гіпервізора. Також представлено новаторські підходи до міграції контейнерів.

1.4.1 Міграція віртуальних машин

Віртуальні машини працюють на віртуальному обладнанні. Програмне забезпечення всередині віртуальних машин, тобто гостьові ОС і додатки, взаємодіє лише зі станом віртуального обладнання. Такий стан повністю представлений у програмному забезпеченні гіпервізора. Ця властивість уможливлює створення контрольних точок та міграцію віртуальних машин.

Перш за все, будь-яку віртуальну машину можна призупинити, так зване призупинення в оперативній пам'яті, просто заморозивши потоки гіпервізора, які виконують інструкції віртуальної машини на фізичних процесорах. Стан

виконання, включаючи контекст процесора, залишається доступним у пам'яті гіпервізора. Таким чином, віртуальні машини можуть миттєво відновити роботу, як тільки потоки будуть розморожені.

Suspend-to-RAM є ще однією функцією віртуалізації: створення контрольних точок. Гіпервізор, призупинивши роботу віртуальної машини, може поставити в контрольну точку всю інформацію, яка описує стан виконання віртуальної машини в цей момент. Ця інформація включає вміст оперативної пам'яті ВМ, контекст віртуального процесора, регістри віртуальних пристроїв. Ця техніка відома як suspend-to-disk, коли місцем призначення для стану з контрольною точкою є сховище. Необхідно зауважити, що створення та скасування стану призупинення на диск (відновлення) потребує часу, особливо для створення контрольних точок/відновлення оперативної пам'яті віртуальної машини, яка містить гігабайти даних.

Завершення/перезапуск ВМ. Здавалося б, здається надуманим називати міграцією просте завершення/перезапуск віртуальної машини. Однак цей підхід ефективно застосовується в щільних центрах обробки даних, таких як публічні хмари, для переміщення екземплярів між різними машинами. Наприклад, Amazon EC2 та Google Compute Engine (GCE) використовують цей підхід для витіснення спотових екземплярів у разі нестачі ресурсів або перевищення ринкової ціни екземпляра. Крім того, Amazon EC2 використовує його, щоб звільнити фізичні машини, які потребують обслуговування (наприклад, оновлення гіпервізора).

Як випливає з назви, віртуальні машини вимикаються і припиняють роботу на фізичній машині. Згодом інша віртуальна машина створюється заново на основі того ж шаблону (тип машини, завантажувальний образ та інші параметри) і завантажується. Такий підхід передбачає певний рівень автоматизації, оскільки програмне забезпечення, розгорнуте на перезавантаженій віртуальній машині, автоматично повторно ініціалізується під час завантаження. Однак стан виконання неминуче втрачається. Сервіси зі збереженням стану можуть тривалий час простоювати, особливо якщо програмам потрібно завантажити великий обсяг даних (наприклад, бази даних у пам'яті).

Холодна міграція. Призупинення на диск скидає стан віртуальної машини на сховище. Однак, якщо місцем призначення стану з контрольною точкою є інший гіпервізор, віртуальна машина може мігрувати і зберегти свій стан виконання. Ця техніка називається холодною міграцією.

Холодна міграція здатна зберегти стан виконання переміщеної ВМ, однак, екземпляр залишається призупиненим на помітний проміжок часу. Віртуальна машина може відновити роботу лише тоді, коли весь її стан буде передано до місця призначення через мережу. Розмір стану віртуальних процесорів, віртуальних мережевих карт, віртуального дискового контролера становить лише порядок мегабайтів, тоді як вміст віртуальної оперативної пам'яті домінує з розмірами порядку гігабайтів. Якщо припустити, що пропускна здатність мережі становить 1, 10 або 40 Гбіт/с, то для віртуальних машин, оснащених принаймні 5 ГБ оперативної пам'яті, передача даних займає не менше однієї секунди, так само як і час їх простою.

Міграція в реальному часі. VMware (з vMotion), а також Кларк та ін., незалежно один від одного, започаткували підхід до міграції віртуальних машин в реальному часі, тобто виконання міграції без помітного простою.

Запропонована конструкція, яка називається міграцією з попереднім копіюванням в реальному часі, виштовхує вміст оперативної пам'яті заздалегідь, поки віртуальні машини продовжують працювати, що значно скорочує обсяг даних, які передаються, поки віртуальні машини призупиняються. Ця стратегія є успішною, оскільки, якщо припустити, що доступ до сховища здійснюється через мережу зберігання даних (SAN) або мережеве сховище (NAS), стан віртуальної машини визначається її оперативною пам'яттю. Виштовхування пам'яті - це ітеративний процес, який відбувається в декілька раундів, на так званій стадії попереднього копіювання. Під час кожного раунду сторінки пам'яті, змінені виконанням віртуальної машини, відомі як брудна пам'ять, повторно надсилаються до місця призначення. Лише коли розмір брудної пам'яті на даному раунді падає нижче певного порогу, етап попереднього копіювання завершується. Після цього стадія зупинки і копіювання зупиняє роботу віртуальної машини для передачі

залишків стану. Після завершення етапу зупинки та копіювання ВМ миттєво відновлює роботу на місці призначення. ВМ зупиняється лише тоді, коли решта даних може бути передана за час, менший, ніж цільовий час простою, отже, ця схема забезпечує фіксований час простою, як правило, порядку десятків мілісекунд. Кларкта та ін. визначають як робочий набір, що записується, набір сторінок, що часто записуються, які не можуть бути попередньо скопійовані достатньо швидко, перш ніж ВМ запише їх знову. Робочий набір, що записується, передається під час фази зупинки та копіювання, і, для певних важких для запису сторінок, він може бути перенесений, це не забезпечує короткого часу простою. Найгірший випадок - коли віртуальна машина забруднює пам'ять зі швидкістю, вищою за пропускну здатність мережі, яка попередньо копіює пам'ять. В такому випадку час простою буде таким же, як і при холодній міграції. Альтернативним підходом до міграції в реальному часі до копіювання є міграція в реальному часі після копіювання. Цей підхід починається з етапу зупинки та копіювання для перенесення стану ВМ, за винятком віртуальної оперативної пам'яті, яка зберігається на джерелі. ВМ негайно відновлюється на цільовій машині, отримуючи пам'ять на вимогу через мережу при першому зверненні. Зрештою, вся пам'ять буде перенесена, і міграція завершиться.

Міграція з попередньою копією в реальному часі набула значного успіху. Всі основні гіпервізори підтримують цю технологію, а також деякі основні хмарні інфраструктури, такі як GCE та Microsoft Azure. З іншого боку, пост-копіювання є успішним у реальному часі міграції важких для запису ВМ з великим робочим набором, який можна записати. Однак, при першому доступі до сторінки пам'яті виникає велика затримка при її віддаленому отриманні, що може сильно вплинути на продуктивність розміщених додатків. Обидва методи можна використовувати разом у гібридному підході. Хоча вони не розкрили свій поріг, GCE автоматично перемикається на міграцію після копіювання, коли ВМ не встигає виконати попереднє копіювання протягом цільового часу простою.

Обмеження контрольних точок та міграцій. Контрольні точки та міграції використовують властивість того, що стан віртуальної машини повністю

інкапсульований у програмному забезпеченні. Ця властивість стосується і центрального процесора, оскільки його контекст експортується Intel VT-x через структуру VMCS (AMD-V є аналогом). Насправді це особливо актуально для повністю емульованих або паравіртуалізованих пристроїв, реалізованих гіпервізором у програмному забезпеченні. Однак це не стосується безпосередньо підключених пристроїв. Наприклад, деякі регістри пристрою можуть бути недоступні для читання, а отже, їх неможливо перевірити. Аналогічно, інші регістри не можуть бути записані без побічних ефектів, таких як запуск передачі мережевих пакетів у контролері мережевого інтерфейсу (NIC). Якщо стан інтерфейсного пристрою не може мігрувати разом з віртуальною машиною, гостьовий драйвер, швидше за все, вийде з ладу через невідповідність станів у місці призначення.

Одним із підходів є міграція стану обладнання, яка, однак, вимагає відповідності драйверів пристроїв, з модифікаціями для конкретних пристроїв для її підтримки. Однак, жоден основний гіпервізор або платформа не підтримує його. Інший підхід полягає у міграції/перевірці стану віртуальної машини без використання безпосередньо підписаного пристрою. Ця методика використовує підтримку гіпервізором та гостьовою ОС гарячого підключення/відключення PCI. Пристрій з прямим призначенням видаляється перед міграцією/контрольною точкою, щоб потім знову вставити його після відновлення роботи ВМ. У разі міграції в реальному часі, для підтримки доступності мережі під час ітеративного етапу попереднього копіювання, трафік перенаправляється через паравіртуалізовану мережеву карту. Безпосередньо підключений пристрій і паравіртуалізована мережева карта повинні бути об'єднані в єдиний зв'язаний інтерфейс, який логічно об'єднує кілька мережеві інтерфейси для забезпечення прозорої стійкості до відмов каналів зв'язку (режим активного резервування) та більшої пропускної здатності (балансування навантаження).

Ця стратегія, тим не менш, є непрозорою для гостьової ОС. Не тільки гостьова ОС повинна підтримувати гаряче підключення PCI/PCIe, але й користувач ВМ повинен розгорнути автоматизований сценарій конфігурації для повторної

ініціалізації мережевого стеку після перепідключення карти. Цей підхід широко підтримується основними ОС (Linux та Windows) та гіпервізорами (QEMU-KVM, Hyper-V, VMware), і навіть офіційно прийнятий у Microsoft Azure.

1.4.2 Міграція контейнерів

Контейнери важче мігрувати у порівнянні з віртуальними машинами. Стан контейнера не є повністю інкапсульованим, як у випадку з віртуальними машинами, а розкиданий по декількох таблицях і структурах ОС, що містять такі дескриптори, як ідентифікатори процесів, дескриптори відкритих файлів, сигнали тощо. Це ускладнює контрольну точку/відновлення контейнера, а отже, і його міграцію.

Міграція контейнерів є підкласом міграції процесів. Міграція процесів була гарячою темою в контексті єдиних образів системи на рівні простору користувача (Single System Images, SSI). Сьогодні міграція процесів залишається актуальною в контексті контейнерів, які, зрештою, є сукупністю процесів.

CRIU (Check- point/Restore In Userspace) - це сучасне рішення з відкритим вихідним кодом для перевірки/відновлення процесів у Linux. Представлений у 2011 році, CRIU був інтегрований, хоча і експериментально, в основні контейнерні системи, такі як Docker. CRIU використовує набір хуків ядра, які тепер доступні в основній версії Linux, для експорту всіх компонентів стану процесу. Аналогічно, CRIU використовує інші хуки Linux для відновлення процесу, наприклад, запис до "/proc/sys/kernel/ns_last_pid", щоб примусово призначити певний PID. Відновлення починається з порожнього процесу, який перетворює себе (відкриває файли, створює дочірні процеси тощо) у процес, на який спрямовано перевірку. Коли CRIU використовується для створення контрольних точок для всього дерева процесів у контейнері, весь контейнер стає контрольною точкою. Міграція

контейнера просто переносить "холості" або живі" процеси з контрольними точками на іншу машину.

Хоча рішення для міграції контейнерів існують, жодна з основних хмарних платформ не підтримує їх, хоча деякі приватні хмарні провайдери пропонують їх у виробництві.

1.5 Управління ресурсами в центрах обробки даних

Центри обробки даних відкривають широкі можливості для скорочення операційних витрат на IT-інфраструктуру. Одним із прикладів є широке впровадження енергоефективного обладнання, наприклад, компонентів, що підтримують динамічне масштабування напруги та частоти, або проектування об'єктів, що використовують поновлювані джерела енергії, такі як вільне охолодження.

Іншим важливим способом є максимальне використання ресурсів, таких як обчислювальні ресурси, ресурси зберігання даних та мережеві ресурси. Ці ресурси надаються широким спектром апаратного забезпечення, від процесорів, дисків і мережевих карт до стійок і джерел безперебійного живлення, і це лише деякі з них. Це обладнання потрібно купувати, встановлювати, вмикати, охолоджувати та обслуговувати, що призводить до високих капітальних та операційних витрат. Тому максимальне використання ресурсів знижує вартість центру обробки даних. Консолідація серверів - це фундаментальний метод покращення використання ресурсів у центрах обробки даних.

Програмне забезпечення, яке називається менеджер ресурсів, розраховує розміщення віртуальних екземплярів на фізичних серверах. Однією з цілей розміщення є розміщення віртуальних екземплярів на мінімальній кількості хостів, щоб більше обладнання могло працювати в режимі енергозбереження. Одночасно менеджер ресурсів керує угодами про рівень обслуговування (SLA), укладеними з

користувачами дата-центру. Менеджер ресурсів повинен гарантувати узгоджену продуктивність, доступність і спорідненість (наприклад, розміщення реплік сервісів у різних підмережах) екземплярів, зазнаючи грошових штрафів у разі недотримання цих умов.

1.5.1 Статична консолідація

Статична консолідація полягає в тому, що обчислюється оптимальне розміщення для реальних екземплярів і ніколи не змінюється. По мірі надходження нових завдань вони розміщуються на відповідних машинах, забезпечуючи наявність необхідних ресурсів. Насправді, статична консолідація повинна уникати будь-яких суперечок за ресурси, які можуть погіршити продуктивність запущених екземплярів.

Віртуальні екземпляри часто мають надмірні вимоги до використання ресурсів, що суттєво обмежує потенціал консолідації в центрі обробки даних. Це явище частково пов'язане з попередньо визначеними розмірами екземплярів, подібно до того, що пропонується в публічних хмарах, але також і з тим, що вимоги до ресурсів розраховуються на пікові навантаження, які можуть траплятися рідко. Отже, необхідно надмірно резервувати ресурси, щоб підвищити рівень використання ресурсів центру обробки даних.

Надмірне резервування - це практика вважати доступними більше віртуальних ресурсів, ніж ті, що підкріплені фізичними ресурсами. Співвідношення між віртуальними та фізичними ресурсами називається коефіцієнтом овербукінгу. Надмірне бронювання може призвести до погіршення продуктивності через боротьбу за отримання зайнятого ресурсу. Така ситуація називається хот-спот. Розміщення віртуальних екземплярів повинно змінюватися відповідно до еволюції використання ресурсу, отже, динамічна консолідація.

1.5.2 Динамічна консолідація

Динамічна консолідація використовує міграцію (завершення/перезапуск, "холодну" або "живу") для корекції розміщення віртуальних екземплярів.

Для моніторингу та запобігання виникненню "гарячих точок" у перевантаженому середовищі аналізується динамічна консолідація екземплярів. Динамічна консолідація також може досягти таких цілей, як зменшення фрагментації деяких ресурсів (наприклад, надання великих фрагментів пам'яті для великих екземплярів) та евакуація віртуальних екземплярів для виконання технічного обслуговування хостів.

Моніторинг та прогнозування робочого навантаження - тема, тісно пов'язана з динамічною консолідацією. Моніторинг використання ресурсів може бути використаний для динамічного регулювання коефіцієнта надлишкового бронювання, який зазвичай встановлюється статично для всіх машин. Багато робіт зосереджено на прогнозуванні використання ресурсів (патерни використання, термін служби). Це дозволяє обґрунтовано розміщувати сумісні екземпляри, хоча і схильне до упереджених прогнозів, особливо для перехідних або тестових робочих навантажень.

1.5.3 Надмірне резервування ресурсів

У розділі 1.5.1 було коротко представлено концепцію надмірного бронювання, також відому як надмірні зобов'язання або надмірні підписки. Тепер було охарактеризовано, як надмірне резервування використовується в центрах обробки даних.

Всі ресурси в центрі обробки даних можуть бути перевантажені. Однак більшість літературних джерел зосереджується на центральному процесорі та пам'яті як представниках двох класів ресурсів, які демонструють різні моделі використання. Процесор характеризується стрибкоподібною поведінкою, з великим розривом між середнім та піковим навантаженням, що відкриває широкі можливості для надмірного бронювання. Крім того, коротка тривалість пікових навантажень призводить до низьких ризиків конфліктів. І навпаки, пам'ять є проблематичною. Загалом, середнє використання пам'яті часто наближається до пікового, що призводить до зменшення можливостей для надмірного резервування. Тому пам'ять рідко буває переповнена, що є обмежувальним фактором у консолідації серверів.

Однак існує різниця між використаною пам'яттю (записаною хоча б один раз) та робочою множиною, а саме підмножиною пам'яті, яку потрібно читати та записувати для просування виконання екземпляру. Слід зауважити, що поняття робочої множини пов'язане, але відрізняється від розміру записуваної множини живої міграції, представленої у розділі 1.2.1. Дійсно, віртуальні екземпляри, які неактивні протягом тривалих періодів часу, тобто простоюють, мають малу робочу множину, що створює можливості для надлишкового резервування. На основі цього спостереження було розроблено SEaMLESS, представлений у розділі 2.

Зменшення продуктивності, пов'язане з процесором або гарячою точкою пам'яті, сильно відрізняється. У випадку з процесором, гіпервізор/ОС мультиплексує процесорний час між екземплярами. Контекстний перемикач процесора є легким, що призводить до низьких накладних витрат на мультиплексування такого ресурсу. З іншого боку, як ОС, так і гіпервізори використовують пам'ять (простір підкачки) для мультиплексування оперативної пам'яті після вичерпання доступного простору. Таким чином, запущені робочі навантаження страждають від більших затримок читання/запису пам'яті, аж до повної втрати відгуку (thrashing).

Надлишкове резервування для віртуальних машин складніше, ніж для контейнерів. Віртуальні машини створюються з обмеженою кількістю ресурсів, які

не можуть бути легко розширені. Крім того, деяка цінна інформація (наприклад, вільні сторінки) не є відкрито доступною для гіпервізора, що не дозволяє ефективно витісняти/преференціювати ресурси. Тепер слід детально розглянути традиційний механізм надмірного резервування, який використовують гіпервізори.

Підкачка пам'яті Гіпервізора. Підкачка - це метод мультиплексування оперативної пам'яті після вичерпання вільної пам'яті. Лише робочий набір запущених програм/інстанцій повинен бути присутнім у пам'яті, щоб дозволити виконання. Якщо немає вільних сторінок, деякі з них повинні бути виселені до сховища, щоб повернути вільну оперативну пам'ять. Ця операція називається витісненням (swapping-out).

Сторінки, до яких навряд чи буде найближчим часом звертання, є найкращими кандидатами на витіснення (наприклад, сторінки, що приблизно відповідають критерію Least Recently Used-LRU-сторінок). Необхідно зауважити, що при зверненні до сторінки, яка витісняється, вона має бути попередньо завантажена в оперативну пам'ять. Ця операція називається підкачуванням (swapping-in).

Підкачка для звільнення оперативної пам'яті та підкачка для відновлення сторінки в оперативній пам'яті збільшують затримку доступу до сторінки на кілька порядків. Коли сукупний розмір робочого набору для всіх додатків/прикладних програм перевищує обсяг оперативної пам'яті, робочі навантаження не можуть прискорити своє виконання через накладні витрати, пов'язані з безперервною підкачкою робочого набору, що потенційно може призвести до зупинки ядра. Це те, що називається трешуванням.

У контексті віртуалізації на основі гіпервізора існує два рівні підкачки. По-перше, гостьова ОС у віртуальній машині може увімкнути свопінг, таким чином витісняючи на диск частину своєї віртуальної оперативної пам'яті. Цей метод не допомагає при надмірному резервуванні пам'яті. Звільнена пам'ять залишається у віртуальній машині, а отже, недоступна ззовні для інших екземплярів. Другий рівень підкачки виконується гіпервізором, так званий гіпервізорний свопінг, здатний ефективно перерезервувати пам'ять. Гіпервізорний свопінг має деякі

недоліки. Гіпервізор, будучи агностичним по відношенню до невикористаної пам'яті в гостьовій системі, може ненавмисно вивільнити вільну пам'ять, що призведе до непотрібного та неефективного звільнення оперативної пам'яті. Іншим подібним недоліком є подвійне підкачування. Гостьова ОС може підкачувати (або записувати в пам'ять) сторінки, які вже були підкачані гіпервізором. Це призводить до завантаження сторінок з диска лише для того, щоб одразу після цього записати їх назад на диск. Ці недоліки долаються завдяки співпраці з гостьовою ОС. Зокрема, балонування дозволяє звільнити вільну пам'ять у гостьовій системі без необхідності використовувати своп, як описано нижче.

Гаряче підключення. Процесори, пам'ять та деякі пристрої вводу/виводу можна "гаряче" підключати та "гаряче" відключати від віртуальних машин, що є підходом як до надання, так і до повернення ресурсів. Гаряче підключення та відключення є непрозорими методами. Гостьова ОС у віртуальній машині повинна відповідати таким вимогам під'єднання/від'єднання, щоб уникнути потенційного збою системи через раптове зникнення апаратного компонента.

З одного боку, гаряче відключення процесора добре підтримується основними гіпервізорами, такими як VMware ESX та KVM. З іншого боку, гаряче відключення пам'яті може призвести до тривалих затримок у відновленні оперативної пам'яті, а також може призвести до збоїв. Підтримка гарячого відключення пам'яті обмежена і рідко використовується у виробничих середовищах.

Як було сказано вище, гіпервізор має мало інформації про стан пам'яті у віртуальній машині. Наприклад, для виділення пам'яті віртуальної машини гіпервізори використовують підкачування на вимогу, тобто лише при першому зверненні до сторінки відбувається резервування фізичного фрейму під цю пам'ять. Якщо виділена сторінка пізніше звільняється гостьовою ОС, відповідний фрейм залишається зайнятим у гіпервізорі. Як було коротко зазначено під час опису свопінгу гіпервізора, відсутність самоаналізу призводить до неефективного звільнення пам'яті, підкачування сторінок, які не мають жодного сенсу.

Балонування - це техніка паравиртуалізації, яка використовує псевдо-драйвер, встановлений у гостьовій ОС, який пересилає інформацію про те, які сторінки у віртуальній машині вільні. Зазвичай, драйвер балонування працює шляхом виділення невикористаної пам'яті у віртуальній машині (надування повітряної кульки). Виділена пам'ять зберігається закріпленою, щоб гостьова ОС не змогла її замінити.

Балонування має ще одну перевагу. Вона збільшує навантаження на пам'ять гостьової ОС, змушуючи її звільняти несуттєву пам'ять, таку як кеш файлової системи. Гіпервізор може повернути пам'ять віртуальній машині, тобто драйвер звільняє пам'ять, зайняту до цього часу.

Балонування є непрозорим. Гостьова ОС повинна відповідати вимогам балонування, встановивши та увімкнувши відповідний драйвер. Як наслідок, ізоляція віртуальної машини послаблюється. У випадку надмірного роздування, коли гіпервізор запитує надмірну кількість пам'яті, гостьова ОС може вийти з пам'яті (Out-Of-Memory (OOM)), завершуючи роботу основного додатку, щоб зменшити тиск.

Балонування вважається опортуністичним підходом до відновлення пам'яті. Тільки вільні сторінки в гостьовій ОС можуть бути ефективно відновлені без негативних наслідків. Тому балонування повинно залишатися в парі з гіпероператорним свопінгом для певного відновлення оперативної пам'яті у випадку хот-споту.

Дедуплікація сторінок. Дедуплікація сторінок є ще однією опортуністичною технікою для зменшення споживання пам'яті віртуальними машинами. Дедуплікація сторінок базується на спільному використанні пам'яті, яка зберігає ідентичний вміст, повертаючи надлишкові копії тих самих сторінок. Хоча дедуплікація сторінок не є суворо пов'язаною з віртуалізацією, вона відіграє важливу роль у підвищенні продуктивності серверів.

Віртуальні машини, особливо в хмарних середовищах, часто створюються з одних і тих самих шаблонів, що робить їх здатними зберігати ідентичні сторінки (наприклад, з однаковим образом ядра або однаковими бібліотеками).

Дедуплікація сторінок виконується гіпервізором, який періодично сканує пам'ять, виділену для віртуальних машин. При виявленні двох або більше ідентичних сторінок, зберігається лише один фрейм, а інші копії звільняються. Потім всі віртуальні сторінки зіставляються з одним фреймом хоста і обробляються як Copy-On-Запис (COW). Будь-який новий запис на будь-якій з цих сторінок призводить до повторного створення копії.

VMware була одним із перших прихильників дедуплікації сторінок, повідомляючи про економію пам'яті на рівні 40% на декількох віртуальних машинах. Однак, результати на практиці відрізняються. Впровадження рандомізації адресного простору (ASLR) та hug-сторінок ускладнює пошук ідентичних сторінок. Крім того, опортуністичний характер дедуплікації сторінок не може детерміновано забезпечити відновлення пам'яті. Тому це рішення є вторинним по відношенню до свопінгу гіпервізора.

1.6 Обслуговування в центрах обробки даних

Центри обробки даних проходять регулярне технічне обслуговування, щоб гарантувати безперервність обслуговування, безпеку та оптимальну продуктивність. Обслуговування передбачає регулярну модернізацію інфраструктури електроживлення та охолодження, а також розгортання нової мережевої структури та фізичних машин, модернізацію або заміну окремих компонентів серверів. Обслуговування апаратного забезпечення завжди призводить до серйозних збоїв на рівні програмного забезпечення. Втручання в інфраструктуру живлення та охолодження вимагають вимкнення фізичних машин. Аналогічно, заміна апаратних компонентів на сервері (наприклад, заміна несправних модулів DIMM, оновлення процесорів) часто вимагає його попереднього вимкнення. Тому технічне обслуговування обладнання зазвичай

виконується на простоюючих або порожніх машинах. У контексті віртуалізації міграція (завершення/перезапуск, "холодний" і "живий") є найкращим інструментом для евакуації віртуальних екземплярів з уражених машин, що зменшує збої, спричинені цими подіями.

Окрім апаратного забезпечення, вкрай важливо підтримувати програмне забезпечення в актуальному стані. Велика база коду сервісів і систем, розгорнутих у дата-центрі, вимагає частого встановлення виправлень і випуску нових версій програмного забезпечення. Протоколи та сертифікати безпеки вимагають негайного усунення вразливостей у чітко визначені терміни (наприклад, Стандарт безпеки даних індустрії платіжних карток). Крім того, багато сервісів постійно доповнюються новими функціями, щоб залишатися конкурентоспроможними. Google повідомляє, що 70% усіх міграцій екземплярів у GCE пов'язані з оновленням програмного забезпечення, тоді як скромні 6% пов'язані з технічним обслуговуванням обладнання. Далі надається огляд того, як вирішуються питання оновлення програмного забезпечення в центрах обробки даних.

1.6.1 Оновлення програмного забезпечення

Оновлення програмного забезпечення в загальному вигляді вимагає перезавантаження відповідних комп'ютерів. Ця операція може бути дуже руйнівною. Програмне забезпечення потрібно вимкнути, оновити та перезапустити. Під час цих операцій послуги, що надаються, залишаються недоступними. Крім того, після вимкнення дані втрачаються, що може зайняти кілька годин для відновлення стану програмного забезпечення. Оновлення, що циклічно оновлюються, є методом пом'якшення цього збою, але вимагає, щоб програмне забезпечення підтримувало реплікацію портів. Перезавантаження виконується на невеликих групах компонентів за один раз, в той час як їх робоче навантаження перенаправляється на здорові репліки. Зрозуміло, що це рішення не

застосовується до нерезервованого програмного забезпечення. Крім того, завершення оновлення всього парку займає певний час, оскільки групи перезавантажуються послідовно. Отже, за останні десятиліття було докладено багато зусиль для розробки оновлень в реальному часі.

Оновлення в реальному часі, також відоме як "гаряче виправлення" та динамічне оновлення програмного забезпечення, - це метод застосування оновлень без необхідності перезапуску програмного компонента. Цей підхід є найбільш складним для узагальнення через проблему передачі стану. Семантичні зміни в існуючих глобальних структурах даних, наприклад, додавання нової змінної, заважають будь-якому автоматизованому інструменту оновлення в реальному часі коректно перенести стан зі старого програмного забезпечення до нового. Тягар вирішення такої проблеми перекладається на розробників, що відлякує їх від впровадження оновлень у реальному часі. Однак існує клас оновлень, для яких живі оновлення підходять. Патчі безпеки часто передбачають прості модифікації на рівні функцій, наприклад, додавання перевірки меж буферів, а отже, їх можна легко застосувати за допомогою інструментів оновлення в реальному часі.

Оновлення на основі перезапуску програмного компонента залишається найбільш загальним, хоча і руйнівним підходом. Однак цей класичний метод можна вдосконалити різними способами. Час вимкнення та перезапуску можна скоротити, а дані/стан відновити швидше. Такий підхід вимагає, щоб програмне забезпечення було оснащене вбудованим механізмом зупинки/відновлення. Наприклад, Facebook оснащує своє програмне забезпечення для роботи з базами даних у пам'яті процедурою швидкого запуску, яка відновлює зі спільної пам'яті дані під час виконання, залишені попередньою версією. Важливим класом програмного забезпечення в центрах обробки даних, що використовує високоефективну контрольну точку/відновлення, є гіпервізори. У решті розділу ми зосередимося на класичних підходах до оновлення гіпервізорів.

1.6.2 Оновлення гіпервізора

Гіпервізори є складними системами, що складаються з декількох програмних шарів. У 1 розділі було представлено типову класифікацію гіпервізорів на тип 1 і тип 2. Для типу 1 можна припустити, що типові оновлення на рівні ядра або VMM вимагають заміни всього програмного забезпечення гіпервізора. І навпаки, гіпервізори типу 2 виграють від власної модульності, з чистим поділом між VMM та ядром. Наприклад, KVM-QEMU має чітко визначений компонент (QEMU), який виконує емуляцію пристрою, далі йде набір модулів ядра (KVM-модулі), які віртуалізують процесор і пам'ять, і, нарешті, решта Linux, яка мультиплексує апаратні ресурси. Цей поділ можна використовувати, щоб уникнути дорогого перезавантаження всієї хостової ОС. Окрім VMM, інші компоненти мають фундаментальне значення для запуску ВМ. Наприклад, віртуальні комутатори (наприклад, Open vSwitch) реалізують внутрішню мережу для віртуальних машин. Загалом, перезапуск будь-якого програмного забезпечення, що сприяє запуску ВМ, може призвести до тимчасової недоступності (наприклад, недоступність мережі під час оновлення віртуального комутатора), або навіть до повної втрати стану виконання ВМ (наприклад, при перезапуску гіпервізора).

Оновлення гіпервізора у реальному часі. Методи оновлення операційних систем у реальному часі можуть бути використані для виправлення хост-операційної системи у гіпервізорах типу 2. Ці інструменти використовують завантажувані модулі ядра для включення оновлених виправлень на рівні функцій у ядро, вводячи інструкції jmp для перенаправлення потоку виконання на новий код. Незважаючи на незначний час простою, можливі оновлення обмежуються простими виправленнями безпеки через проблему передачі стану.

Існує інструмент оновлення, що спеціалізується на виправленні VMM гіпервізорів на основі KVM: Orthus. Цей інструмент може замінити емулятор (QEMU) та KVM-модулі, при цьому час простою ВМ становить десятки мілісекунд. Orthus змінює архітектуру KVM-модулів, щоб включити можливості

передачі стану між двома послідовними версіями, а також механізм, який використовує спільну пам'ять для передачі стану віртуальної машини між двома версіями QEMU. Оскільки Orthus не може бути націлений на складне оновлення хостової ОС, його основне застосування - виправлення платформ, де віртуальні машини містять виключно прохідні пристрої PCI, обмежуючи помилки, недоліки та поверхню атаки лише QEMU та KVM.

Евакуація хоста. Складні оновлення гіпервізорів вимагають перезавантаження всього хосту, очищення його стану і, як наслідок, завершення роботи всіх запущених екземплярів. Завдяки вбудованій в гіпервізор можливості перевірки/відновлення, а отже, і можливості міграції віртуальних машин, основною стратегією є переміщення віртуальних машин в інше місце. Після переміщення екземплярів у безпечне місце, можливо, на вже оновлений гіпервізор, порожній хост можна перезавантажити. Цей підхід називається евакуацією хоста.

Завершення/перезапуск ВМ є основним підходом до міграції, коли мова йде про репліки або пакетні завдання, які не потребують високої доступності. Однак він не забезпечує прозорості, оскільки не зберігає і не відновлює стан ВМ, який був до завершення роботи. З тієї ж причини він потенційно призводить до тривалого простою через перезапуск і прогрівання екземплярів сервісів. Міграція з попереднім копіюванням в реальному часі має перевагу у вигляді незначного часу простою, але вона створює навантаження на мережеву інфраструктуру під час перенесення вмісту оперативної пам'яті гостей, що займає лічені хвилини навіть у швидких мережах.

Крім того, час простою для великих активних екземплярів пропорційний розміру їх робочого набору, що призводить до помітного простою. При міграції після копіювання екземпляри завжди відчувають незначні простої, хоча і за рахунок значних накладних витрат під час виконання через віддалену вибірку з пам'яті.

Незалежно від методу міграції, доступність ресурсів у центрі обробки даних обмежує кількість хостів, які можуть бути евакуйовані за один і той самий проміжок часу. Через високу завантаженість ресурсів у центрах обробки даних

хости працюють майже на повну потужність. Тому еквівалентна кількість ресурсів, доступних на хості (процесори, пам'ять, графічні процесори тощо), повинна бути зарезервована в іншому місці для розміщення переміщених віртуальних машин. Така погана масштабованість обмежує кількість одночасних хостів, які можуть проходити технічне обслуговування, і затримує впровадження оновленого програмного забезпечення.

Оновлення та міграція в режимі реального часу є компромісом між прозорістю, масштабованістю, коротким часом простою та тривалістю оновлення, що відкриває третій спосіб підходу до оновлення гіпервізорів. У цій дипломній роботі досліджується можливість оновлення гіпервізорів на основі перезапуску. Замість того, щоб використовувати міграцію для захисту доступності віртуальних машин, оптимізується процедура перезапуску та відновлення стану, що призводить до прозорого та легкого оновлення, придатного для великих центрів обробки даних.

2 МІГРАЦІЯ ВІРТУАЛЬНИХ МАШИН У КОНТЕЙНЕРИ ДЛЯ КОНСОЛІДАЦІЇ В ЦЕНТРАХ ОБРОБКИ ДАНИХ

У центрах обробки даних велика кількість віртуальних машин простоює через те, що мережеві сервіси очікують на вхідні з'єднання або підтримують бездіяльність додатків з постійним мережевим сеансом. У публічних хмарах це відбувається, коли користувачі інстанціюють свої DNS, поштові сервери та інші довготривалі сервіси з низьким коефіцієнтом використання. У приватних дата-центрах віртуальні машини, що використовуються розробниками програмного забезпечення для розробки та тестування нових додатків, часто простоюють, оскільки їх рідко вимикають, навіть у неробочий час або під час свят. Це явище було підтверджено у звіті 2017 року, в якому досліджувалися рівні активності віртуальних машин, розгорнутих у близько 2'000 фізичних машин. Автори повідомляють, що лише 20% віртуальних машин демонструють ознаки роботи процесора, мережі, користувача та пам'яті протягом більш ніж 5% всього часу спостереження, тоді як решта 80% простоюють. Близько 30% ВМ взагалі не виявляють жодної активності протягом більше шести місяців.

Простої віртуальних машин призводять до марнування пам'яті, що є наслідком обмеженої ефективності традиційного надлишкового резервування пам'яті в центрах обробки даних (як описано в розділі 1.5.3). Навантаження на процесор рідко досягає піку, що робить надмірне резервування пам'яті ідеальним для ефективного використання часових відрізків, коли ресурс використовується недостатньо. Однак, невикористана пам'ять трапляється рідко, і надмірне резервування пам'яті призводить до активного повернення використаної пам'яті, яка ще не зайнята (пам'ять поза робочим набором віртуальних машин), за допомогою таких методів, як балонування, дедуплікація сторінок та свопінг гіпервізора.

Балонування та дедуплікація сторінок є опортуністичними методами, які не гарантують детермінованого відновлення пам'яті, тоді як гіпервізорний свопінг

страждає від відсутності самоаналізу гостьової ОС і призводить до значного зниження продуктивності у випадку раптових стрибків навантаження. Тому надмірне резервування пам'яті застосовується рідко, що робить його обмежувальним фактором при консолідації серверів.

Ці неактивні ВМ не можна просто вимкнути, оскільки вони можуть містити важливі сервіси з віддаленим доступом, а нові запити можуть надходити несподівано і непередбачувано. Таким чином, необхідно повернути об'єм пам'яті, виділений для непрацюючих ВМ. Існуючі рішення або покладаються на спеціальні проксі-сервери для заміни вимкнених ВМ під час простою, або вимагають радикальних змін на рівні платформи чи гіпервізора. Як наслідок, вони не можуть запропонувати загальну або легко реалізовану методологію.

У цьому розділі представлено SEaMLESS, фреймворк для заміни непрацюючих віртуальних машин на легкі віртуальні мережеві функції (VNF). SEaMLESS мігрує в контейнер стан процесів шлюзу - сукупності процесів, які роблять віртуальну машину доступною ззовні (тобто з мережі). Процеси шлюзу, пересаджені в контейнер, є сутностями, які реалізують легку і безресурсну VNF, що замінює непрацюючу віртуальну машину. VNF діє як проксі-сервер для додатків, який надає кінцевим користувачам відчуття доступності, коли ВМ відключена. VNF перехоплює нові запити кінцевих користувачів, щоб запустити прозоре відновлення відключеної ВМ. Крім того, VNF може обробляти тривіальні запити (наприклад, keep-alive messages), які не вимагають відновлення ВМ. Віртуальні машини можна вимкнути за допомогою цілого ряду існуючих методів, щоб повернути ресурси, які вони використовують. Призупинення в оперативну пам'ять забезпечує швидке відновлення ВМ, але не повертає пам'ять, тоді як призупинення на диск повертає пам'ять, але призводить до тривалої затримки при відновленні. Тому було розроблено новий метод призупинення, який називається suspend-to-swap, для швидкого відновлення ВМ після виявлення активності користувача, при цьому зберігаючи більшу частину пам'яті екземпляра. SEaMLESS розроблений для легкої інтеграції в існуючі хмарні платформи. Suspend-to-swap використовує застарілу систему підкачки гіпервізора, тоді як,

процес, який перетворює неактивну ВМ у VNF, використовує лише інструменти користувацького простору, такі як простори імен Linux та міграція процесів Linux, тобто CRIU (попередньо описано в розділі 1.2.2). Таким чином, SEaMLESS може працювати на рівні орендаря, збільшуючи використання ресурсів без узгодження з оператором центру обробки даних (хоча методи призупинення роботи віртуальних машин повинні бути доступними).

Основний внесок полягає в розробці та впровадженні SEaMLESS. SEaMLESS реалізує нову процедуру перетворення непрацюючої ВМ (з Linux як гостьовою ОС) на полегшену VNF. SEaMLESS спирається на нову техніку відключення ВМ, яка називається suspend-to-swap, розроблену для гіпервізора на основі KVM, який використовує свопінг Linux. Крім того, SEaMLESS можна комбінувати зі старими методами відключення віртуальних машин, такими як suspend-to-RAM або suspend-to-disk, щоб повернути ресурси, заблоковані простоюючими екземплярами. Оцінка SEaMLESS показала:

- після виявлення активності користувача відновлюється початкове середовище ВМ, і сервіси продовжують виконувати свою роботу прозоро з мінімальним впливом на якість роботи;

- можна замінити сотні непрацюючих віртуальних машин відповідними VNF, консолідованими на одному фізичному сервері або віртуальній машині;

- Suspend-to-swap легко відновлює роботу зупиненої ВМ, а також повертає більшу частину пам'яті ВМ, що простоює.

У минулому були зроблені деякі пропозиції щодо вирішення проблеми простою віртуальних машин у центрах обробки даних. Більшість робіт, пов'язаних з неактивними віртуальними машинами, зосереджені на економії енергії шляхом переведення фізичних серверів у режими з низьким енергоспоживанням. Хоча неактивні віртуальні машини не дають серверам переходити в режим глибокого сну (коли це можливо), щоб уникнути порушення доступності сервісів, розгорнутих у віртуальних машинах. Вимкнення невикористовуваних серверів і збільшення

консолідації серверів - це дві сторони однієї медалі. Дійсно, оскільки пам'ять є вузьким місцем для консолідації, повернення пам'яті непрацюючих віртуальних машин призведе до економії енергії. SEaMLESS дозволяє досягти і того, і іншого. Будучи фреймворком для заміни виконання неактивної ВМ легким проксі, ВМ можна відключити, щоб звільнити пам'ять, або, навпаки, консолідувати з сервером, який переходить у сплячий режим.

Часткова міграція віртуальних машин. Це сімейство рішень спрямоване на зменшення енергоспоживання дата-центру шляхом консолідації робочого набору (РН) непрацюючих віртуальних машин на меншій кількості машин, переводячи сервери, на яких зберігається решта пам'яті, в режим низького енергоспоживання. Часткова міграція віртуальних машин базується на припущенні, що РС простоюючих віртуальних машин невелика і статична в часі. Часткова міграція ВМ є варіантом традиційної міграції в реальному часі після копіювання без фонового перенесення сторінок.

Jettison є першою пропозицією, що використовує таку техніку. Jettison консолідує ВС непрацюючих десктопних ВМ на одному сервері (сервері консолідації). Фізичні машини з основною частиною пам'яті віртуальних машин переводяться в сплячий стан ACPI S3 (відомий як "призупинення в оперативну пам'ять" (suspend-to-RAM1). Ці сплячі фізичні машини будуть відновлені, як тільки буде здійснено доступ до сторінки пам'яті за межами WS частково переміщених віртуальних машин (що призведе до збою сторінки). Ті ж автори запропонували наступну роботу, Oasis, рішення для простоюючих ВМ з типовими додатками (Jettison націлений лише на фізичні настільні комп'ютери, на яких розміщена окрема ВМ). Oasis дозволяє уникнути пробудження сплячих фізичних машин при першій же помилці сторінки, використовуючи ідеальний стан апаратного забезпечення на фізичному сервері, що дозволяє надавати помилкові сторінки в режимі енергозбереження. Автори не описують, як саме реалізується такий стан.

Nitu описують схоже рішення, проте детально описують реалізацію режиму низького енергоспоживання сервера, який може обслуговувати віддалені помилки сторінок. Зокрема, автори пропонують новий сплячий стан ACPI, названий Sz,

схожий на S3 (живлення від PCI/PCIe NIC), але з фактично доступною для читання DRAM. Проте, такий стан з низьким енергоспоживанням не підтримується жодним виробником.

Ці рішення є прозорими по відношенню до віртуальних машин, що простоюють. І навпаки, SEaMLESS використовує взаємодію віртуальних машин для точної ідентифікації компонентів всередині екземплярів, які роблять їх доступними ззовні (наприклад, шлюзові процеси), не вимагаючи оцінки WS. Крім того, SEaMLESS безпосередньо втручається між новим запитом користувача і вирішує, які дії слід виконати щодо ВМ, наприклад, відновити екземпляр на тому ж сервері або перенести його в інше місце.

Рішення на основі проксі. Щоб прозоро вимкнути неактивну ВМ, незалежно від того, чи вона виведена в оперативну пам'ять, чи на диск, чи весь хост переведений у режим сну, організація повинна перехоплювати нові запити користувачів, щоб вимкнена ВМ могла відновити роботу і зберегти доступність розміщених на хостінгу сервісів. Цього можна досягти за допомогою проксі-сервера, який перехоплює весь або частину мережі, спрямованої на ВМ. Автори використовували сплячий проксі для кожної підмережі, щоб розбудити клієнтську машину при надходженні мережевого пакету для ВМ. Рішення фільтрує лише TCP-пакети з увімкненим прапором SYN (нове з'єднання), щоб відкинути хибнопозитивні запити (такі як пінги), які не є ознакою нової активності користувача. Однак він не є універсальним і ігнорує виклики пробудження, викликані вже встановленими, але неактивними TCP-з'єднаннями або UDP-додатками. DreamServer також використовує спеціальний проксі для конкретної програми, розміщуючи проксі разом з балансувальником навантаження, який перехоплює запити для вимкненої ВМ.

SEaMLESS також базується на проксі-сервері, VNF, який отримує мережевий трафік, що спочатку був спрямований на простоюючу ВМ. Однак SEaMLESS покладається на шлюзовий процес простоюючої ВМ, пересаджений всередину VNF, щоб коректно відповідати на будь-які тривіальні запити, такі як запити на підтримку працездатності (keep-alive). SEaMLESS відстежує взаємодію між

шлюзовим процесом і контейнером поглинача, який його інкапсулює, щоб виявити справжню активність користувача, яка вимагає відновлення роботи ВМ. Таким чином, SEaMLESS прозоро підтримує будь-який протокол (TCP і UDP) і навіть зашифровані канали, такі як той, що створюється SSH.

Платформні рішення. Автори пропонують Picocenter для повторного витребування пам'яті у непрацюючих віртуальних машин у центрі обробки даних. Замість ВМ, які є громіздкими об'єктами, що несуть великі витрати пам'яті (через наявність повноцінної гостьової ОС), Picocenter використовує спеціальні контейнери, які забезпечують часткову підкачку пам'яті додатків, що працюють всередині. Picocenter використовує модифіковану версію CRIU, утиліти для виконання міграції процесів у Linux для часткової міграції робочого набору додатків, що простоюють у контейнерах, на сервер консолідації (подібно до методів часткової міграції ВМ). Потім Picocenter повертає решту пам'яті, переміщуючи її до сховища (наприклад, Amazon S3 в AWS).

Picocenter має багато спільного з SEaMLESS. Обидва рішення використовують сам додаток, який підтримується в активному стані, щоб підтримувати мережеву присутність сервісу в мережі. Крім того, обидва рішення використовують CRIU для ізоляції частини стану виконання сервісу, до якої є доступ під час простою. Picocenter підтримує резидентний робочий набір додатків, тоді як SEaMLESS підтримує резидентні процеси шлюзу, які є точкою входу до віртуальної машини.

Фундаментальна відмінність полягає у виявленні активності користувача. Що стосується часткової міграції ВМ, Picocenter ліниво отримує замінені сторінки після збоїв у непрацюючому додатку. Він використовує інформацію, отриману з DNS-запитів, щоб виявити нові запити користувачів (розшифровуючи ім'я хоста програми), щоб запустити повну підкачку пам'яті контейнера. SEaMLESS пісочниці виконує процеси шлюзу, що простоюють, спостерігаючи за взаємодією між процесами і середовищем, щоб оцінити, чи вимагає нова активність користувача відновлення віртуальних машин.

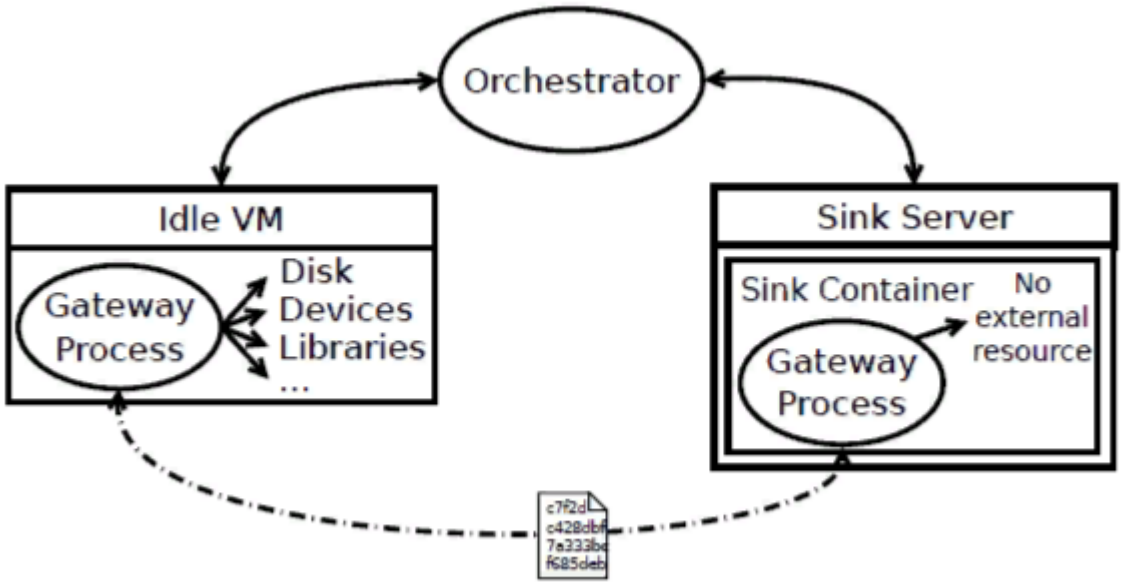


Рисунок 2.1 - Компоненти та архітектура SEaMLESS

2.1 Вирішення проблеми з непрацюючою віртуальною машиною

Віртуальні машини в дата-центрах доступні через кілька процесів, які очікують на вхідні з'єднання або запити, прослуховуючи мережеві порти. Ми називаємо кожен з цих процесів процесами шлюзу. У цьому розділі показано, як SEaMLESS мігрує кожен шлюзовий процес з віртуальної машини, яка простоює досить довго, до контейнера-відстійника, що становить VNF, який може замінити простоюючий екземпляр.

Якщо говорити більш детально, використовуючи рисунок 2.1 як ілюстрацію, віртуальна машина зазвичай містить великий набір процесів, включаючи процеси шлюзу. Як тільки віртуальна машина виявлена як така, що простоює, SEaMLESS-оркестратор - агент, відповідальний за синхронізацію міграції - переносить процес шлюзу до контейнера поглинача з усіма його станами, включаючи будь-який відкритий сокет, фактично перетворюючи його на віртуальну мережеву функцію. Зазвичай, процеси обробляють елементи доступу та відображення, такі як файли, пристрої або бібліотеки. Ці зовнішні ресурси не копіюються в контейнер

поглинача, щоб забезпечити легкість середовища. Коли VNF виявляє ознаки активності користувача, процес шлюзу буде перенесено назад з контейнера поглинача і відновлено у вихідному середовищі віртуальної машини, щоб виконати запити користувача.

SEaMLESS є повністю прозорим з точки зору кінцевого користувача. Дійсно, міграція процесів шлюзу між віртуальною машиною та контейнером поглинача відбувається швидше, ніж міграція всієї віртуальної машини, а завдяки тому, що процеси шлюзу працюють, коли віртуальна машина призупинена або вимкнена, SEaMLESS може підтримувати будь-які постійні з'єднання, що простоюють, як на транспортному, так і на прикладному рівні.

Підрозділ 2.1.1 присвячено створенню та структурі контейнера поглинача для розміщення шлюзового процесу. Далі, у підрозділі 2.1.2, описуються процедури міграції для перетворення неактивної ВМ на VNF, а також методику виявлення ознак активності користувача на VNF, що спонукає до зворотного процесу відновлення ВМ.

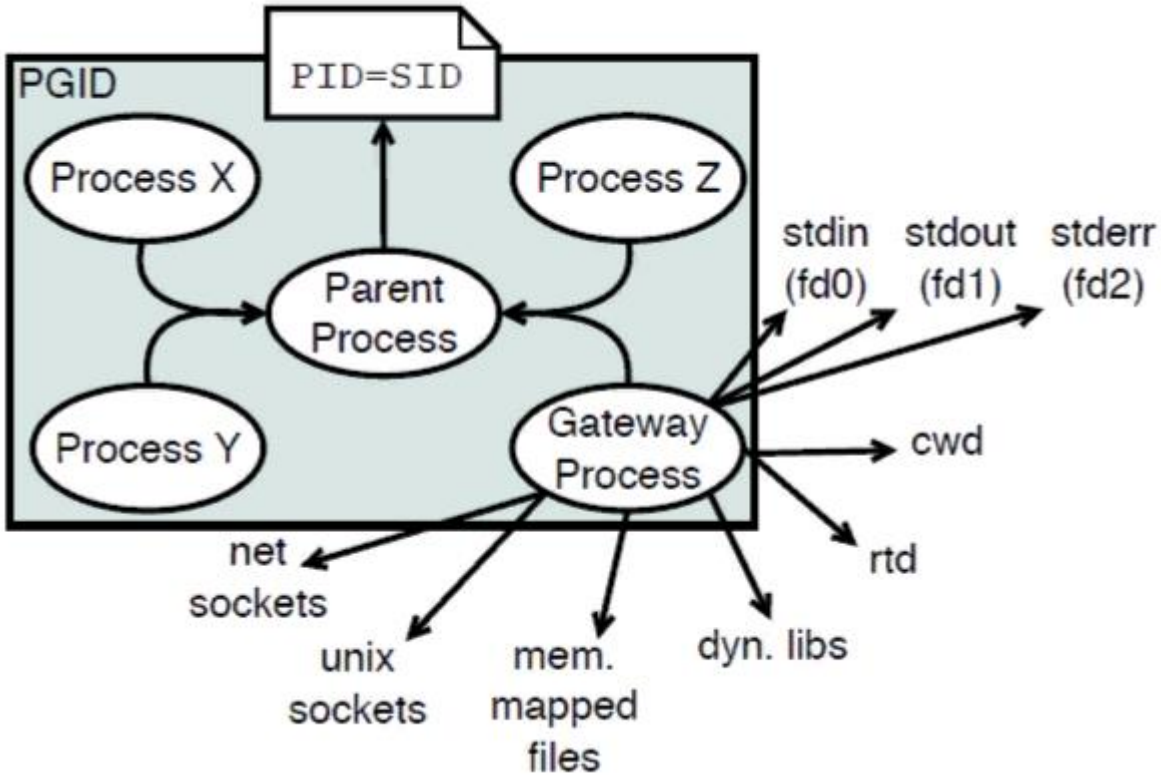


Рисунок 2.2 - Можлива екосистема процесу шлюзу та її ресурси

2.1.1 Процес шлюзу VNF

VNF має поводитися по відношенню до кінцевих користувачів так само, як і простоююча віртуальна машина. Тому ми мігруємо кожен процес шлюзу з його початкового середовища, всередині віртуальної машини, до контейнера поглинача. Процеси шлюзу повинні працювати в середовищі, яке забезпечує ізоляцію, а також є легким з точки зору споживання пам'яті та процесора. З цих причин було вирішено реалізувати контейнер поглинача за допомогою просторів імен Linux, що лежить в основі технології контейнерів Linux, таких як Docker.

Як показано на рисунку 2.2, коли процес шлюзу працює у віртуальній машині, він є частиною екосистеми процесів, яка формується, серед іншого, його ідентифікатором процесу (PID), маячками файлової системи, дескрипторами файлів, бібліотеками тощо. Наша мета в SEaMLESS - перенести лише сам процес шлюзу, без цих побічних елементів. Однак, ця екосистема необхідна для того, щоб процес шлюзу продовжував працювати без збоїв.

По-перше, процес ідентифікується за допомогою унікального PID. Його значення має бути збережено, щоб гарантувати успішну міграцію, чого можна досягти, покладаючись як на монтування, так і на простір імен PID у нашому контейнері-"поглиначі" для ізоляції файлових систем proc і sys. Отже, коли декілька процесів з декількох віртуальних машин переносяться на сервер-стік, ці процеси шлюзу можуть мати однаковий PID.

По-друге, процеси покладаються на зовнішні, динамічно зв'язані спільні бібліотеки, які завантажуються, вивантажуються і зв'язуються з процесом під час виконання. Якщо ці бібліотеки недоступні, програма може зіткнутися з помилкою сегментації. Оскільки віртуальні машини, розгорнуті в середовищах інфраструктури як послуги (IaaS), створюються з шаблонного образу, SEaMLESS може використовувати цей самий образ для інстанціювання сервера поглинання, а саме віртуальних машин, на яких розміщуються всі VNF з одного шаблону

віртуальної машини. Зокрема, це запобігає перенесенню бібліотек між віртуальною машиною та сервером-відстійником.

По-третє, процес відкриває багато дескрипторів файлів для стандартних пристроїв виводу та вводу (наприклад, stdout, stderr та stdin), а також звичайні файли, каталоги, файли пристроїв та інші елементи файлової системи. Щоб впоратися з цими зовнішніми ресурсами, необхідними для роботи шлюзу, було вирішено вказати на них у спеціально створених фіктивних файлах у контейнері sink. Ми використовуємо простір імен монтування, щоб забезпечити узгодженість між контейнером-відстійником і деревом файлової системи у вихідній віртуальній машині. Слід зауважити, що процес шлюзу отримує доступ до зовнішніх ресурсів лише у випадку активності користувача, що спричиняє міграцію процесу шлюзу назад до початкової ВМ. Таким чином, фіктивні файли у контейнері поглинача не призведуть до помилок виконання під час роботи процесу шлюзу.

По-четверте, процес зазвичай залежить від сокетів Unix або мережевих сокетів для зв'язку із зовнішнім середовищем (тобто мережею) або іншими робочими процесами. SEaMLESS має підтримувати існуючі з'єднання (наприклад, встановлені SSH-сесії) навіть після зупинки віртуальної машини. Кінцеві користувачі повинні залишатися підключеними до процесу шлюзу після міграції процесу, без розриву з'єднання або втрати даних. Щоб досягти цього, а також уникнути використання кастомізованих ядер з незрозумілими патчами, ми покладаємося на CRIU (Checkpoint/Restore In Userspace), сучасне програмне забезпечення, що активно розробляється, для забезпечення міграції процесів у просторі користувача. Гачки ядра, які використовує CRIU, починаючи з версії Linux 3.3, об'єднано в основну лінію ядра.

CRIU може зупинити певний процес, щоб пізніше відновити його, зберігши всі сокети та канали. SEaMLESS використовує CRIU для запису на диск стану процесів шлюзу, а потім відновлює їх з їхньої контрольної точки разом з усіма встановленими мережевими з'єднаннями (наприклад, зберігаючи вже встановлений сеанс SSH), або на контейнері-стікері, або на віртуальній машині.

Підсумовуючи, ми перенесемо процес шлюзу у віртуальній машині до контейнера поглинача. Контейнер sink складається з просторів імен Linux: простору імен PID, простору імен mount і простору імен мережі, заповненого двома віртуальними ethernet-пристроями (veth), які підтримуються за замовчуванням у поточних дистрибутивах Linux. Один veth підключається до робочої локальної мережі і повинен відображати мережеву конфігурацію (таблиці маршрутизації, IP-адреси) у непрацюючій ВМ. Другий veth, підключений до керуючої локальної мережі, використовується для передачі всередину контейнерів-відстійників образу контрольної точки процесу шлюзу.

2.1.2 Міграційні процедури

У цьому розділіпредставлено організацію міграції процесів шлюзу між контейнером-відстійником та віртуальною машиною (і навпаки). Необхідно зауважити, що міграція процесів шлюзу координується з перемаршрутизацією мережевих пакетів для перенаправлення трафіку до VNF (і назад до ВМ).

Міграція з віртуальної машини на сервер поглинача. Коли оркестратор виявляє, що віртуальна машина простоює, він запускає створення VNF. Міграція процесів шлюзу всередину контейнера поглинача відбувається у такий спосіб, як описано нижче.

Крок №1: Оркестратор просить робочу віртуальну машину створити дамп стану процесу шлюзу у файл. Якщо виявлено будь-яку активність користувача, ВМ надсилає оркестранту повідомлення про переривання міграції, щоб скасувати її без втрати повідомлень.

Крок №2: Після завершення встановлення контрольних точок робоча ВМ надсилає свій стан на сервер-"поглинач".

Крок №3: Контейнер поглинача відновлює процес шлюзу.

Крок №4: Мережева інфраструктура налаштовується на перенаправлення пакетів на контейнер поглинача.

Крок №5: Оркестратор переходить до відключення ВМ (призупинення до оперативної пам'яті тощо).

Міграція з сервера-поглинача на віртуальну машину. Процедура міграції процесів шлюзу назад близька до зворотної процедури, описаної раніше. Однак, щоб запобігти довгій затримці повторної передачі, яка виникає у TCP (TCP Retransmission Timeout) при наступній спробі повторної передачі, ми буферизуємо пакети під час міграції процесів шлюзу. Міграція процесів шлюзу назад на віртуальну машину відбувається у такий спосіб, як описано нижче.

Крок №1: При виявленні активності користувача на VNF, оркестратор сигналізує, що відповідна ВМ має бути відновлена.

Крок №2: За допомогою черги мережевих фільтрів (NFQUEUE), доступної в поточних дистрибутивах Linux, на фізичному сервері, де розміщена ВМ, розгортається буфер для зберігання пакетів, що надсилаються до ВМ. У той же час, мережева інфраструктура налаштовується на маршрутизацію пакетів до ВМ, а не до VNF.

Крок №3: Процес шлюзу в контейнері in the sink скидається.

Крок №4: Після перезавантаження ВМ образ контрольної точки процесу шлюзу завантажується на ВМ.

Крок №5: Процеси шлюзу відновлюються з оновленим станом (після виявлення активності користувача).

Крок №6: Буферизовані пакети звільняються з NFQUEUE, фільтр знищується, і зв'язок тепер обробляється віртуальною машиною.

Вирішення проблем маршрутизації SEaMLESS покладається на програмно-визначену мережу (SDN) для реконфігурації мережевої інфраструктури з метою перенаправлення пакетів від віртуальної машини до процесів шлюзу в контейнері поглинача. Оркестратор повторно запитує контролер SDN, щоб ввести спеціальні правила в комутатори SDN.

Інтерфейс veth контейнера-відстійника або налаштований на ту саму MAC-адресу, що й інтерфейс ВМ, або MAC-адреса призначення будь-якого пакету повинна бути переписана. Якщо обладнання SDN недоступне, ми припускаємо, що дата-центри, які надають IaaS, підтримують мережеві операції з перенаправлення пакетів у разі міграції ВМ в реальному часі. SEaMLESS підключається до тієї ж інфраструктури перенаправлення, що і традиційна міграція в реальному часі для вирішення проблеми переміщення запущених екземплярів.

2.1.3 Виявлення активності користувачів

Після розгортання контейнера поглинача, процеси шлюзу всередині нього обробляють будь-яку першу комунікацію з кінцевими користувачами. На цьому етапі виникають два питання: як виявити активність користувача? Як запобігти перехопленню VNF повного зв'язку з кінцевими користувачами (що може призвести до помилок), замість того, щоб відновити і передати процесам шлюзу початкову ВМ?

По-перше, ми підкреслюємо, що не всі вхідні мережеві пакети є прелюдією до дій користувача, таких як ICMP, ARP або меседжі keep-alive на рівні додатків. Такі запити не повинні викликати відновлення роботи простоюючої ВМ. Зазвичай на повідомлення на транспортному рівні або нижче безпосередньо відповідає стек протоколів, доступний у контейнері поглинача (наприклад, ICMP, ARP). Однак, повідомлення про підтримання працездатності на прикладному рівні вимагають виконання процесів шлюзу. Однак ми припускаємо, що такі повідомлення досить прості, щоб не вимагати для обробки повноцінного контексту у віртуальній машині. SEaMLESS реалізує точне рішення для успішної обробки повідомлень keep-alive додатків без необхідності відновлення ВМ.

Щоб зрозуміти, як SEaMLESS виявляє нетривіальну активність кінцевого користувача, потрібно зрозуміти поведінку процесу шлюзу після отримання запиту. Якщо процес шлюзу використовує протокол TCP, то повідомлення користувача може містити запит на встановлення нового з'єднання з сервером, перевірку існування каналу прикладного рівня (наприклад, повідомлення SSH keep-alive) або запит на отримання зовнішніх даних (недоступних у контейнері поглинача). Якщо процес шлюзу використовує UDP, повідомлення користувача може містити перевірку/оновлення членства на рівні програми або запит на зовнішні дані (знову ж таки, недоступні в контейнері поглинача). Проаналізувавши декілька додатків шлюзових процесів, було виявлено, що нове TCP-з'єднання повертає системний виклик accept() (або інші пов'язані системні виклики, такі як poll, epoll), викликаний процесом шлюзу на мережевому сокеті, прив'язаному до порту.

SEaMLESS створює контейнер зливної ями з фіктивними файлами як заміну всіх об'єктів файлової системи, які існують у контексті початкової віртуальної машини, але втрачають сенс після перенесення за її межі. Слід зауважити, що це правило поширюється не лише на об'єкти файлової системи. Інші аспекти середовища користувацького простору також підпадають під дію цього правила: об'єкти IPC (UNIX або мережеві сокири), а також об'єкти, процеси (створення або надсилання сигналу процесу: fork, clone, kill), змонтована файлова система (mount, umount) тощо. Ми додаємо до білого списку підмножину цих об'єктів користувацького простору, які мають у контейнері змиву семантично еквівалентний стан відповідному об'єкту у віртуальній машині. Наприклад, мережеві сокети в контейнері відновлюються з тим самим станом (IP-адреса, порт прослуховування, встановлені з'єднання), що й у віртуальній машині.

Отже, процеси шлюзу можуть взаємодіяти з такими об'єктами з білого списку і очікувати семантично еквівалентного результату, як якщо б вони виконувалися у віртуальній машині. Можна припустити, що будь-який запит кінцевого користувача, який спонукає процес шлюзу взаємодіяти лише з об'єктами контексту з білого списку, є тривіальним і не призводить до відновлення початкової ВМ.

З іншого боку, коли процес шлюзу намагається отримати доступ до фіктивної сутності, яка не відображає стан оригінальної ВМ, виконання негайно зупиняється, відновлюється оригінальна ВМ, і обчислення продовжуються у вихідному середовищі, де вони можуть завершитися.

Отже, для точного виявлення дій кінцевого користувача, які спонукають отримати доступ до частини стану, не експортованої до контейнера поглинача, SEaMLESS покладається на відстеження системних викликів через процеси шлюзу всередині контейнера поглинача. Трасування системних викликів можна легко виконати у Linux за допомогою бібліотеки ptrace, базового фреймворку, який використовують налагоджувачі, такі як GNU Debugger (GDB). Технічно, за допомогою бібліотеки ptrace для кожного (вказаного) системного виклику ядро перехоплює процес шлюзу (за допомогою сигналу SIGTRAP) і сповіщає SEaMLESS про цей виклик. А саме, процес шлюзу зупиняється на кожній пастці.

Після того, як було проаналізовано одержувача системного виклику, можна перезапустити виконання шлюзового процесу за допомогою сигналу перезапуску (SIGCONT), якщо евристика SEaMLESS вирішить, що контейнер поглинача може обробити повідомлення самостійно. В іншому випадку SEaMLESS повністю зупиняє виконання шлюзового процесу (за допомогою сигналу SIGSTOP) і запускає процедуру міграції шлюзового процесу з контейнера-стікача на ВМ. Таким чином, саме ВМ, а не контейнер поглинача, відповідатиме на запит користувача.

2.2 Вирішення проблеми марнотратства пам'яті

SEaMLESS замінює простоюючу ВМ легким безресурсним контейнером поглинача, реалізуючи VNF, який виконує роль простоюючої ВМ. Отже, поки процеси шлюзу очікують на вхідні запити користувачів, незадіяні ресурси, що належать ВМ, можуть бути звільнені. Слід зауважити, що ВМ, яка простоює,

зрештою, відновлює свою роботу, коли з'являється нова активність користувача. Отже, стан виконання ВМ має бути збережено, поки отримані ресурси (процесор, пам'ять) будуть повернуті.

Більшість гіпервізорів підтримують два режими вимкнення віртуальної машини зі збереженням стану її виконання: призупинення до оперативної пам'яті та призупинення до диска. Suspend-to-RAM заморожує потоки, у яких виконується робота центрального процесора віртуальної машини. Вміст файлу пам'ять віртуальної машини залишається резидентною на фізичному хості. Пам'ять не звільняється, але відновлення роботи віртуальної машини займає лише кілька мілісекунд. Suspend-to-disk переносить весь стан в енергонезалежну пам'ять. Отже, затримка при відновленні віртуальної машини довша, ніж при зупинці в оперативну пам'ять, і пропорційна розміру пам'яті віртуальної машини, що зупинилася.

Затримка відновлення залежить від енергонезалежного носія, на якому зберігається стан зупинки (HDD, SSD, NVMe), а також від процедури, що використовується гіпервізором. Наприклад, QEMU-KVM перед перезапуском потоків завантажує всю пам'ять віртуальної машини. VMware ESXi використовує підхід лінивого відновлення. Потоки запускаються з не повністю резидентною пам'яттю в оперативній пам'яті, генеруючи помилки сторінки, які спонукають гіпервізор завантажити на вимогу відсутню сторінку.

Було розроблено рішення під назвою suspend-to-swap, яке здатне повернути більшу частину пам'яті, що використовується непрацюючою віртуальною машиною, забезпечуючи при цьому швидке відновлення. Suspend-to-swap розроблено для гіпервізора QEMU-KVM. Suspend-to-swap відповідає цілям SEaMLESS: уникати будь-яких модифікацій на рівні гіпервізора/ядра, використовуючи лише доступні функції Linux для легкої інтеграції в існуючі платформи.

Дана технологія suspend-to-swap поєднує в собі функцію підкачки та підкачки гіпервізора, яка доступна у всіх основних гіпервізорних рішеннях. Він також

використовує групи Linux cgroups для примусового вивільнення пам'яті віртуальної машини з підкачки. Suspend-to-swap працює наступним чином.

Щоб вивільнити пам'ять з неактивної віртуальної машини, ми спочатку надуваємо кульку всередині віртуальної машини, щоб відновити невикористану пам'ять у гостьовій машині. Надування кульки за межі доступної вільної пам'яті може призвести до критичного тиску на пам'ять у гостьовій ОС, що може спричинити помилки відсутності пам'яті.

Використовуючи cgroups у гіпервізорі (QEMU-KVM), ми обмежуємо максимальну пам'ять, дозволену для ВМ, до 10 МБ. Це призводить до спрацьовування свопінгу гіпервізора. Гіпервізор повертає пам'ять ВМ, підкачуючи сторінки пам'яті, щоб задовольнити обмеження у 10 МБ. Після цього кроку пам'ять простоюючої ВМ значно зменшується. Ми знімаємо обмеження на пам'ять cgroup на ВМ і виконуємо фіктивне відновлення процесу шлюзу. Мета цього фіктивного відновлення - прозоро розігріти (підкачати) сторінки пам'яті, які зберігають дані, необхідні процесу шлюзу під час відновлення, включаючи деяку пам'ять ядра гостьової ОС. Цей крок дозволяє пришвидшити час відгуку після відновлення простоюючої ВМ. Наприкінці попереднього кроку частина пам'яті, зайнятої для виконання фіктивного відновлення, знову стає вільною. Тому ми надуваємо кульку ще раз, щоб повністю відновити її. Нарешті, віртуальна машина призупиняється (suspend-to-RAM). Виконання наведених вище кроків призведе до того, що розмір пам'яті неактивної ВМ буде меншим за 600 МБ.

Можна зауважити, що чисте використання гіпервізорного підкачування (примусове за допомогою обмежень пам'яті cgroups) не є найкращим варіантом. Підкачка гіпервізора призводить до неефективного використання пам'яті рекультивації, коли вільна пам'ять у непрацюючій віртуальній машині потрапляє до області підкачки. Тому дуже важливо поєднувати свопінг гіпервізора з превентивним спустошенням, щоб мінімізувати кількість невикористаних сторінок, які закінчуються у просторі підкачки. Після відновлення віртуальної машини, куля повністю здувається, щоб повернути віртуальним машинам їх номінальну пам'ять.

Представлена функція suspend-to-swap забезпечує швидке відновлення віртуальної машини, включаючи активацію процесу шлюзу, як показано в оцінці нашої затримки.

3 ОЦІНКА ПРОДУКТИВНОСТІ ФРЕЙМВОРКУ SEAMLESS

У цьому розділі ми оцінюємо продуктивність SEaMLESS з точки зору сприйняття кінцевим користувачем якості обслуговування (QoE) та отриманої економії пам'яті. У розділі 3.2 ми оцінюємо затримку, спричинену основними компонентами SEaMLESS, коли процес шлюзу мігрує назад з контейнера поглинача. У розділі 3.3 ми оцінюємо вплив нашої стратегії "призупинити-замінити". Масштабованість SEaMLESS аналізується в розділі 3.4, її реактивність - в розділі 3.5, і, нарешті, ми надаємо інформацію про обсяг вивільненої пам'яті за допомогою SEaMLESS в розділі 3.6.

3.1 Мережевий тестовий стенд

Було протестовано даний прототип SEaMLESS на одному з кластерів Grid5000, масштабного тестового майданчика для дослідницьких експериментів з розподіленими системами. Було використовувано сервери Dell PowerEdge R430, оснащені 2 процесорами Intel Xeon E5-2620, 32 ГБ пам'яті, 2 жорсткими дисками Dell PERC H330 в RAID-0 та мережевими картами Ethernet 10 Гбіт/с.

Тестовий стенд складається з трьох фізичних машин. На першій знаходиться оркестр. На другій - контейнер для змиву. На третій - простоюючі віртуальні машини. Мережева інфраструктура (де проходить трафік кінцевого користувача) використовує тунелі VxLAN. Перенаправлення пакетів досягається за допомогою правил OpenFlow (SDN), закладених у комутаторах Open vSwitch, розгорнутих на кожному хості.

3.2 Вплив на якість досвіду

Повний процес відновлення передбачає виконання наступних фаз, кожна з яких впливає на період недоступності сервісу:

- дамп стану шлюзового процесу та стиснення;

- передача зображення;

- розкладання зображення та відновлення шлюзового процесу;

- додатковий час синхронізації між контейнером-відстійником, оркестрантом та віртуальною машиною для дампу, передачі та відновлення шлюзового процесу.

Щоб оцінити час, необхідний SEaMLESS для відновлення процесу шлюзу на ВМ, було проведено кілька тестів з різними процесами шлюзу на їхню конфігурацію за замовчуванням. Результати наведено в Таблиці 3.1, де ми переносимо затримки через дампінг-стиснення, передачу та розпакування-відновлення процесу шлюзу.

Стовпчик "Всього" відповідає сумі всіх попередніх затримок. Час відгуку (позначений як "Час реакції") відповідає спостережуваній затримці між першим пакетом, надісланим клієнтом на віртуальну машину, і першим пакетом, надісланим назад з відновленої машини. Час відгуку складається з компонентів зі стовпчика "Всього", а також часу, витраченого на синхронізацію різних подій (наприклад, сигналізація про активність користувача від контейнера зливу до оркестратора). Стовпчики "Розмір зображення" і "Розмір VNF" відповідають розміру стисненого зображення процесу шлюзу (у файлі tar.lzo) і розміру контейнера-збірника, у якому розміщено такий процес шлюзу, відповідно. Зображення безпечно передаються за допомогою команди scp, як це робиться у реальному центрі обробки даних.

Таблиця 3.1 - Час відгуку реальних додатків шлюзових процесів.

Додаток	Основні завдання Час (с)			Всього (с)	Час реакції (с)	Зображення Розмір (МБ)	VNF Розмір (МБ)
	Дамп	Переказ	Відновлення				
Dropbear	0.04	0.12	0.04	0.20	0.41	0.12	11.18
Vsftpd	0.12	0.11	0.05	0.28	0.41	0.11	7.81
OpenSSH	0.13	0.13	0.07	0.33	0.46	0.13	15.93
Lighttpd/PHP	0.16	0.29	0.16	0.61	0.71	0.29	46.43
Apache2/PHP	0.23	0.43	0.24	0.90	0.95	0.43	67.52
Tomcat	0.46	1.17	0.38	2.02	2.16	1.17	206.96

З результатів, наведених у таблиці 3.1, ми бачимо, що програмою з найбільшим файлом зображення є Tomcat (1,172 МБ), за ним йде Apache 2 з увімкненим PHP (0,428 МБ). Найменший розмір має vsftpd (FTP/SFTP сервер) - лише 0,107 МБ. Ми хотіли б зазначити, що PHP-додаток, який використовується з Apache 2, не впливає ні на розмір контейнера для збору даних, ні на розмір зображення процесу шлюзу. Дійсно, образ процесу шлюзу включає лише основні бібліотеки PHP-рушія, а не самі PHP-додатки, які завантажуються як зовнішні ресурси, коли це необхідно.

Кількість робочих процесів SSH як для Dropbear, так і для OpenSSH (і, відповідно, розмір образу процесу) залежить від кількості встановлених SSH-сесій. У наших тестах ми підтримували одне єдине SSH-з'єднання, в результаті чого було створено і відновлено два процеси шлюзу (тобто основний процес демона і робочий процес).

З наших тестів у Таблиці 3.1 ми бачимо, що час відгуку SEaMLESS загалом менший за 1 секунду (за винятком Tomcat). Це менше, ніж час відгуку, який зазвичай очікують кінцеві користувачі. Ми дійшли висновку, що затримка, спричинена SEaMLESS, має незначний вплив на якість роботи кінцевого користувача.

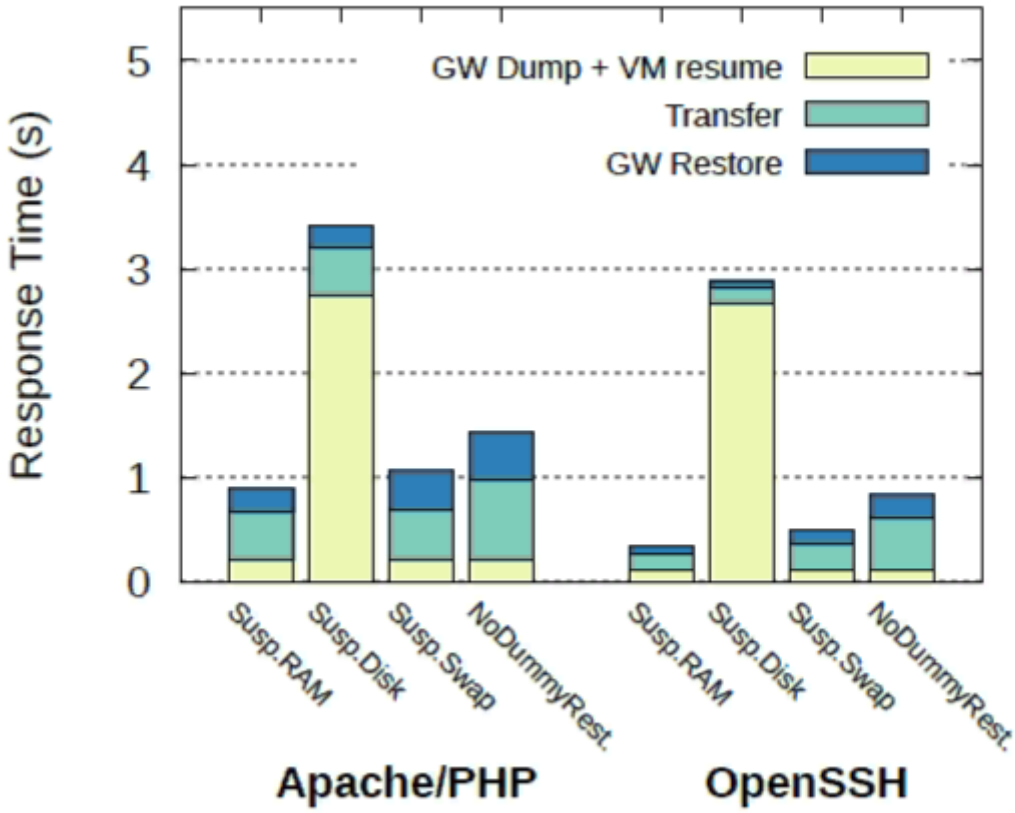


Рисунок 3.1 - Час відгуку при використанні різних методів відключення на віртуальній машині об'ємом 3 ГБ

3.3 Вплив призупинення на заміну

У розділі було розглянуто декілька стратегій вимкнення ВМ, а саме: вимкнення в оперативну пам'ять, вимкнення на диск та наш варіант вимкнення на підкачку. На рисунку 3.1 показано час відгуку процесів Apache 2-PHP та шлюзу OpenSSH при різних підходах до вимкнення ВМ.

На рисунку 3.1 можна побачимо, що suspend-to-RAM має найшвидший час відгуку (близько 1 секунди для Apache 2-PHP і менше 0.5 секунди для OpenSSH). Однак, призупинення в оперативну пам'ять неефективне для звільнення пам'яті. Призупинення на диск має найгірший час відгуку: час лінійно зростає зі

збільшенням розміру пам'яті віртуальної машини, що призводить до 3-секундного часу відгуку для екземпляра об'ємом 3 ГБ.

Представлений в роботі метод suspend-to-swap поєднує в собі найкраще з обох варіантів. Показано час відгуку suspend-to-swap з відновленням процесу фіктивного шлюзу та без нього. Без відновлення фіктивного шлюзу час відгуку становить близько 1,5 секунд для Apache 2-PHP і 0,9 секунд для OpenSSH. Зауважте, що suspend-to-swap фактично повертає більшу частину пам'яті віртуальної машини.

Suspend-to-swap з відновленням процесу фіктивного шлюзу (dummy gateway process restoration) значно покращує затримку, що призводить до часу відгуку близько 1 і 0.5 секунди для Apache 2-PHP і OpenSSH відповідно.

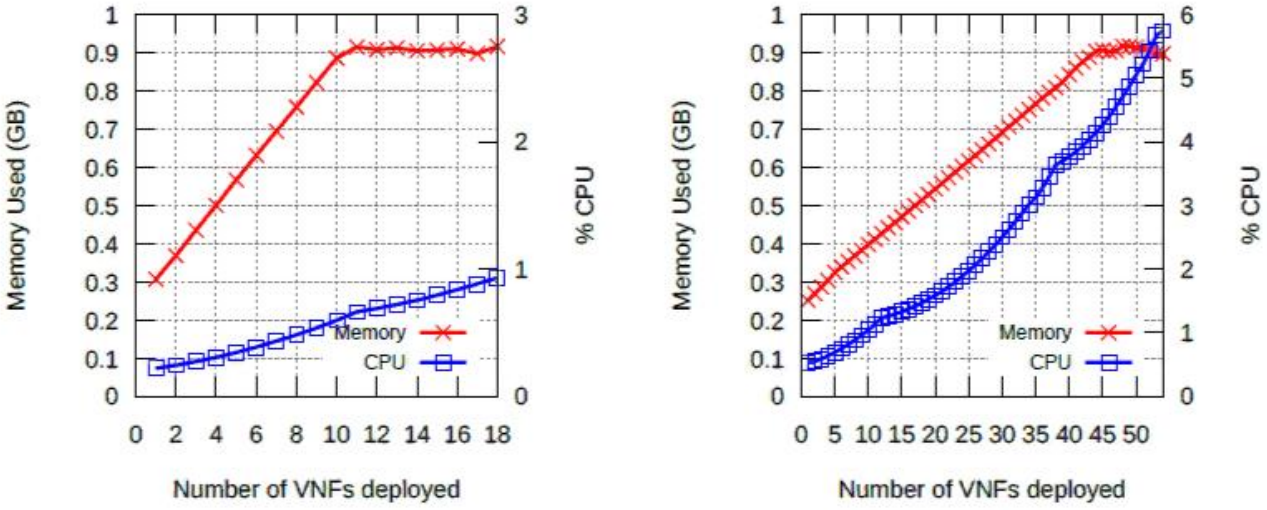


Рисунок 3.2 - Використання оперативної пам'яті та центрального процесора в залежності від кількості розгорнутих VNF

3.4 Масштабованість сервера-відстійника

У цьому розділі було зосереджено на масштабованості контейнера-відстійника з точки зору використання процесора і пам'яті. Рисунки 3.2a та 3.2b

ілюструють споживання пам'яті та процесорного часу віртуальної машини, на якій розміщено контейнер-стік, при розгортанні виключно 2-PHP VNF Apache або OpenSSH VNF.

Як і очікувалося, використання процесора та пам'яті лінійно зростає зі збільшенням кількості розгорнутих контейнерів поглиначів. Кількість VNF, які можна створити, обмежена пам'яттю: розгортання більше 10 VNF Apache 2-PHP досягає межі в 1 ГБ (використання пам'яті 100%). З іншого боку, завантаження процесора залишається нижче 1%. OpenSSH VNF займають менше пам'яті. Можна створити 43 OpenSSH VNF до насичення 1 ГБ пам'яті, при цьому споживання процесора залишається нижчим за 6%.

Ці результати показують, що на сервері дата-центру з 32 ГБ оперативної пам'яті можна розмістити близько 320 контейнерів зливів Apache 2-PHP або 1376 контейнерів зливів OpenSSH, перш ніж виникне проблема з підкачкою пам'яті. Ці цифри значно перевищують кількість простоюючих віртуальних машин, які можуть бути одночасно запущені на тому ж сервері з 32 ГБ оперативної пам'яті.

3.5 Реактивність

Щоб оцінити реактивність SEaMLESS, було проведено стрес-тести, що складаються з одночасного відновлення роботи групи простоюючих ВМ. Розгортаютьчя VNF на одній ВМ з 5 ГБ пам'яті та 1 vCPU. На VNF розміщуються процеси шлюзу OpenSSH та Apache 2-PHP. Кожен тест було виконано 20 разів.

На рисунку 3.3 показано час відгуку (без урахування часу відновлення роботи ВМ) для одночасної міграції зростаючої кількості контейнерів. Можна спостерігати лінійне збільшення часу відгуку. Для одного Apache 2-PHP для одного контейнера складає близько 0,9 секунди; 8 секунд для 10 контейнерів; і 13 секунд для 20 контейнерів. Для OpenSSH для одного контейнера потрібно трохи менше 0,5 секунди; близько 3 секунд для 10 контейнерів; і близько 6 секунд для 20

контейнерів. Слід зауважити, що спостережуваний час відгуку мало варіюється, з дуже вузькими міжквартильними діапазонами. Швидкість запису/читання з диска домінує над часом відгуку. Тестований хост оснащений механічним диском із значно обмеженою пропускною здатністю порівняно з більш швидкими SSD. Можна рекомендувати використовувати SSD або швидші носії інформації (наприклад, диски оперативної пам'яті) для створення дампа та відновлення процесів шлюзу.

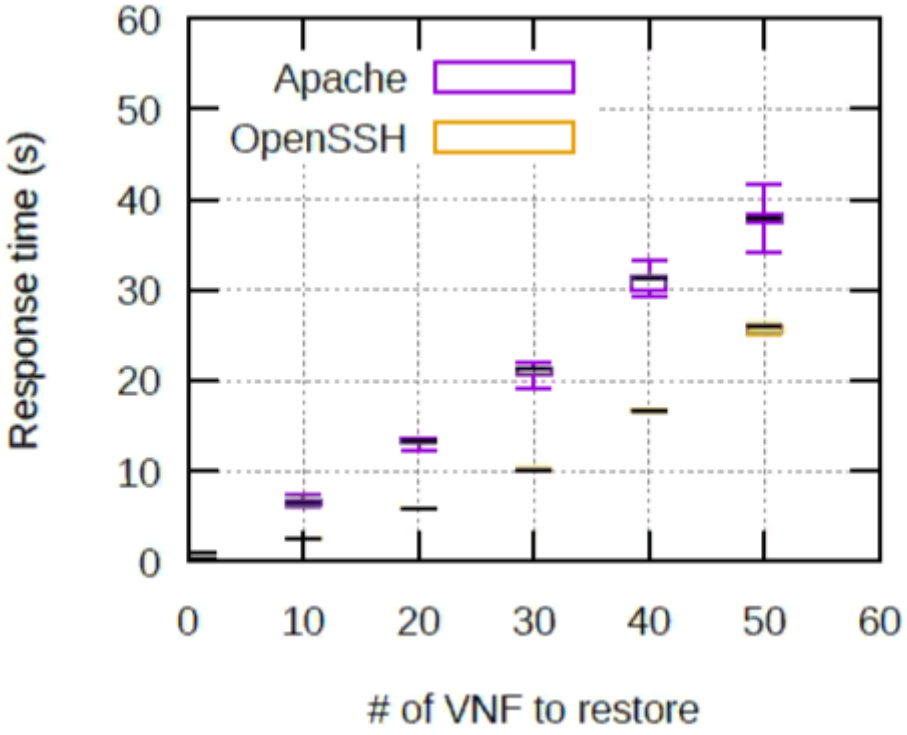


Рисунок 3.3 - Час відгуку в залежності від кількості паралельних OpenSSH VNF з активністю користувачів

3.6 Економія пам'яті

Основною метою SEaMLESS є вивільнення пам'яті з непрацюючих віртуальних машин. Щоб визначити обсяг оперативної пам'яті, який можна

звільнити, необхідно оцінити очікуване споживання пам'яті віртуальною машиною зі зменшеним обсягом пам'яті та очікуване споживання пам'яті її відповідним контейнером-відстійником. Призупинення-відновлення підкачки обчислює розмір пам'яті, що залишається резидентною на хості, наступним чином.

Віртуальна машина об'ємом 1,7 ГБ займає в резидентному режимі 474 МБ. ВМ об'ємом 34 ГБ займає близько 598 МБ. Розмір контейнера для зливу залежить від розгорнутих процесів шлюзу. Apache 2-PHP потребує близько 68 МБ, тоді як OpenSSH - лише 16 МБ.

У Таблиці 3.2 показано звільнену пам'ять для типових розмірів оперативної пам'яті ВМ відповідно до типів екземплярів Amazon EC2, припускаючи, що ВМ виділяють всю свою пам'ять.

Таблиця 3.2 - Потенційна економія пам'яті для поширених типів екземплярів AWS, за умови розміру VNF 67 МБ.

Тип AWS	Розмір (ГБ)	Зменшений розмір (ГБ)	Потенційна економія (ГБ)	Заощадження, %
t1.micro	0.61	0.501	0.109	17.87
c1.medium	1.7	0.474	1.226	72.12
m1.small	1.7	0.474	1.226	72.12
c3.large	3.75	0.496	3.254	86.77
m1.medium	3.75	0.496	3.254	86.77
m3.medium	3.75	0.496	3.254	86.77
c1.xlarge	7	0.515	6.485	92.64
m1.large	7.5	0.511	6.989	93.19
m1.xlarge	15	0.527	14.473	96.49
m2.xlarge	17.1	0.56	16.54	96.73
m2.2xlarge	34.2	0.598	33.602	98.25

Існують сценарії, в яких SEaMLESS не може працювати через технічні обмеження в базових технологіях (CRIU та ptrace). CRIU версії 3.5 не підтримує повторне підключення потокових сокетів Unix. Дійсно, встановлення контрольних точок і завершення процесу призводить до того, що одноранговий потоковий сокет Unix закриває з'єднання. Втім, CRIU постійно перебуває у розробці зі стабільним щомісячним циклом випусків. Метою розробника є підтримка всього масиву IPC

та інших файлових дескрипторів для повного рішення для контрольних точок/відновлення процесів у Linux.

У пункті 3.1 було зазначено, що створення сервера-відновлювача з того самого шаблону, що й непрацююча ВМ, дозволяє уникнути перенесення бібліотек, завантажених процесом шлюзу. Будь-яка зміна оригінального шаблону може призвести до збоїв при відновленні. Однак різні версії легко виявити, вимкнувши SEaMLESS у таких випадках.

Оскільки SEaMLESS пропонує два взаємодоповнюючих рішення - одне для створення легкого контейнера з процесом шлюзу, а інше для вивільнення пам'яті з неактивної ВМ - може виникнути питання, чому неактивна ВМ не просто стискається, а залишає процес шлюзу на ВМ, не призупиняючи його. Відокремлення процесу шлюзу (який завжди має бути запущено) від ВМ (яку можна призупинити) дає кілька переваг. По-перше, базовий набір пам'яті простоюючої ВМ, на якій запущено шлюзовий процес, повільно зростає з часом і навіть може показувати раптове збільшення пам'яті через деякі сервіси, такі як автоматичне оновлення додатків. SEaMLESS переносить будь-яку нову активність користувача, уникаючи автоматичної реакції гіпервізора, яка може призвести до надмірної заміни пам'яті, що вплине на всі робочі навантаження, розміщені на непрацюючих ВМ, що відновлюють роботу. SEaMLESS може виконувати міграцію в реальному часі, щоб зменшити потенційні "гарячі точки" після відновлення роботи неактивних віртуальних машин одночасно.

SEaMLESS підходить для обраної групи робочих навантажень. Монолітні сервіси, які використовують спільну пам'ять для обробки запитів користувачів у додатку, не підтримуються. SEaMLESS повинен мати можливість ізолювати процеси шлюзу, зазвичай легкі, від громіздких робочих процесів, які використовують ресурси віртуальної машини (процесор і пам'ять) для виконання запиту. Якщо процеси шлюзу та робочі процеси прозоро взаємодіють через спільну пам'ять, SEaMLESS не може ізолювати такі взаємодії.

У цьому розділі було представлено SEaMLESS, фреймворк для трансформації та заміни неактивних віртуальних машин на легкі віртуальні

мережеві функції, що дозволяє відключати такі віртуальні машини кількома способами. Щоб досягти такої заміни, SEaMLESS ідентифікує на рівні процесів ту частину стану ВМ, яка є інтерфейсом між зовнішнім світом та ВМ: процеси-шлюзи. За допомогою міграції процесів з урахуванням старіння (CRIU) у поєднанні з пісочницею процесів (ptrace), SEaMLESS уможливлює виконання процесів шлюзу вхолосту, пересаджених за межі оригінальної ВМ і розміщених у контейнері: VNF. Відстежуючи системні виклики, за допомогою яких процеси шлюзу взаємодіють з контейнером, SEaMLESS здатна виявляти нетривіальні запити кінцевих користувачів, які вимагають відновлення роботи оригінальної ВМ. В рамках SEaMLESS було розроблено нову технологію, яка називається suspend-to-swap, для відключення ВМ і повернення виділеної пам'яті, зберігаючи при цьому швидкий час відновлення. Suspend-to-swap - це комбінація свопінгу гіпервізора та балонування, яка дозволяє проактивно виселяти пам'ять віртуальної машини у сховище, звільняючи дорогоцінний ресурс для інших цілей (наприклад, для подальшої консолідації).

Представлені в роботі експерименти демонструють, що SEaMLESS незначно впливає на якість роботи завдяки лінивому відновленню, яке забезпечує suspension-to-swap, повертаючи при цьому більшу частину пам'яті, виділеної непрацюючому екземпляру. Сервіси, розгорнуті на вимкнених ВМ за допомогою SEaMLESS, показують час відгуку близько 1 секунди для Apache 2 і близько 0,5 секунди для OpenSSH; обидва значення включають в себе затримку відновлення ВМ і будь-яку процедуру відновлення сервісу. Що ще важливіше, SEaMLESS використовує легкодоступні функції основних гіпервізорів та операційних систем (підкачка, балонування, пісочниця трасування, контейнери), що вимагає мінімальних зусиль для впровадження та підтримки.

ВИСНОВКИ

Віртуалізація та консолідація серверів є основою для зменшення операційних витрат центрів обробки даних. Міграція використовується для зміни конфігурації розміщення віртуальних екземплярів (динамічна повторна консолідація), основною метою якої є перегрупування рідко використовуваних ресурсів і, зрештою, розгортання додаткових екземплярів або вимкнення живлення порожніх фізичних машин. Крім того, заходи з технічного обслуговування сприяють міграції, щоб уникнути перебоїв у роботі, пов'язаних з вимкненням серверів, мережі, електроживлення та обладнання для охолодження. Однак міграція не є панацеєю. Для віртуальних машин, що простоюють, динамічна повторна консолідація в поєднанні з традиційними механізмами надмірного резервування або не забезпечує достатньої кількості ресурсів (наприклад, через роздування або дедуплікацію сторінок), або може призвести до непередбачуваного падіння продуктивності (відомий недолік технології заміни гіпервізорів). Що стосується оновлення гіпервізорів, то переваги обмеженого часу простою, отримані завдяки міграції в реальному часі, стикаються з поганою масштабованістю такої стратегії, оскільки міграція в реальному часі є ресурсоємним процесом як з точки зору використання мережі, так і з точки зору доступності вільних ресурсів. Таким чином, оновлення гіпервізора на основі міграції в реальному часі потенційно призводить до тривалих кампаній оновлення, що подовжує вразливість до фатальних вразливостей.

У цій роботі було використано універсальність трансплантації стану виконання віртуальних екземплярів, зокрема віртуальних машин. За допомогою SEaMLESS частина стану віртуальної машини ізолюється та витягується, щоб ввести в оману кінцевих користувачів (або агентів моніторингу), змусивши їх повірити, що віртуальна машина все ще працює, тоді як активною залишається лише інтерфейсна частина сервісів. Гіпервізор може повернути собі виділені (заблоковані) ресурси, відключивши ВМ, що простоює.

У цій кваліфікаційній роботі було представлено SEaMLESS - рішення для підвищення рівня консолідації в дата-центрах з великою кількістю віртуальних машин, що простоюють без діла. Незважаючи на те, що віртуальні машини простоюють, вони зберігають свою пам'ять. Таким чином, оперативна пам'ять стає вузьким місцем при розгортанні більшої кількості екземплярів на одній фізичній машині. Просте завершення роботи неактивних віртуальних машин, щоб звільнити їхню пам'ять, підриває доступність (а в деяких випадках і порушує SLA), яку оператори дата-центрів повинні гарантувати. Будь-яка стратегія, спрямована на вирішення цієї проблеми, повинна враховувати раптові запити кінцевих користувачів, які припиняють бездіяльність простоюючих віртуальних машин, обробляючи їх з найкоротшою затримкою.

Різноманітні сучасні рішення пропонують механізм вивільнення пам'яті з непрацюючих віртуальних машин. За допомогою часткової міграції ВМ, невелика частина стану простоюючих ВМ, яка робить доступними їх вбудовані сервіси, тобто робочий набір, може бути консолідована на декількох фізичних машинах, призупиняючи роботу решти хостів, які тримають основну частину стану. Визначення робочої множини для виконання вхолосту, необхідне для визначення частини стану ВМ, яку потрібно підтримувати доступною, може, однак, не враховувати повідомлення про підтримку працездатності або інші тривіальні запити. Тому отримання такого класу запитів може не обов'язково призвести до відновлення ВМ. Проксі-сервери, що замінюють вимкнені неактивні ВМ, дозволяють краще контролювати, які запити кінцевих користувачів вимагають відновлення автономних екземплярів. Однак проблема полягає в тому, щоб реалізувати універсальний проксі, який підтримує будь-який мережевий протокол (наприклад, TCP, UDP тощо), а також програми, які працюють на неактивних ВМ (веб-сервер, SSH-сервер тощо). Саме тому було запропоновано SEaMLESS - фреймворк, здатний перенести процеси шлюзу, що є точкою входу до сервісів ВМ, до контейнера. Процеси шлюзу, пересаджені всередину контейнера, діють як проксі-замінник відключеної простоюючої ВМ. Таким чином, контейнер являє собою полегшену віртуальну мережеву функцію (VNF), яка підтримує мережеву

присутність сервісів. Активність користувача виявляється в контейнері шляхом моніторингу взаємодії між пересадженими процесами шлюзу та їх контейнером, що дозволяє успішно ідентифікувати запити, які вимагають відновлення роботи ВМ.

SEaMLESS дозволяє використовувати різні методи для відключення віртуальних машин (процеси шлюзу яких було перенесено) та повернення їх ресурсів, особливо пам'яті. Suspend-to-disk повністю повертає всю пам'ять, але відновлення ВМ відбувається повільно, пропорційно до розміру гостьової оперативної пам'яті.

Призупинені до оперативної пам'яті віртуальні машини миттєво відновлюються, але пам'ять не повертається. В даній роботі було запропоновано комбінацію найкращого з обох світів - suspend-to-swap, яка витісняє більшу частину пам'яті віртуальних машин до сховища і миттєво перезапускає їх. Suspend-to-swap використовує свопінг гіпервізора, оперативно переміщуючи пам'ять віртуальної машини в своп, щоб звільнити місце для інших екземплярів, використовуючи вибірку сторінок пам'яті в міру їх необхідності.

SEaMLESS є вдосконаленим способом переповнення пам'яті хоста. Як наслідок, перспективною подальшою роботою буде вивчення інтеграції SEaMLESS з таким менеджером платформи, як OpenStack. SEaMLESS здатна оперативно виявляти будь-яку нову активність користувача, отже, оператори дата-центру можуть виконати найбільш підходящу операцію для відновлення роботи ВМ. При наявності "гарячої точки" пам'яті, тобто групи ВМ, які одночасно виходять з простою, ВМ не може відновитися на своєму хості через ризик конфлікту пам'яті та неконтрольованого обміну. Міграція віртуальних машин у реальному часі корисна для вирішення ситуацій переповненості, однак вибір того, яку віртуальну машину мігрувати і як саме, пов'язаний з певними проблемами. Наприклад, міграція з попереднім копіюванням не відразу звільняє пам'ять, виділену віртуальною машиною, тоді як міграція з подальшим копіюванням поступово повертає пам'ять по мірі надсилання сторінок до місця призначення. Найбільш відповідний вибір залежить від наявних ресурсів і типу сервісів, розміщених на

задіяних ВМ. Це вимагає глибокої інтеграції менеджера ресурсів (Openstack або незалежний від платформи планувальник віртуальних машин, такий як BtrPlace).

ПЕРЕЛІК ПОСИЛАНЬ

1. Arnold, J., and Kaashoek, M. F. Ksplice: Automatic Rebootless Kernel Updates. In Proceedings of the 4th ACM European Conference on Computer Systems (2019), EuroSys ’19, pp. 187–198.

2. Anthony Liguori. Powering Next-Gen EC2 Instances: Deep Dive into the Nitro System. https://www.youtube.com/watch?v=e8DVmwj3OEs, November 2018

3. Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. Benchmarking Cloud Serving Systems with YCSB. In Proceedings of the 1st ACM Symposium on Cloud Computing (2020), SoCC ’20, pp. 143–154.

4. Cortez, E., Bonde, A., Muzio, A., Russinovich, M., Fontoura, M., and Bianchini, R. Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms. In Proceedings of the 26th Symposium on Operating Systems Principles (2017), SOSP ’17, pp. 153–167.

5. Google Compute Engine, Nested virtualization overview. https://cloud.google.com/compute/docs/instances/nestedvirtualization/overview.

6. Google Compute Engine, Preemptible VM Instances. https://cloud. google.com/compute/docs/instances/preemptible.

7. Intel Processor Microcode Package for Linux. https://github.com/intel/Intel-Linux-Processor-Microcode-Data-Files.

8. Intel Software Guard Extensions (Intel R SGX). https://www.intel.com/content/www/us/en/architecture-andtechnology/software-guard-extensions.html.

9. Kpatch: Dynamic Kernel Patching, GitHub Repository. https://github.com/dynup/kpatch.

10. KVM, CPU Hotplug. https://www.linux-kvm.org/page/CPUHotPlug.

11. Open vSwitch. http://openvswitch.org/

12. Open vSwitch, Bonding. https://docs.openvswitch.org/en/latest/topics/bonding/.

13. Openstack web page. https://www.openstack.org/.

14. Physical Memory Model. https://www.kernel.org/doc/html/latest/vm/memory-model.html.

15. QEMU Migration Documentation. https://github.com/qemu/qemu/blob/master/docs/devel/migration.rst.

16. Redis, Web Page. https://redis.io/.

17. SUSE Linux Enterprise Live Patching. https://www.suse.com/products/live-patching/.

18. The Linux x86 Boot Protocol. https://www.kernel.org/doc/Documentation/x86/boot.txt.

19. Virtuozzo Hybrid Server. https://www.virtuozzo.com/virtuozzohybrid-server/.

20. VMware vSphere, Enable CPU Hot Add . https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.vm_admin.doc/GUID-285BB774-CE69-4477-9011-598FEF1E9ACB.html.

21. Xen Project Software Overview. https://wiki.xenproject.org/wiki/Xen_Project_Software_Overview.

22. Hyper-V Architecture. https://docs.microsoft.com/enus/windows-server/administration/performance-tuning/role/hyper-v-server/architecture

23. Payment Card Industry Data Security Standard, Requirements and Security Assessment Procedures Version 3.2.1. https://www.pcisecuritystandards.org/

24. CRIU, Upstream Kernel Commits. https://criu.org/Upstream_kernel_commits

25. Microsoft Azure, Hypervisor Security on the Azure Fleet. https://docs.microsoft.com/en-us/azure/security/fundamentals/Hypervisor.

ДЕМОНСТРАЦІЙНІ МАТЕРІЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІКЦІЙНА РОБОТА

«ДОСЛІДЖЕННЯ МЕТОДІВ ОПТИМІЗАЦІЇ ВИКОРИСТАННЯ РЕСУРСІВ У МЕРЕЖАХ ЦЕНТРІВ ОБРОБКИ ДАНИХ (DATA CENTERS) З ВИКОРИСТАННЯМ ВІРТУАЛІЗАЦІЇ ТА КОНТЕЙНЕРИЗАЦІЇ»

виконав студент: **Голосун А. І.**
керівник: **Лащевська Н.О.**, к.т.н., доцент

Мета магістерської роботи:

Забезпечення ефективного використання ресурсів та підвищення продуктивності мереж центрів обробки даних

Об'єкт дослідження:

Методи оптимізації використання ресурсів у мережах центрів обробки даних

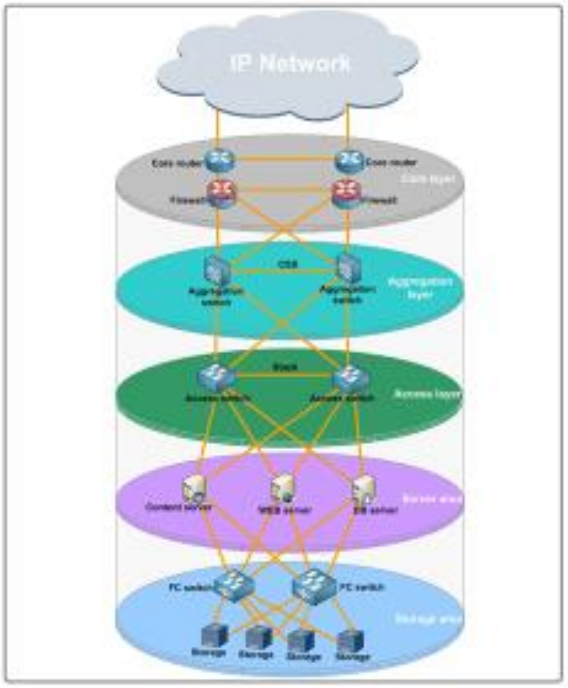
Предмет дослідження:

Мережі центрів обробки даних

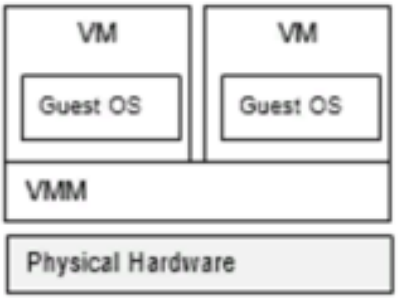
Актуальність:

Дослідження методів оптимізації використання ресурсів у мережах центрів обробки даних є актуальним, оскільки зростає значення цих мереж у сучасному інформаційному суспільстві. Використання віртуалізації та контейнеризації може допомогти ефективніше використовувати ресурси та забезпечити високу продуктивність мереж центрів обробки даних.

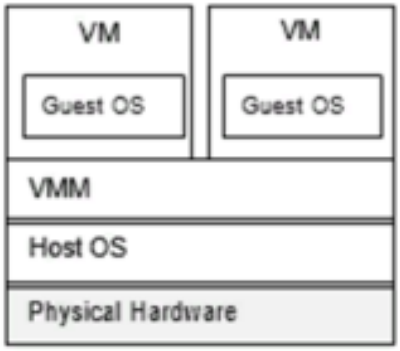
Інфраструктура ЦОД



Архітектури гіпервізорів типу 1 та типу 2

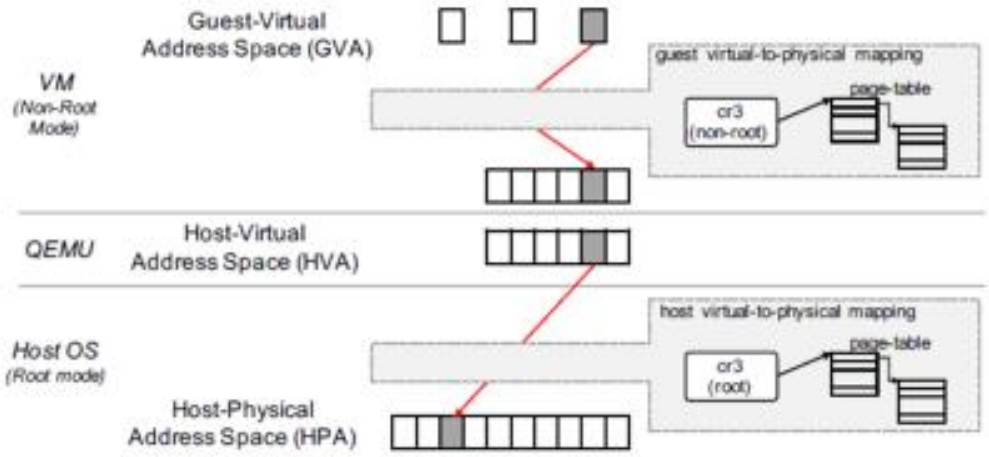


Гіпервізор типу 1



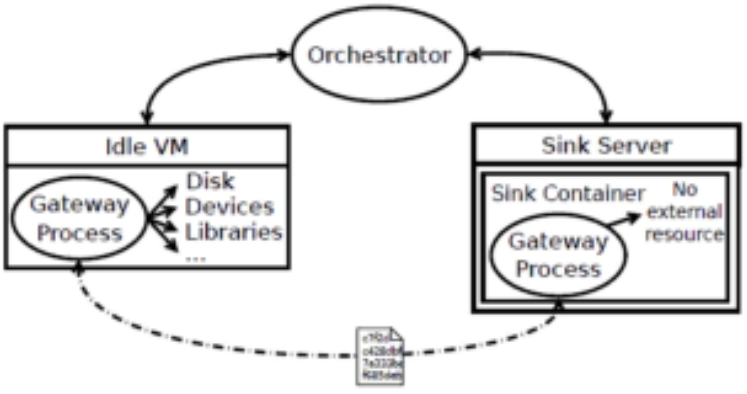
Гіпервізор типу 2

Взаємозв'язок між двовимірними адресними просторами в апаратній віртуалізації



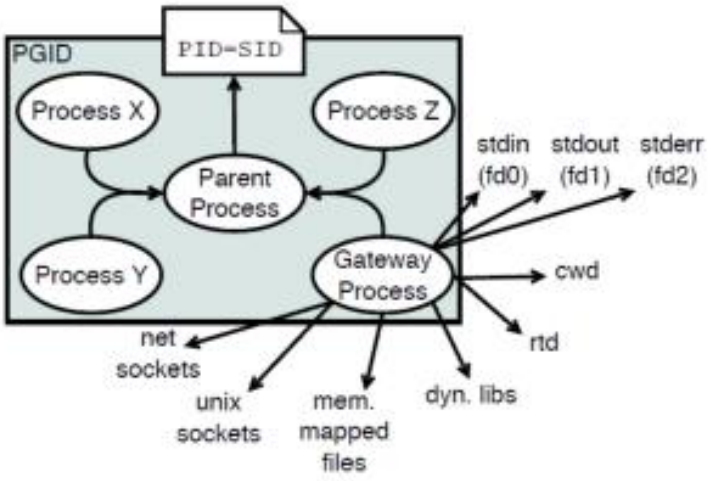
6

Компоненти та архітектура SEaMLESS



7

Можлива екосистема процесу шлюзу та її ресурси

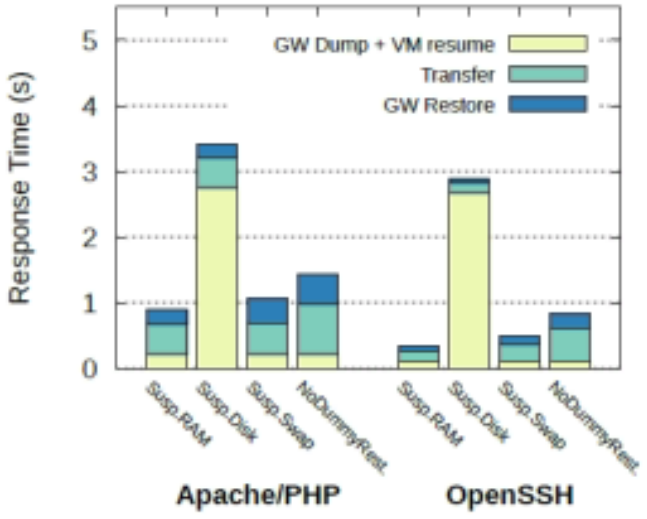


Час відгуку реальних додатків шлюзових процесів

Додаток	Основні завдання Час (с)			Всього (с)	Час реакції (с)	Зображення Розмір (МБ)	VNF Розмір (МБ)
	Дамп	Переказ	Відновлення				
Dropbear	0.04	0.12	0.04	0.20	0.41	0.12	11.18
Vsftpd	0.12	0.11	0.05	0.28	0.41	0.11	7.81
OpenSSH	0.13	0.13	0.07	0.33	0.46	0.13	15.93
Lighttpd/PHP	0.16	0.29	0.16	0.61	0.71	0.29	46.43
Apache2/PHP	0.23	0.43	0.24	0.90	0.95	0.43	67.52
Tomcat	0.46	1.17	0.38	2.02	2.16	1.17	206.96

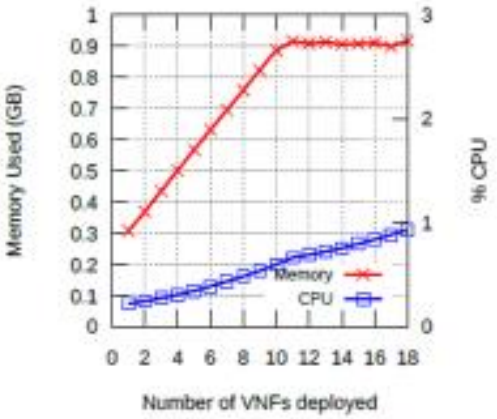
9

Час відгуку при використанні різних методів відключення на віртуальній машині об'ємом 3 ГБ

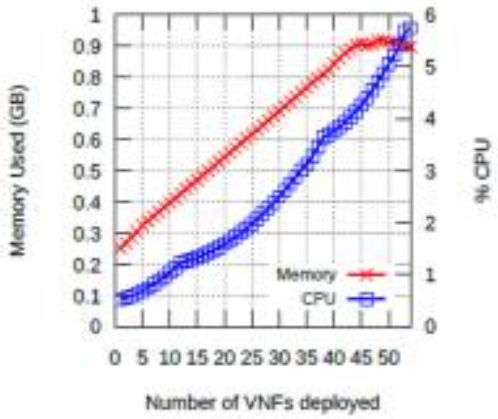


10

Використання оперативної пам'яті та центрального процесора в залежності від кількості розгорнутих VNF

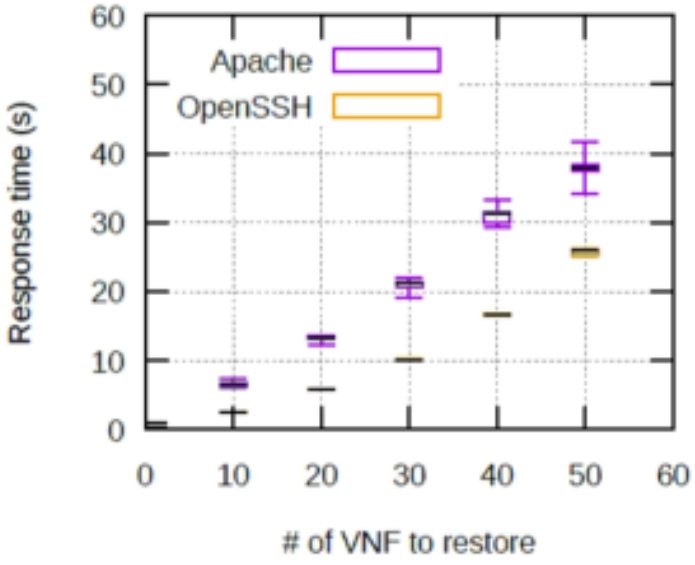


(a) Apache 2 w/ PHP.



(b) OpenSSH w/ 1 SSH сxп.

Час відгуку в залежності від кількості паралельних OpenSSH VNF з активністю користувачів



Потенційна економія пам'яті для поширених типів екземплярів AWS, за умови розміру VNF 67 МБ

Тип AWS	Розмір (ГБ)	Зменшений розмір (ГБ)	Потенційна економія (ГБ)	Заощадження, %
t1.micro	0.61	0.501	0.109	17.87
c1.medium	1.7	0.474	1.226	72.12
m1.small	1.7	0.474	1.226	72.12
c3.large	3.75	0.496	3.254	86.77
m1.medium	3.75	0.496	3.254	86.77
m3.medium	3.75	0.496	3.254	86.77
c1.xlarge	7	0.515	6.485	92.64
m1.large	7.5	0.511	6.989	93.19
m1.xlarge	15	0.527	14.473	96.49
m2.xlarge	17.1	0.56	16.54	96.73
m2.2xlarge	34.2	0.598	33.602	98.25

Висновки

Основний внесок роботи полягає в розробці та впровадженні SEaMLESS. SEaMLESS реалізує нову процедуру перетворення непрацюючої ВМ (з Linux як гостьовою ОС) на полегшену VNF. SEaMLESS спирається на нову техніку відключення ВМ, яка називається suspend-to-swap, розроблену для гіпервізора на основі KVM, який використовує свопінг Linux. Крім того, SEaMLESS можна комбінувати зі старими методами відключення віртуальних машин, такими як suspend-to-RAM або suspend-to-disk, щоб повернути ресурси, заблоковані простоюючими екземплярами. Оцінка SEaMLESS показала:

- після виявлення активності користувача відновлюється початкове середовище ВМ, і сервіси продовжують виконувати свою роботу прозоро з мінімальним впливом на якість роботи;
- можна замінити сотні непрацюючих віртуальних машин відповідними VNF, консолідованими на одному фізичному сервері або віртуальній машині;
- Suspend-to-swap легко відновлює роботу зупиненої ВМ, а також повертає більшу частину пам'яті ВМ, що простоює.

Представлені в роботі експерименти демонструють, що SEaMLESS незначно впливає на якість роботи завдяки лінивому відновленню, яке забезпечує suspension-to-swap, повертаючи при цьому більшу частину пам'яті, виділеної непрацюючому екземпляру. Сервіси, розгорнуті на вимкнених ВМ за допомогою SEaMLESS, показують час відгуку близько 1 секунди для Apache 2 і близько 0,5 секунди для OpenSSH; обидва значення включають в себе затримку відновлення ВМ і будь-яку процедуру відновлення сервісу. Що ще важливіше, SEaMLESS використовує легкодоступні функції основних гіпервізорів та операційних систем (підкачка, балонування, пісочниця трасування, контейнери), що вимагає мінімальних зусиль для впровадження та підтримки.

ДЯКУЮ ЗА УВАГУ!