



Московский государственный университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра автоматизации систем вычислительных комплексов

Михеев Павел Алексеевич

**Исследование эффективности использования
физических ресурсов
легковесными контейнерами в облачных системах.**

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Научный руководитель:
к.ф.-м.н.
В.А. Антоненко

Москва, 2018

Аннотация

Контейнер - изолированная группа процессов в операционной системе, имеющая доступ к ограниченному количеству ресурсов системы. Данная изоляция достигается за счет возможностей ядра операционной системы, таким образом, контейнеры используют ядро хостовой системы, не тратя ресурсы на поддержку гостевого ядра.

Контейнеры обладают небольшим временем запуска, большой пропускной способностью, и производительность вычислений в них ближе к производительности вычислений на физических ресурсах. Однако не существует систем, позволяющих исследовать производительность и деградацию производительности контейнеров при масштабировании и сравнить ее с производительностью и деградацией для виртуальных машин.

В данной работе предлагается использовать существующие методики измерения производительности виртуальных машин и применить их к контейнерам.

Целью работы является разработка системы измерения производительности и ее деградации у виртуальных машин и контейнеров, запущенных на физическом сервере.

Оглавление

| | |
|---|-----------|
| Аннотация | 1 |
| Введение | 4 |
| 1. Постановка задачи | 7 |
| 1.1. Цель работы | 7 |
| 1.2. Задачи | 7 |
| 2. Технологии виртуализации | 9 |
| 2.1. Полная виртуализация | 10 |
| 2.2. Паравиртуализация | 11 |
| 2.3. Аппаратная виртуализация | 11 |
| 2.4. Общие достоинства и недостатки | 12 |
| 2.5. Виртуализация на уровне операционной системы | 13 |
| 2.6. Контейнеризация приложений | 15 |
| 2.7. Выводы | 18 |
| 3. Обзор существующих решений | 20 |
| 3.1. Производительность виртуальной машины | 20 |
| 3.2. Производительность виртуальных машин и контейнеров . | 22 |
| 3.3. Выводы | 23 |
| 4. Практическая реализация | 25 |
| 4.1. Требования | 25 |
| 4.2. Алгоритм работы реализованной системы | 26 |

| | |
|--|-----------|
| 4.3. Применимость | 29 |
| 5. Экспериментальное исследование | 30 |
| 5.1. Экспериментальный стенд | 31 |
| 5.2. Подбор паролей | 32 |
| 5.3. Python-код | 37 |
| 5.4. Пропускная способность оперативной памяти | 40 |
| 5.5. Выводы | 43 |
| 6. Заключение | 46 |

Введение

Облачные вычисления в современной информационной среде получили широкое распространение. Развитие открытого программного обеспечения по управлению облаками увеличивает количество тех, кто использует облачные системы как внутри компании, так и для предоставления услуг клиентам. Одним из немногих сдерживающих факторов использования облачных вычислений является цена физического оборудования.

Традиционно облачные вычисления используют виртуальные машины как сущности, с помощью которых предоставляются услуги пользователю.

Количество виртуальных машин, которое может быть запущено в облаке, напрямую зависит от количества физических ресурсов. Под количеством физических ресурсов в данной работе будет пониматься совокупное (по всем серверам) количество ядер процессоров, количество оперативной памяти и объем жестких дисков системы.

Пусть X - количество доступных физических ресурсов в облаке, а χ_i - количество ресурсов, требующихся для i -ой виртуальной машины. Тогда n - максимальное число виртуальных машин с такими требованиями, которое может быть запущено на данных ресурсах, если $\sum_{i=1}^n \chi_i = X$.

Однако может возникать ситуация, когда количество запрашиваемых виртуальных машин превосходит максимальное число n , определенное выше. При этом у владельца облака нет возможности или необходимости увеличить количество физических ресурсов. Важным моментом является тот факт, что ресурсы виртуальной машины могут быть использованы ее процессами не на 100% процентов. При предоставлении доступа к этим же физическим ресурсам или их части той виртуальной машине, номер которой превосходит n , все виртуальные машины смогут осуществлять свои функции. В таком случае $\sum_{i=1}^n \chi_i > X$, но при этом все виртуальные машины размещены.

Введем коэффициент $\theta = \frac{\sum_i \chi_i}{X}$ - коэффициент перекрытия (overlap), являющийся отношением суммы ресурсов, требующихся виртуальным машинам системы, к физическим ресурсам системы.

При увеличении коэффициента перекрытия производительность процессов в виртуальных машинах падает, так как при разделении одних физических ресурсов (в первую очередь, ядер процессора) разными виртуальными машинами исполнение машин одновременно будет невозможно. Из-за этого будет наблюдаться деградация производительности виртуальных машин при увеличении коэффициента перекрытия.

В облачных системах в последние годы стала использоваться легковесная виртуализация. При данной технологии виртуализации гостевая операционная система не используется, а все процессы запускаются в ядре хостовой операционной системы. Изоляция процессов друг от друга и ограничение доступных им ресурсов достигается за счет специальных возможностей ядра. Такие процессы и их потомки с наложенными на них ограничениями называются контейнерами. Так же, как процессы в одной виртуальной машине изолированы от процессов другой виртуальной машины, процессы в одном контейнере изолированы от процессов других контейнеров. Отличие заключается в том, что системные вызовы в виртуальной машине идут в операционную систему машины, тогда как системные вызовы контейнеров идут напрямую в ядро операционной системы, в которой данный контейнер запущен.

Контейнеры могут быть использованы в облачных системах так же, как и виртуальные машины. Они потребляют некоторое количество ресурсов и могут разделять их между собой, а при увеличении коэффициента перекрытия будет наблюдаться деградация производительности.

В данной работе предлагается исследовать и сравнить производительность виртуальных машин и контейнеров, а так же деградацию производительности машин и контейнеров при увеличении коэффициента

перекрытия. Для этого была разработана система, способная запускать виртуальные машины или контейнеры и запускать в них приложения, позволяющие измерять производительность, а так же собирать результаты работы этих приложений.

Основной гипотезой данной работы является следующее: контейнеры обладают более высокой производительностью по сравнению с виртуальными машинами, деградация производительности виртуальных машин при увеличении коэффициента перекрытия происходит быстрее, чем у контейнеров.

Глава 1.

Постановка задачи

1.1. Цель работы

Разработать и реализовать систему, позволяющую сравнить производительность виртуальных машин и контейнеров, а так же деградацию производительности виртуальных машин и контейнеров при увеличении коэффициента перекрытия.

С помощью разработанной системы провести эксперименты, позволяющие проверить следующую гипотезу: контейнеры обладают более высокой производительностью по сравнению с виртуальными машинами, деградация производительности виртуальных машин при увеличении коэффициента перекрытия происходит быстрее, чем у контейнеров.

1.2. Задачи

1. Сравнить различные технологии виртуализации.
2. Составить обзор существующих методик измерения производительности виртуальных машин и контейнеров. По результатам выбрать методику оценки производительности или разработать свою, которую возможно применить для оценки производительности виртуаль-

ных машин и контейнеров и их сравнения.

3. Разработать систему, реализующую методику оценки производительности из предыдущего пункта, позволяющий численно оценить производительность и ее деградацию при увеличении коэффициента перекрытия в случае виртуальных машин и контейнеров.
4. С помощью разработанной системы провести эксперименты, позволяющие проверить следующую гипотезу: контейнеры обладают более высокой производительностью по сравнению с виртуальными машинами, деградация производительности виртуальных машин при увеличении коэффициента перекрытия происходит быстрее, чем у контейнеров.

Глава 2.

Технологии виртуализации

В данном разделе под хостовой операционной системой будет пониматься система, в которой могут быть запущены гостевые операционные системы. Гостевые операционные системы - это те системы, которые видят свое изолированное окружение, и которые не осведомлены о наличии других гостевых систем. Под виртуальной сущностью будет пониматься тот процесс в хостовой операционной системе, который исполняет вычисления гостевой операционной системы.

Гипервизор - специализированное программное обеспечение, которое занимается управлением гостевыми операционными системами: запуском, остановкой, наблюдением и выделением ресурсов. Гипервизоры бывают двух типов:

Виртуализация - подход к организации вычислений, при котором каждая виртуальная сущность изолирована от других, причем ей может быть доступна лишь часть общих ресурсов. В данном разделе будет рассматриваться виртуализация центрального процессора, то есть каким образом возможно исполнять гостевую операционную систему изолированно на центральном процессоре хостовой системы.

Существует четыре основных вида виртуализации: полная виртуализация, паравиртуализация, аппаратная виртуализация и легковесная

виртуализация. Далее будут рассмотрены каждый из этих видов, а также их достоинства, недостатки и применимость в облачных системах.

2.1. Полная виртуализация

Основная особенность данного вида виртуализации заключается в том, что гостевая операционная система полностью отделяется от управления инфраструктурой хоста. Гостевая ОС не требует никаких изменений.

Как следует из названия, данный вид виртуализации позволяет запустить любую гостевую операционную систему в любой хостовой. Единственное требование - это наличие динамического транслятора из машинного языка архитектуры, с которой работает гостевая операционная система, в машинный язык хостовой архитектуры [1].

Преимуществом данного вида виртуализации является гипотетическая возможность запускать любую гостевую операционную систему.

При этом основное преимущество оборачивается и основным недостатком. При современном разнообразии вычислительной техники невозможно иметь трансляторы с любого машинного языка в любой. Но даже при наличии транслятора возможно, что скорость исполнения гостевого кода будет гораздо медленнее, чем в случае исполнения на настоящей архитектуре. Дополнительно к этим недостаткам добавляется сложность реализации динамической трансляции, связанной, к примеру, с неразличимостью команд и данных [1].

При этом существует ряд гипервизоров, поддерживающих полную виртуализацию. Примерами могут быть VMware ESXi, Microsoft Virtual Server [1]. Стоит отметить, что в силу закрытости данного программного обеспечения, их исследование в данной работе произведено не было.

2.2. Паравиртуализация

Проблемой динамической трансляции является исполнение привилегированных инструкций гостевой операционной системы. При обработке данных инструкций работа гостевой операционной системы ухудшается.

Паравиртуализация пытается исправить данный недостаток. Это подход подразумевает модификацию гостевой операционной системы таким образом, чтобы исполнение привилегированных инструкций было оформлено как системный вызов в хостовую операционную систему. При этом при использовании паравиртуализации производительность гостевой операционной системы выше, чем при использовании полной виртуализации [3].

Основным же недостатком данного вида виртуализации является необходимость модификации ядра гостевой системы. Это уменьшает количество операционных систем, которые могут быть виртуализированы с помощью данного подхода.

Наиболее популярные гипервизоры, поддерживающие паравиртуализацию: Xen [4] и VMware [1]. Однако в данной работе они не будут рассматриваться. Это связано с тем, что технология Xen не используется в облачных системах. Продукты же VMware являются закрытыми, что так же не позволяет их рассматривать в данной работе.

2.3. Аппаратная виртуализация

Подход аппаратной виртуализации подразумевает совпадение машинного языка гостевой системы и хостовой. При этом архитектура процессора хостовой операционной системы должна поддерживать аппаратную виртуализацию, то есть исполнять код гостевой системы так, будто бы это код хостовой. Привилегированные вызовы автоматически отслеживаются гипервизором и при обнаружении исполняются на процессоре напря-

мую [1].

Такой подход позволяет решить проблему необходимости модификации ядра для повышения производительности гостевой системы, а также использования полной виртуализации, несмотря на требование специфической архитектуры процессора хостовой системы. Поддержка данной технологии есть почти во всех современных процессорах, что позволяет использовать данный вид виртуализации в облачных системах.

Аппаратная виртуализация поддерживается множеством гипервизоров. Основные представители - это VirtualBox [5], KVM [6], VMware ESXi [2]. В данной работе будет рассматривать гипервизор KVM, который является модулем ядра Linux.

2.4. Общие достоинства и недостатки



Рис. 2.1. Схема аппаратной виртуализации

Рассмотренные выше технологии виртуализации имеют ряд общих достоинств и недостатков. С одной стороны, поддержка гостевого ядра расширяет набор возможных операционных систем, которые можно запустить. С другой стороны, поддержка гостевого ядра требует ресурсов: гостевая операционная система будет потреблять часть выделенной оперативной памяти, часть выделенного дискового пространства и часть ресурсов процессора. При запуске виртуальной машины будет требоваться время на запуск ядра ОС. Оперативная память, выделяемая гипервизором под виртуальную машину, чаще выделяется непрерывным участком,

что может приводить к внутренней фрагментации памяти.

2.5. Виртуализация на уровне операционной системы

Основная идея данного вида виртуализации заключается в том, что операционная система способна выделять своим процессам (или группе процессов) и их потомкам некоторое подмножество своих ресурсов. Пусть так же ОС способна выделять для каждого такого подмножества ресурсов свои множества идентификаторов процессов. Тогда для каждого подмножества ресурсов возможно запускать процессы, которые в общем случае могут быть запущены только в единичном экземпляре (в первую очередь, это процесс `init` ядра Linux). Таким образом можно изолировать подгруппы физических ресурсов, имея возможность запустить на каждой подгруппе свой экземпляр операционной системы. Однако стоит отметить, что ядро этих ОС общее, а именно ядро хостовой операционной системы.

Другим названием для этого подхода служит контейнеризация, легковесная виртуализация или контейнерная виртуализация, а каждую подгруппу ресурсов со своим множеством процессов называют контейнер.

Изначально этот подход был реализован в ядре Linux. Модуль `control groups` (`cgroups`) ядра позволял выделять подгруппы ресурсов, а модуль `namespaces` - выделять свои иерархии идентификаторов процессов. На базе этого подхода реализовано несколько видов контейнеров - в первую очередь `Linux Containers`(далее `LXC`).

Выделим необходимые множества ресурсов ОС, требующих выделения подгрупп:

- Точка монтирования: разные контейнеры должны монтироваться в

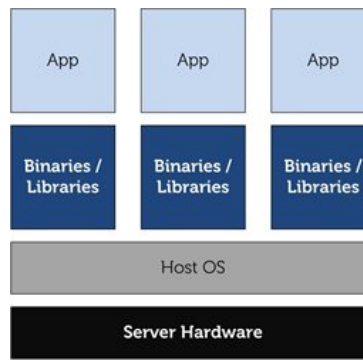


Рис. 2.2. Схема виртуализации на уровне ОС
labelконтейнеры

разные точки файловой системы хостовой ОС. Иначе действия контейнеров над файлами не будут синхронизированы.

- Network namespace: это необходимо для обеспечения полной изоляции на уровне сокетов, IP адресов и портов от соседа. Таким образом каждый контейнер имеет строго зафиксированный отдельный IP адрес, а также пространство сокетов, портов и роутинг-таблицу.
- IPC namespace: изолирует семафоры, очереди, мьютексы, shm память для IPC и IPC Sys V в таком виде, что каждый контейнер видит только свои IPC ресурсы и ничьи более.
- PID namespace: изолирует идентификаторы процессов в разных контейнерах друг от друга и от ОС, в которой запущен этот контейнер.
- UTS: позволяет выдавать уникальные hostname для каждого контейнера так, чтобы они не совпадали друг с другом и именем ОС, в которой контейнеры запущены.

Так же между контейнерами надо разделять следующие физические ресурсы: процессор, память и жесткий диск. Для реализации разграничения использования ресурсов используются следующие *cgroups*: *cpu*, *memory* и *blkio*. Например, с помощью *cpu* можно ограничить число ядер, которые используются в этой *cgroup*. Стоит отметить, что при контейнерной виртуализации для каждого контейнера используется отдельный

набор пространств имен (namespace) и отдельный набор *cgroups*, то есть разделение их между контейнерами не используется.

Типичными представителями контейнеров легковесной виртуализации в ОС Linux является Linux Containers [7] и OpenVZ [8]. Однако, в отличие от LXC, OpenVZ использует свой набор утилит для контейнеризации, пусть общая идея такая же, как описано выше. Чтобы работать с OpenVZ, необходима специальная настройка ядра под эти утилиты [8]. В то время как LXC работает в рамках стандартного ядра Linux.

Важной особенностью является тот факт, что все эти хосты работают внутри одной ОС, то есть все контейнеры, запущенные в рамках одной ОС, используют ядро ОС как свое ядро. Таким образом, нет возможности изменить ядро ОС для одного контейнера, не затронув остальные.

2.6. Контейнеризация приложений

Рассмотрим ситуацию, в которой пользователь хочет запустить новое приложение на своей машине. Однако скачивание, установка и настройка системы для работы с этим приложением занимает много времени и усилий, а некоторые дополнительные пакеты имеют несовместимости с существующими на машине пользователя. Возникает необходимость создания изолированного окружения для приложения, которое содержит все необходимые зависимости и настройки, при этом это окружение недоступно приложениям извне.

Контейнеризация приложения - это упаковка приложения и всех его зависимостей в легковесный контейнер и последующая настройка этого контейнера. Примером может послужить перенос ftp-сервера с машины на машину. После установки в LXC-контейнер самого сервера, необходимо настроить проброс 21 порта контейнера на 21 порт машины пользователя. Таким образом, когда пользователь скачает такой LXC-контейнер

и запустит его, то сразу получит работающий FTP-сервер, слушающий 21 порт.

Следующим шагом служит автоматизация работы с LXC-контейнером, то есть необходимые действия по упаковке и запуску осуществляет не пользователь, а специализированное программное обеспечение.

Docker [9] - средство для контейнеризации приложения со всеми его зависимостями, обеспечивающее API высокого уровня к LXC-контейнерам. Приложение упаковывается в Docker-контейнер, файловая система которого может переноситься с машины на машину.

Docker позволяет модифицировать Docker-контейнеры, добавляя в них новые приложения. При этом создается новый контейнер.

Последнее свойство нуждается в пояснении своей реализации. Если в LXC-контейнере используется обычная файловая система, то файловая система Docker-контейнера образована "слоями" из файловых систем - то есть в Docker-контейнере добавление файла в контейнер осуществляется путем записи нового слоя, который содержит файл, поверх старых слоев. В момент запуска Docker-контейнера старые слои интерпретируются как единая файловая система с помощью средства union-filesystem (разработка команды создателей Docker), причем эта единая система доступна только для чтения. Все изменения записываются поверх нее в новый слой (стоит отметить, что в документации Docker нет пояснений по поводу реализации этой системы). Такой подход позволяет использовать чужие Docker-контейнеры как основу для новых.

Таким образом, Docker - удобная система для управления приложениями. Так же как и LXC-контейнеры, Docker-контейнеры, запущенные в рамках одной ОС, обладают общим ядром. Это приводит к тем же ограничениям на запуск приложения в контейнере, что и для LXC-контейнеров.

За управление Docker-контейнерами отвечает Docker-daemon, процесс,

работающий в хостовой ОС. Он запускает необходимые контейнеры (то есть либо собирает новый LXC-контейнер, либо запускает копию уже существующего). Docker-daemon выдает IP-адрес контейнеру из диапазона, задаваемого IP-адресом и маской сети сетевого интерфейса, который указан в настройках Docker.

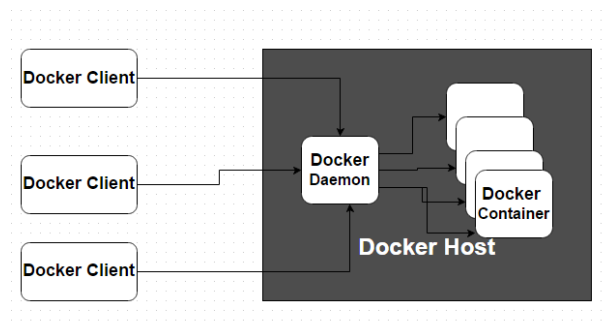


Рис. 2.3. Docker-демон

Сборка контейнера может производиться автоматически по Dockerfile - специальному файлу, в котором описана последовательность действий, необходимая для формирования нужного контейнера. Пользователь может сам написать подобный файл, а затем тот, кто получит такой файл, может сам собрать по нему нужный контейнер. При запуске Docker-контейнера возможно указать ограничения на размер доступной этому контейнеру оперативной памяти и число ядер процессора, на которых можно исполнять приложения этого контейнера.

Стоит отметить, что Docker не единственное средство для контейнеризации приложения. Существует так же Rocket для Linux [10] и Drawbridge для Windows [11]. Однако наиболее развитым и активно поддерживаемым со стороны общественности является Docker.

Начиная с 2016 года, был разработан runC - описание поведения системы, производящей управление контейнерами. Хотя официально runC не является стандартом, но позволяет унифицировать описание работы контейнера, абстрагируясь от конкретных сущностей, и этому описанию стараются следовать все разработчики контейнеров. Однако родоначаль-

ником runC является Docker, то есть Docker работает по классической схеме runC [12]. Таким образом де-факто Docker становится стандартом контейнерной виртуализации.

Начиная с 2016 года, разработчики Windows и Docker анонсировали внедрение Docker в ядро Windows. Достигнуто это будет за счет того, что была добавлена возможность запуска Ubuntu в Windows за счет трансляции системных вызовов ядра Linux в системные вызовы ядра Windows. Аналогичные разработки ведутся и под MacOS.

2.7. Выводы

Наиболее часто используемыми технологиями в сфере облачных вычислений являются легковесная и аппаратная виртуализации. Первая технология позволяет управлять контейнерами, а вторая - виртуальными машинами.

В рамках данной работы будут рассматриваться гипервизор KVM для управления виртуальными машинами и система Docker для управления контейнерами приложений.

В KVM существует понятие виртуального процессора (vCPU). Эта абстракция описывает ядро, доступное виртуальной машине. Обычно виртуальной машине выделяются не все ядра хостовой системы, а лишь часть. При этом по умолчанию в разные моменты времени виртуальная машина может использовать разные ядра хостовой системы. Это контролирует гипервизор. В KVM присутствует возможность закрепить конкретные ядра за конкретной машиной. [13]

В Docker система выделения ресурсов процессора контейнерам ближе к системе выделения ресурсов для процессов. При этом отсутствует возможность ограничивать число доступных контейнеру ядер, возможно лишь зафиксировать, какими ядрами будет пользоваться контейнер. [13]

В KVM выделение оперативной памяти виртуальной машине происходит непрерывным блоком. То есть, даже если машина пользуется не всей доступной памятью, нет возможности передать неиспользуемую память другой машине. [13]

Контейнеры обладают более гибкой системой управления оперативной памятью. Возможно задать лишь верхнюю границу выделяемой контейнеру памяти. При этом неиспользованная память может быть передана другому контейнеру. [13]

Файловая система виртуальной машины является отдельным файлом в хостовой операционной системе. Контейнеры же монтируют свою файловую систему в файловую систему хоста.

Контейнеры потребляют меньше ресурсов, чем виртуальные машины, так как нет необходимости поддерживать ядро гостевой ОС, и при этом позволяют запускать изолированные приложения. Запуск контейнеров происходит быстрее, чем запуск виртуальной машины. [14].

Глава 3.

Обзор существующих решений

В данном разделе будут рассмотрены существующие работы по исследованию производительности виртуальных машин и контейнеров. Будут выделены методики исследования.

3.1. Производительность виртуальной машины

Несмотря на то, что виртуальные машины, управляемые одним гипервизором, формально изолированы друг от друга, они используют одно и тоже физическое оборудование. При этом каждая виртуальная машина обладает своим планировщиком ресурсов, который никак не связан с планировщиками других машин, но при этом пытается управлять одним с ними физическим оборудованием. В случае паравиртуализации, при вызове нескольких системных вызовов от разных виртуальных машин в случае операций ввода-вывода обработка этих вызовов гипервизором так же может быть нетривиальной, а потому работа виртуальной машины будет отлична от работы той же системы на физическом оборудовании

без виртуализации.

В работе [15] исследовалось влияние двух Xen-виртуальных машин друг на друга. В них запускались специализированные приложения, измеряющие производительность вычислений на процессоре, пропускную способность оперативной памяти, производительность работы с жестким диском. Каждой машине было доступно по половине ресурсов хостовой системы, то есть они не конкурировали за ресурсы напрямую. При этом показатели производительности приложений были ниже, чем у аналогичной хостовой системы с таким же, как у машин, количеством ресурсов.

В работе [16] рассматривалась уже аппаратная виртуализация. В качестве гипервизора использовался KVM. Авторы исследовали производительность одной виртуальной машины с помощью приложений, написанных на Matlab. Эти приложения так же измеряли производительность вычислений на процессоре, пропускную способность оперативной памяти, производительность работы с жестким диском.

После чего, с помощью тех же приложений, проводилось исследование деградации производительности при использовании машинами одного и того же ядра, то есть в случае, когда обе виртуальные машины использовали один и тот же L1 и L2 кэши. Производительность была снижена из-за того, что гипервизор запускал виртуальные машины на ядре не параллельно, а последовательно.

В третьем эксперименте проводились исследования, в которых виртуальные машины работали на разных ядрах одного процессора. В данной конфигурации машины обладали разными L1-кэшами, но одним L2-кэшем. Производительность была выше, чем во втором эксперименте, но ниже, чем в первом.

Во всех экспериментах в случае нескольких машин в каждой из них запускались одни и те же приложения.

Можно заключить, что использование специализированных систем

оценки производительности является подходом к измерению производительности. Результат работы таких приложений на единственной запущенной машине принимается за базовую производительность. После чего происходит запуск дополнительных машин, и во всех запускаются системы оценки. Размещение дополнительных машин может производиться по двум разным стратегиям: размещение виртуальных машин максимально изолированно друг от друга или с частичным пересечением по доступным ресурсам.

В дальнейшем в данной работе будет использоваться термин бенчмарк, означающий систему оценки производительности.

Стоит отметить, что в данных работах не исследовалась деградация производительности в случае, когда коэффициент перекрытия превышает 1 (см. Введение).

В качестве дополнительных источников исследований производительности виртуальных машин можно рассмотреть работы [17] и [18]. Однако приведенные в них математические модели предназначены для решения проблемы миграции виртуальных машин и деградации их производительности в этом случае, что выходит за рамки данной работы.

3.2. Производительность виртуальных машин и контейнеров

Существует ряд работ, сравнивающих производительность виртуальных машин и контейнеров.

В работе [19] в качестве гипервизора был выбран KVM, а в качестве системы управления контейнерами - Docker. В качестве приложений, оценивающих производительность, использовались бенчмарки, а в качестве реального приложения - MySQL. Сравнение производилось между контейнером и виртуальной машиной, которым были доступны все ресурсы

системы. По результатам работы можно заключить, что контейнеры превосходят виртуальную машину, однако незначительно.

В работе [20] сравнивалась производительность виртуальной машины под управлением гипервизора Virtual Box и Docker-контейнера. Каждому были доступны все ресурсы системы. В качестве бенчмарков использовались стандартные бенчмарки, входящие в пакет бенчмарков Phoronix-test-suite [21]. По результатам экспериментов производительность контейнера либо значительно превышает производительность виртуальной машины, либо практически эквивалентна, но так же в большую сторону.

Работа [13] затронула проблему деградации производительности при небольших (1.5 - 2) размерах коэффициента перекрытия. В данной работе авторы сравнивали виртуальные машины под управлением KVM и LXC-контейнеры. В них запускались специально разработанные бенчмарки. По результатам данной работы можно заключить, что контейнеры даже при коэффициенте перекрытия более 1 показывают большую производительность, чем виртуальные машины. Авторы работы связывают это с тем, что виртуальные машины, в отличие от контейнеров, жестко ограничены в своих ресурсах. То есть гипервизор не даст виртуальной машине ресурсов больше, чем ей выделено. Однако система управления контейнерами может такое позволить применительно к контейнерам.

3.3. Выводы

По результатам обзора можно сделать следующие выводы:

1. Для измерения производительности можно использовать готовые системы оценки производительности, например, систему Phoronix-test-suite.
2. Базовые измерения производительности должны производиться в

виртуальной сущности, то есть виртуальной машине или контейнере, запущенной в единственном экземпляре.

3. Запуск дополнительных виртуальных сущностей должен производиться по двум стратегиям: либо новой сущности доступны все ресурсы, либо только часть.
4. Следующая гипотеза может быть подтверждена: контейнеры обладают более высокой производительностью по сравнению с виртуальными машинами, а деградация производительности виртуальных машин при увеличении коэффициента перекрытия происходит быстрее, чем у контейнеров.

Глава 4.

Практическая реализация

4.1. Требования

По результатам обзора, проведенном в 3, были сделаны выводы о требованиях к системе измерения и сравнения производительности виртуальных машин и контейнеров, а так же измерения и сравнения процесса деградации производительности виртуальных машин и контейнеров.

Производительность предлагается измерять с помощью бенчмарков. В силу их великого разнообразия, система измерения должна быть способна запустить любой бенчмарк в виртуальной сущности. Стоит отметить, что не существует понятия производительность виртуальной машины или контейнера в отрыве от термина производительность приложения. То, насколько производительна виртуальная сущность, можно оценивать только по результатам работы приложения, измеряющего производительность в данной виртуальной сущности.

В качестве виртуальных сущностей, используемых в системе, предлагается использовать виртуальные машины под управлением гипервизора KVM и Docker-контейнеры.

Система будет работать с одним сервером со своим набором физических ресурсов. Все виртуальные сущности будут запускаться только на

этом сервере.

Пусть X - количество доступных ядер процессора на сервере, а χ_i - количество ядер, требующихся для i -ой виртуальной сущности. Введем коэффициент $\theta = \frac{\sum_i \chi_i}{X}$ - коэффициент перекрытия (overlap), являющийся отношением суммы ядер, требующихся виртуальным сущностям системы, к количеству ядер сервера.

Система должна уметь запускать виртуальные сущности с ограничением на доступные ресурсы и без ограничений. Под ресурсами понимается объем оперативной памяти и количество ядер процессора, а так же доступный объем жесткого диска.

В первом случае предполагается, что все виртуальные сущности обладают одинаковыми требованиями по количеству ресурсов. Тогда при увеличении количества запущенных виртуальных сущностей коэффициент перекрытия растет со скоростью $v = \frac{\chi}{X}$, где χ - количество ядер виртуальной сущности, X - общее число ядер сервера.

Во втором случае предполагается, что виртуальной сущности доступны все ресурсы системы. Тогда при увеличении количества запущенных виртуальных сущностей коэффициент перекрытия растет со скоростью $v = 1.0$.

Система будет исследовать деградацию производительности виртуальной сущности как зависимость производительности P от количества запущенных виртуальных сущностей.

4.2. Алгоритм работы реализованной системы

На рисунке 4.1 представлена диаграмма классов разработанной системы.

Под носителем в данном разделе будет пониматься та виртуальная сущность, исследование которой производится, то есть либо контейнер,

либо виртуальная машина.

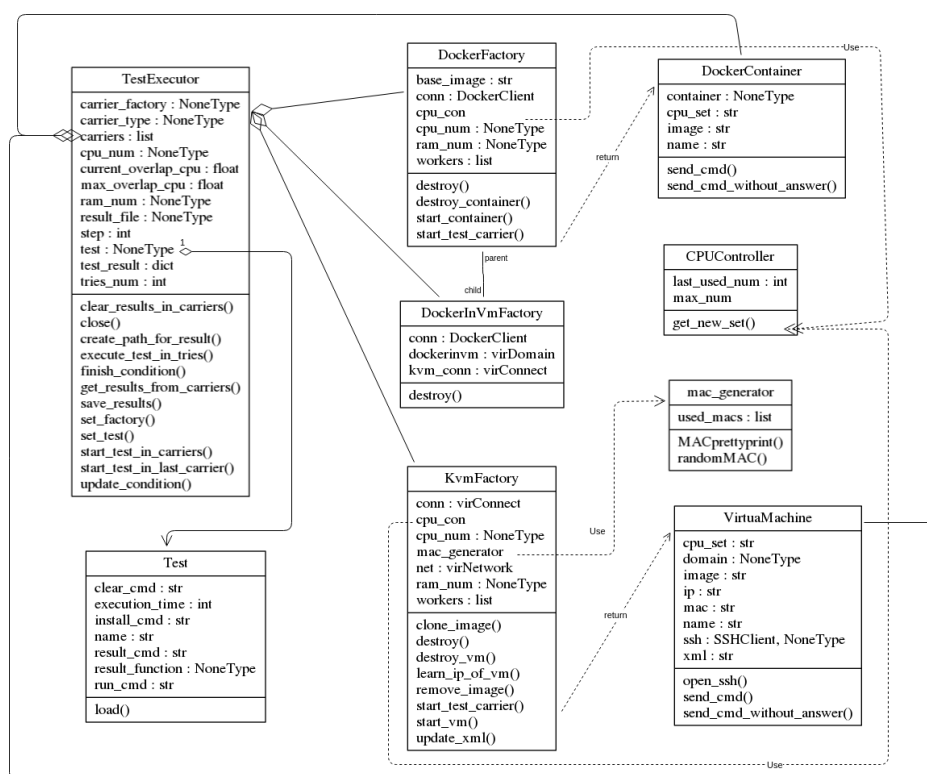


Рис. 4.1. UML-диаграмма классов

Основной алгоритм работы системы можно описать следующим образом. В начале работы создается объект класса `TestExecutor`, содержащий описание теста, который должен производиться в виртуальном носителе, тип исследуемого виртуального носителя, а так же ограничения на ресурсы экземпляров носителя.

После этого происходит запуск первого виртуального носителя. В нем производится установка необходимого приложения-теста, команда по установке содержится в описании приложения. Образ, полученный в результате установки приложения, становится базовым, то есть тем, чьи копии будут использовать будущие экземпляры виртуальной сущности в данном запуске системы.

После получения объекта виртуальной сущности, являющейся объектом класса `DockerContainer` или `VirtualMachine` (в зависимости от конфигурации `TestExecutor`), во всех уже запущенных экземплярах проис-

ходит запуск приложения. Чтобы не было ситуации, при которой запуск в одном носителе происходит раньше, чем в другом, запуск приложения планируется на определенный момент в будущем.

После этого система ожидает завершения работы приложения-теста в последнем запущенном носителе. Это необходимо для корректного получения результатов, которые будут готовы только после завершения работы во всех носителях.

При проведении тестовых запусков было замечено, что не всегда приложения завершаются во всех носителях в момент завершения работы приложения в последнем носителе. Поэтому получение результатов выглядит следующим образом: по всем носителям запускается команда сбора результатов, так же содержащаяся в описании приложения-теста. Если результаты отсутствуют в носителе, то это означает, что работа приложения в данном носителе еще не завершена. За каждый неполученный результат система добавляет время ожидания. В случае, если получены не все результаты, система блокируется на это время ожидания. По истечении времени, происходит новая попытка получить результаты.

После успешного получения результатов, система переходит к запуску нового экземпляра и повторения шагов, описанных выше.

При запуске каждого экземпляра происходит обновление текущего значения коэффициента перекрытия.

В классе `TestExecutor` существует поле `max_overlap_cpu`, В данном поле содержится то значение коэффициента перекрытия, при котором надо остановить систему и удалить текущие запущенные виртуальные сущности.

Классы `VirtualMachine` и `DockerContainer` содержат методы `send_cmd` и `send_cmd_without_waiting`. С помощью данных методов происходит исполнение переданной как параметр команды в виртуальной сущности.

За запуск новых экземпляров виртуальной сущности отвечают клас-

са KvmFactory, DockerFactory и DockerInVmFactory. Данные классы содержат методы `start_test_carrier`, которые возвращают объекты типа DockerContainer или VirtualMachine. Так же эти классы содержат поле типа CPUController. Этот класс отвечает за планирование ядер процессора виртуальным сущностям.

Класс Test хранит описание теста в виде, понятном классу TestExecutor. Он создается один раз при запуске системы.

Класс `mac_generator` является вспомогательным классом для класса KvmFactory. Его функция - генерация уникальных MAC-адресов для виртуальных машин.

4.3. Применимость

Реализованная система предназначена в первую очередь для того, чтобы исследовать производительность виртуальных машин и контейнеров. Несмотря на то, что в данной работе сравниваются KVM-машины и Docker-контейнеры, система легко может быть доработана для работы с другими гипервизорами и системами управления контейнерами.

В данной системе может быть запущено любое приложение, причем это может быть необязательно стандартное приложение оценки производительности, как приложения из Phoronox-test-suite. Поэтому данная система может быть использована в облачных системах для исследования поведения приложений в облачных системах. С помощью нее возможно запустить приложение в разных виртуальных сущностях, изучить его производительность и сделать вывод о наилучшей стратегии запуска данного приложения в облаке, причем с учетом возможного масштабирования и работы данного приложения рядом с другими приложениями, работающими на том же физическом сервере.

Глава 5.

Экспериментальное исследование

В данной главе описаны проведенные эксперименты и их результаты. В данных экспериментах использовалась разработанная система.

В качестве контейнеров использовались Docker-контейнеры, в качестве виртуальной машины - виртуальные машины под управлением гипервизора KVM.

Приложения, определяющие производительность виртуальной сущности в данных экспериментах, по разному используют физические ресурсы системы. Некоторые приложения могут использовать все доступные ядра процессора, а некоторые лишь часть. Некоторые приложения могут активно совершать обмены данными с оперативной памятью и жестким диском.

Каждый эксперимент проходил по одной и той же схеме. Сначала происходили запуски приложения, измеряющего производительность, в виртуальных сущностях, которым доступна лишь часть ресурсов хостовой операционной системы, с последовательным увеличением количества виртуальных сущностей до тех пор, пока коэффициент перекрытия (см. 4.1) не превысит некоторого порогового значения, отличающегося в раз-

ных экспериментах.

После этого происходит измерение производительностей виртуальных сущностей в случае, когда им доступны все ресурсы хостовой операционной системы.

Запуски приложения происходили либо только в контейнерах, либо только в виртуальных машинах, но никогда не одновременно.

Мерой производительности виртуальной сущности при каждом значении коэффициента перекрытия может являться как среднее значение производительности по всем экземплярам виртуальной сущности, так и медиана от этих данных. В каждом эксперименте измерялись обе эти величины.

Каждый эксперимент производился по три раза, в качестве итогового результата для каждого значения коэффициента перекрытия принималось усредненное значение по результатам из разных запусков, полученных для того же значения коэффициента перекрытия.

Так же в каждом эксперименте рассматривалась зависимость значения величины разности производительности приложения в Docker-контейнере и в виртуальной машине KVM от коэффициента перекрытия.

5.1. Экспериментальный стенд

Ниже представлены характеристики физического сервера, на котором производились эксперименты.

Процессор: Intel Xeon E5-2640 v4 @ 3.40GHz (20 ядер)

Материнская плата: Supermicro X10DDW-i v1.10

Оперативная память: 16384 Мбайт

Жесткий диск: HDD 1000GB Seagate ST1000NM0033-9ZM

Операционная система: Linux CentOS 7.0, Kernel: 3.10.0-693.17.1.el7.x86_64

5.2. Подбор паролей

В данном эксперименте используется приложение, подбирающее пароль. Оно работает по алгоритму John-the-Ripper. В рамках работы принцип работы данного алгоритма не рассматривается. Мерой производительности данного приложения является количество хэшей паролей, которое оно перебирает за единицу времени (секунду). Чем больше полученная величина, тем выше производительность.

Особенностью данного приложения является то, что оно максимально загружает все доступные ядра процессора. Поэтому оно используется для измерения производительности центрального процессора.

Рассмотрим результаты, полученные в случае запуска виртуальных сущностей, каждой из которых доступны не все ресурсы операционной системы. Конкретно, каждой виртуальной сущности доступно по 4 ядра ЦП, 256 Мбайт памяти и 10 Гбайт жесткого диска. В данном случае коэффициент перекрытия при запуске дополнительной виртуальной сущности увеличивается на 0.2. Результаты эксперимента представлены на графиках 5.1 и 5.2.

Стоит заметить, что хостовая операционная система не позволила превысить коэффициент перекрытия 6.0 при запуске виртуальных машин под управлением KVM. На Docker-контейнеры таких ограничений наложено не было.

Визуально можно заметить, что производительность приложения в контейнере, запущенном в единственном экземпляре, превышает производительность его же в виртуальной машине, так же работающей в единственном экземпляре. Данное свойство отображено в первых точках графиков.

Любопытной является форма графиков. Даже в случае, когда виртуальные сущности не конкурируют напрямую за ядра процессора, производительность приложения падает, причем нелинейно. Это может быть

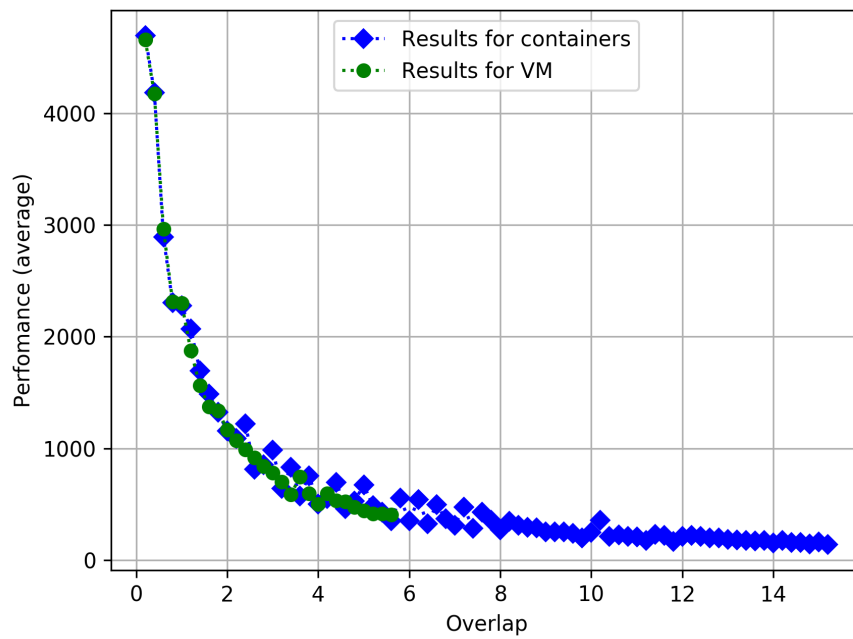


Рис. 5.1. Зависимость среднего значения производительности приложения «Подбор паролей», запущенного в Docker-контейнерах и KVM-виртуальных машинах, от коэффициента перекрытия в случае неполных ресурсов

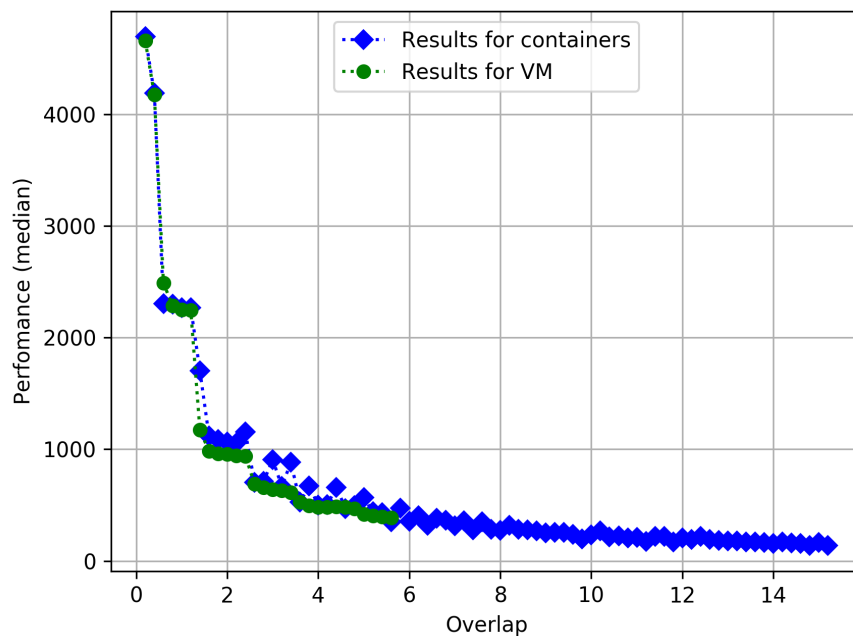


Рис. 5.2. Зависимость медианного значения производительности приложения «Подбор паролей», запущенного в Docker-контейнерах и KVM-виртуальных машинах, от коэффициента перекрытия в случае неполных ресурсов

свидетельством тому, что производительность одной виртуальной сущности влияет на производительность другой нетривиальным способом (см. статью [15]).

Рассмотрим отдельно зависимость значения величины разности производительности приложения в Docker-контейнере и в виртуальной машине KVM от коэффициента перекрытия на рисунке 5.3 в случае неполных ресурсов на участке от 1.0 до 5.0.

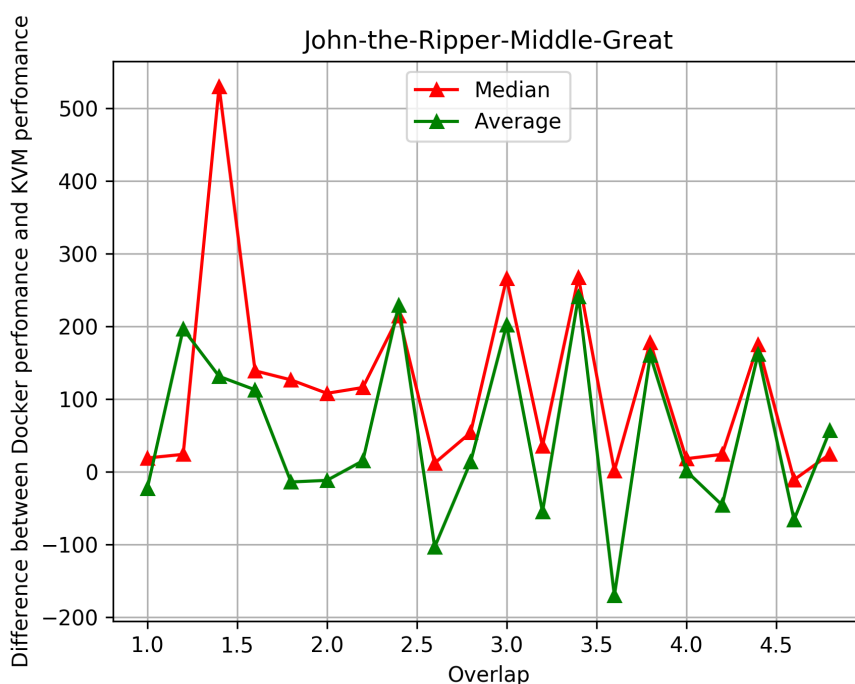


Рис. 5.3. Зависимость разности производительности приложения «Подбор паролей», запущенного в Docker-контейнерах и KVM-виртуальных машинах, от коэффициента перекрытия в случае неполных ресурсов

Как видно из графика, большая часть значений и в случае медиан, и в случае средних значений лежит выше 0, что означает, что приложение в контейнерах чаще более производительнее, чем в виртуальных машинах в случае, когда виртуальным сущностям доступны не все ресурсы системы.

Рассмотрим результаты, полученные в случае запуска виртуальных сущностей, каждой из которых доступны все ресурсы операционной системы. В данном случае коэффициент перекрытия при запуске дополнительной виртуальной сущности увеличивается на 1.0. Результаты экспе-

римента представлены на графиках 5.4 и 5.5.

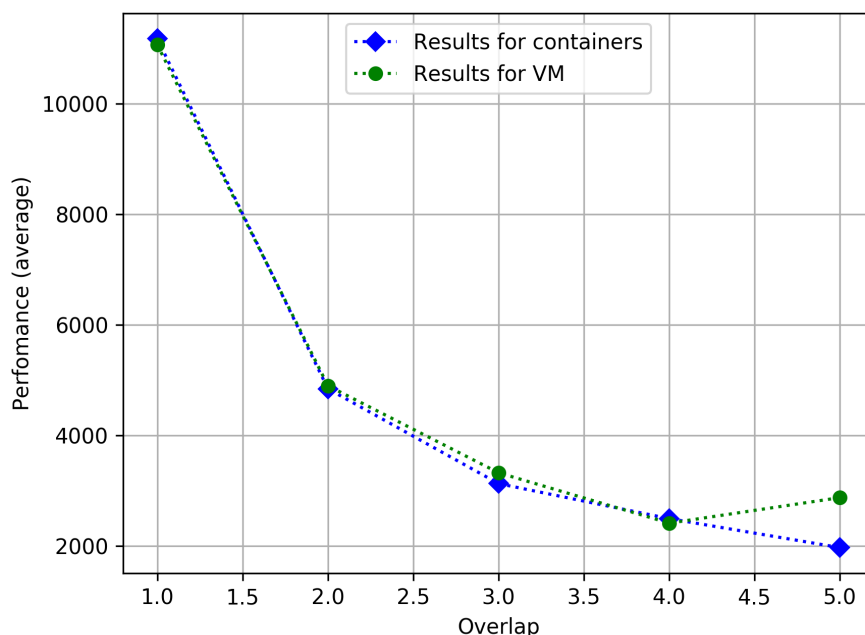


Рис. 5.4. Зависимость среднего значения производительности приложения «Подбор паролей», запущенного в Docker-контейнерах и KVM-виртуальных машинах, от коэффициента перекрытия в случае полных ресурсов

Стоит отметить, что вновь не удалось превысить коэффициент перекрытия 6.0 в случае виртуальных машин. Поэтому на графиках представлен участок для коэффициента перекрытия от 1.0 до 5.0.

Рассмотрим отдельно зависимость значения величины разности производительности приложения в Docker-контейнере и в виртуальной машине KVM от коэффициента перекрытия на рисунке 5.6 на участке от 1.0 до 5.0.

Из графика видно, что производительность приложения в контейнере в случае одного экземпляра выше производительности в виртуальной машине. Однако при увеличении количества экземпляров нельзя утверждать то же самое, то есть производительность приложений в виртуальной машине может быть как выше, так и ниже производительность в контейнере. Из этого можно сделать вывод, что производительности контейнеров и виртуальных машин в случае, когда виртуальным сущностям

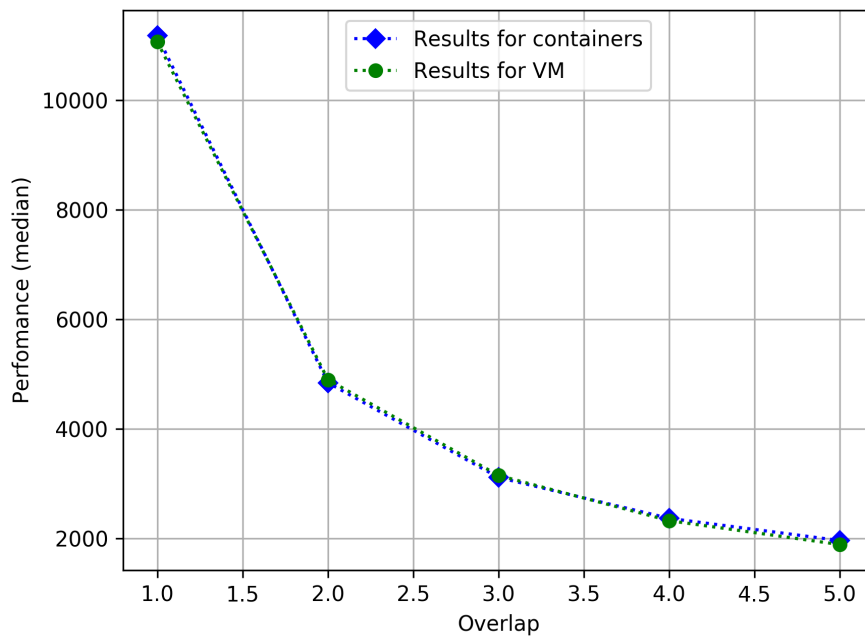


Рис. 5.5. Зависимость медианного значения производительности приложения «Подбор паролей», запущенного в Docker-контейнерах и KVM-виртуальных машинах, от коэффициента перекрытия в случае полных ресурсов

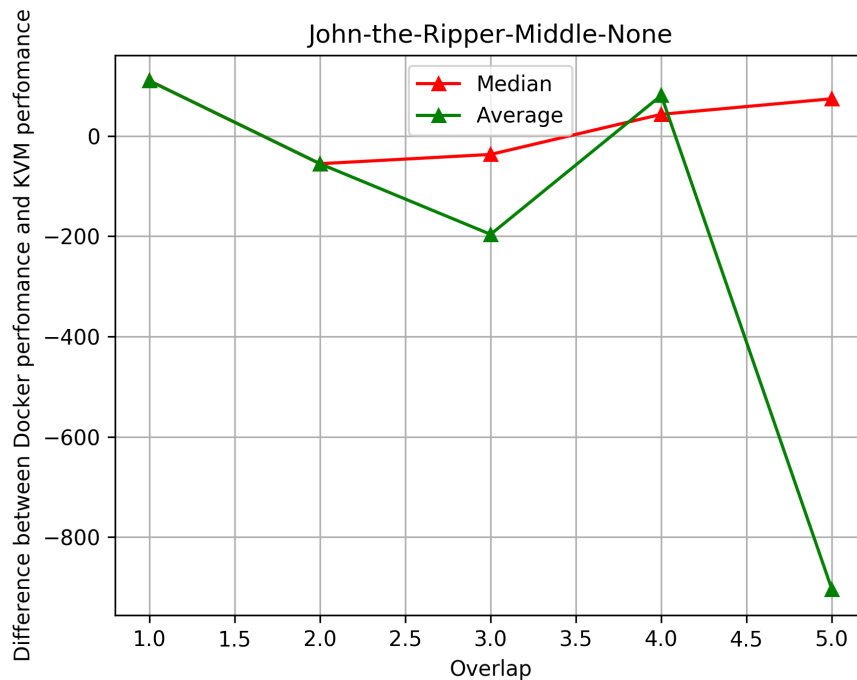


Рис. 5.6. Зависимость разности производительности приложения «Подбор паролей», запущенного в Docker-контейнерах и KVM-виртуальных машинах, от коэффициента перекрытия в случае полных ресурсов

доступны все ресурсы системы, приблизительно одинаковы.

В итоге можно заключить, что производительность приложения «Подбор паролей» в контейнерах либо выше, либо равна производительности приложения в виртуальных машинах при увеличении коэффициента перекрытия.

В дальнейших экспериментах не будет рассматриваться производительность приложений при значении коэффициента перекрытия больше, чем 6.0, в силу отсутствия возможности получить значения производительности приложений в виртуальных машинах.

5.3. Python-код

В данном эксперименте используется приложение, написанное на языке Python. Мерой производительности данного приложения является его время работы. Чем больше полученная величина, тем ниже производительность.

Подобное приложение можно считать одним из типовых приложений, запускаемых в облачных системах.

Рассмотрим результаты, полученные в случае запуска виртуальных сущностей, каждой из которых доступны не все ресурсы операционной системы. Конкретно, каждой виртуальной сущности доступно по 4 ядра ЦП, 256 Мбайт памяти и 10 Гбайт жесткого диска. В данном случае коэффициент перекрытия при запуске дополнительной виртуальной сущности увеличивается на 0.2. Результаты эксперимента представлены на графиках 5.7 и 5.8.

Можно видеть, что в Docker-контейнерах время выполнения Python-кода меньше, чем время выполнения в виртуальных машинах KVM, при значениях коэффициента перекрытия до 3.0. Однако при больших значениях коэффициента наблюдается, что время выполнения меньше то

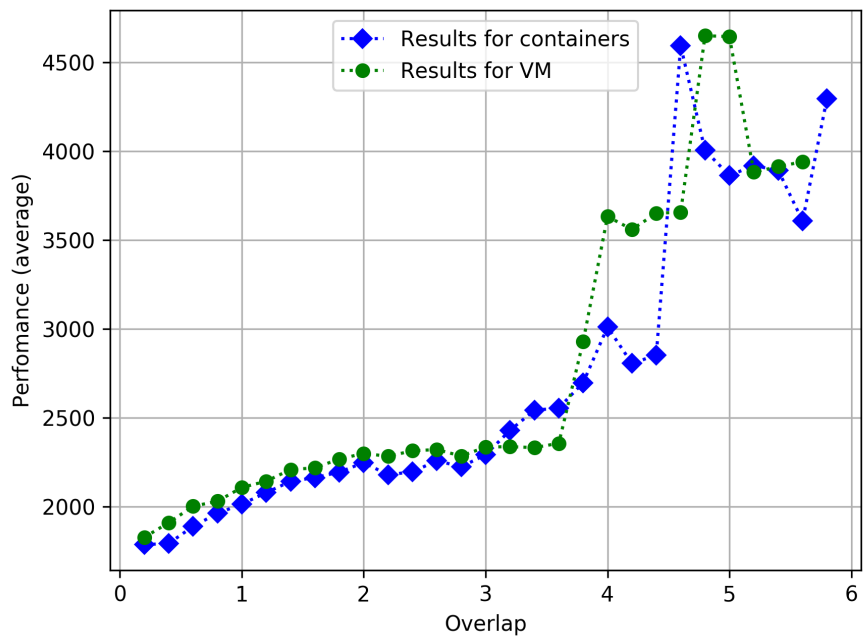


Рис. 5.7. Зависимость среднего значения производительности приложения «Python-код», запущенного в Docker-контейнерах и KVM-виртуальных машинах, от коэффициента перекрытия в случае неполных ресурсов

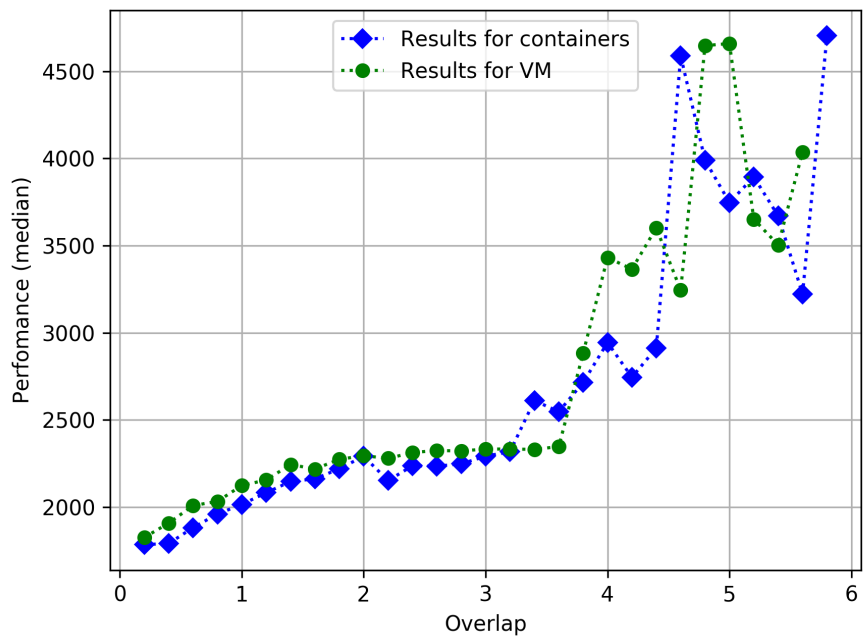


Рис. 5.8. Зависимость медианного значения производительности приложения «Python-код», запущенного в Docker-контейнерах и KVM-виртуальных машинах, от коэффициента перекрытия в случае неполных ресурсов

в виртуальных машинах, то в контейнерах. Можно заключить, что при коэффициенте перекрытия больше 3.0 время выполнения кода и в контейнерах, и в виртуальных машинах приблизительно одинаково.

Рассмотрим результаты, полученные в случае запуска виртуальных сущностей, каждой из которых доступны все ресурсы операционной системы. В данном случае коэффициент перекрытия при запуске дополнительной виртуальной сущности увеличивается на 1.0. Результаты эксперимента представлены на графиках 5.9 и 5.10.

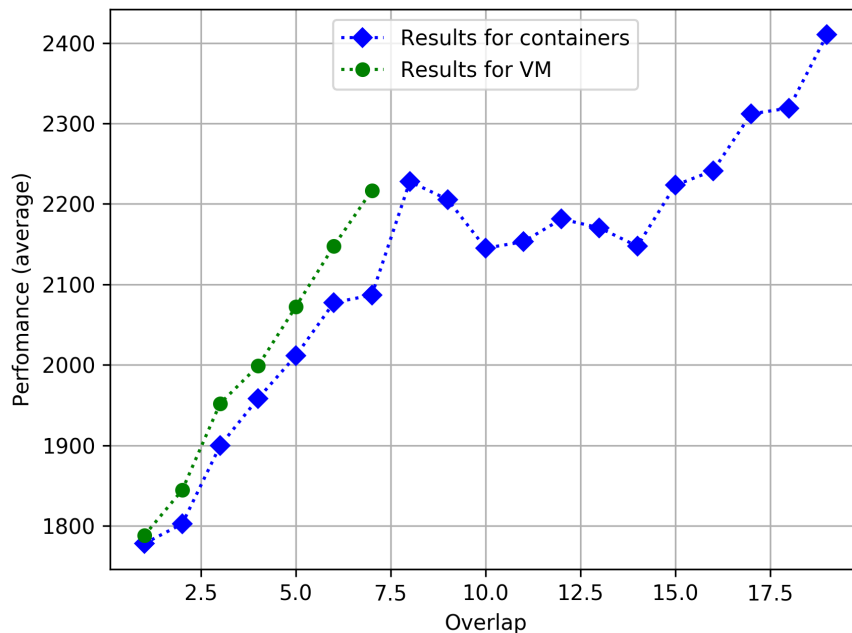


Рис. 5.9. Зависимость среднего значения производительности приложения «Python-код», запущенного в Docker-контейнерах и KVM-виртуальных машинах, от коэффициента перекрытия в случае полных ресурсов

Можно видеть, что при доступности всех ресурсов, время исполнения кода в виртуальных машинах KVM значительно выше, чем в Docker-контейнерах.

По результатам эксперимента можно сделать вывод, что приложение «Python-код» обладает более высокой производительностью в Docker-контейнерах, чем в виртуальных машинах под управлением KVM.

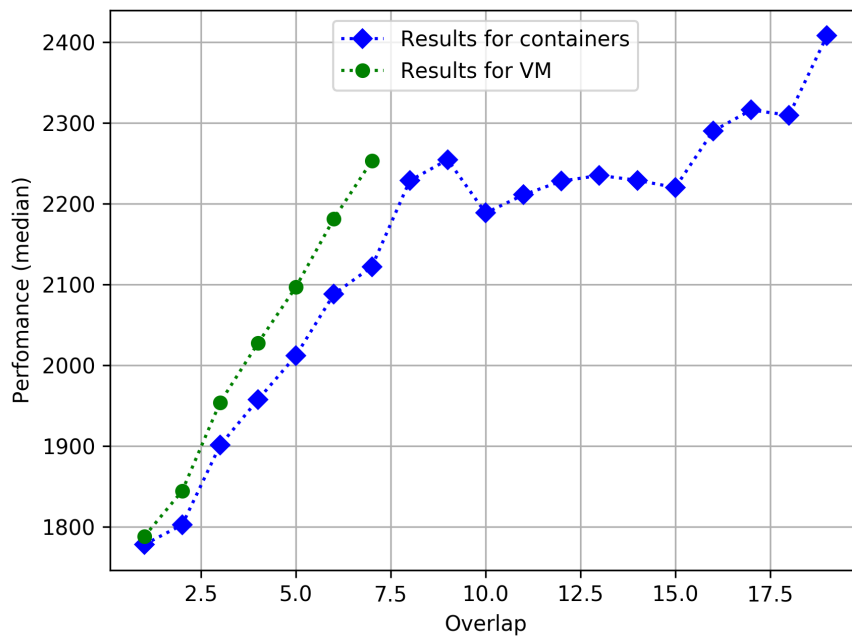


Рис. 5.10. Зависимость медианного значения производительности приложения «Python-код», запущенного в Docker-контейнерах и KVM-виртуальных машинах, от коэффициента перекрытия в случае полных ресурсов

5.4. Пропускная способность оперативной памяти

Данное приложение измеряет пропускную способность оперативной памяти в виртуальной сущности. Основная подсистема, на которую идет нагрузка, это оперативная память. Мерой производительности будет пропускная способность. Чем больше результат, тем выше производительность.

Рассмотрим результаты, полученные в случае запуска виртуальных сущностей, каждой из которых доступны не все ресурсы операционной системы. Конкретно, каждой виртуальной сущности доступно по 4 ядра ЦП, 512 Мбайт памяти и 10 Гбайт жесткого диска. В данном случае коэффициент перекрытия при запуске дополнительной виртуальной сущности увеличивается на 0.2. Результаты эксперимента представлены на графиках 5.11 и 5.12.

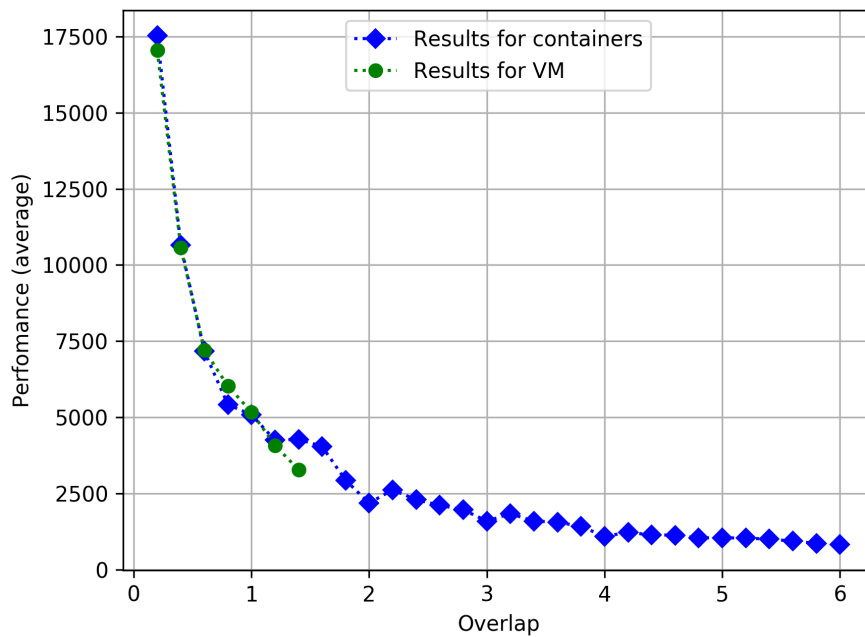


Рис. 5.11. Зависимость среднего значения производительности приложения «Пропускная способность оперативной памяти», запущенного в Docker-контейнерах и KVM-виртуальных машинах, от коэффициента перекрытия в случае неполных ресурсов

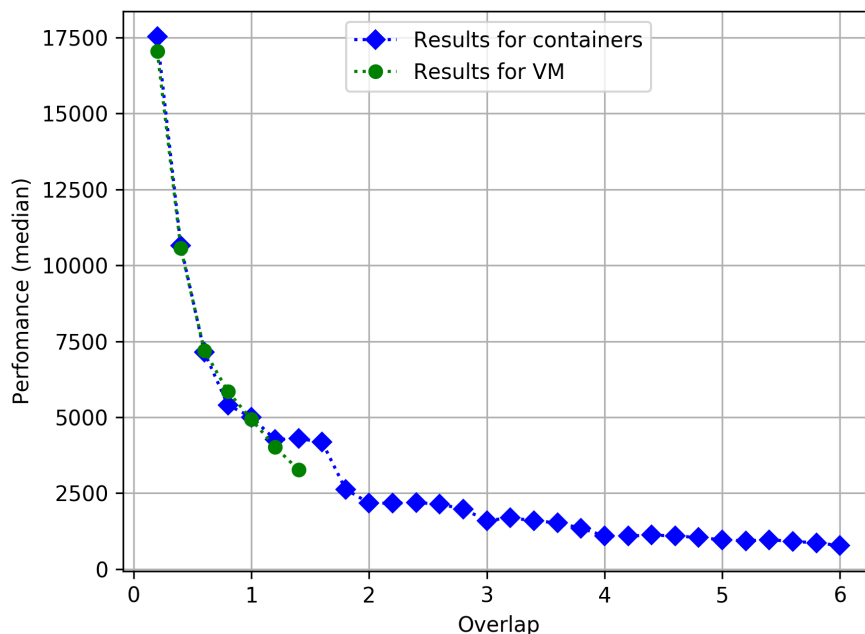


Рис. 5.12. Зависимость медианного значения производительности приложения «Пропускная способность оперативной памяти», запущенного в Docker-контейнерах и KVM-виртуальных машинах, от коэффициента перекрытия в случае неполных ресурсов

Стоит заметить, что в ходе эксперимента оказалось невозможным запустить более 8 виртуальных машин KVM с данным приложением. Пропускная способность оперативной памяти в контейнере выше, чем в виртуальной машине в случае одной виртуальной сущности.

Рассмотрим отдельно зависимость значения величины разности производительности приложения в Docker-контейнере и в виртуальной машине KVM от коэффициента перекрытия на рисунке 5.13 на участке от 0.2 до 1.4.

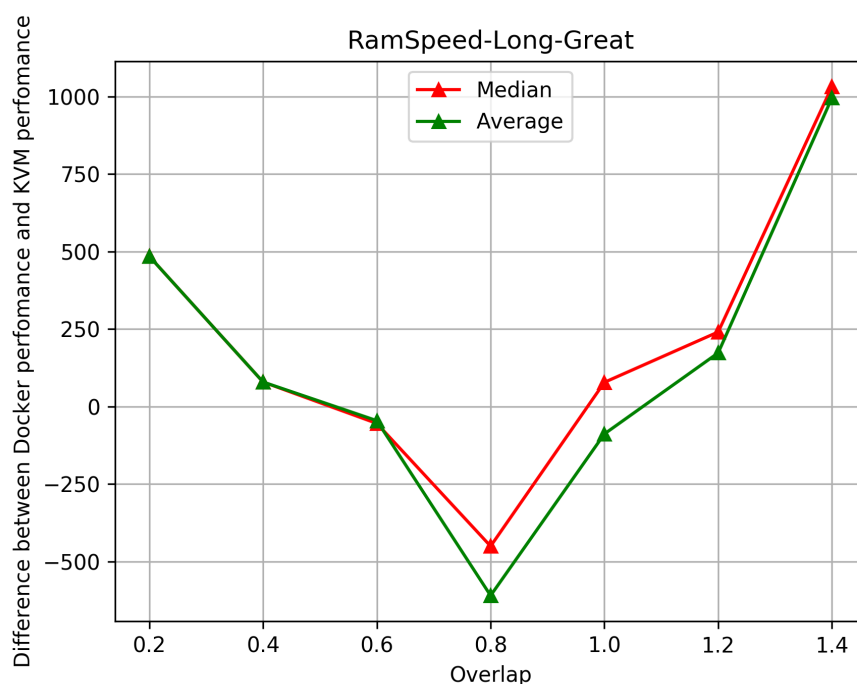


Рис. 5.13. Зависимость разности производительности приложения «Пропускная способность оперативной памяти», запущенного в Docker-контейнерах и KVM-виртуальных машинах, от коэффициента перекрытия в случае неполных ресурсов

Из этого графика следует, что пропускная способность оперативной памяти в контейнере в среднем выше, чем в виртуальной машине, однако нельзя утверждать по данным результатам, что пропускная способность в контейнере всегда выше, чем в виртуальной машине.

Рассмотрим результаты, полученные в случае запуска виртуальных сущностей, каждой из которых доступны все ресурсы операционной системы. В данном случае коэффициент перекрытия при запуске дополни-

тельной виртуальной сущности увеличивается на 1.0. Результаты эксперимента представлены на графиках 5.14 и 5.15.

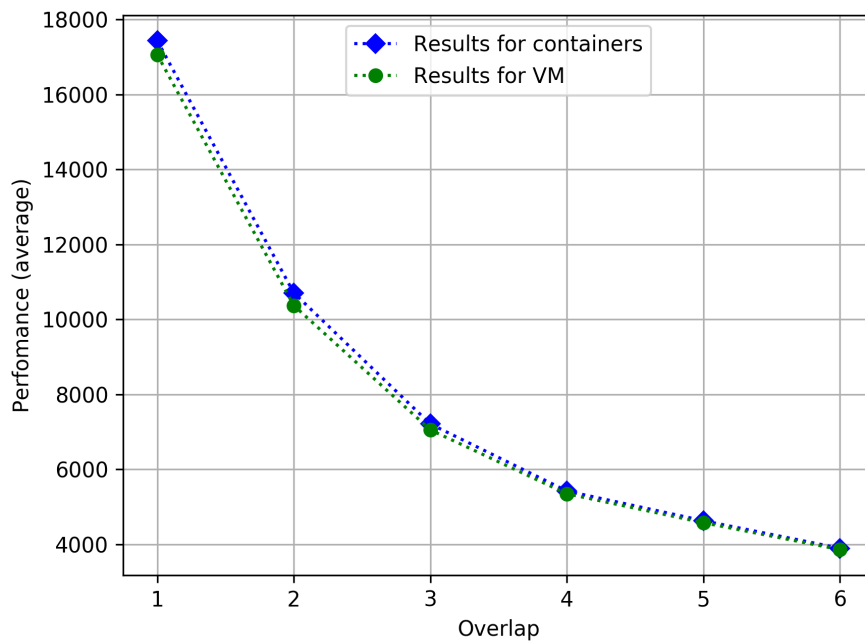


Рис. 5.14. Зависимость среднего значения производительности приложения «Пропускная способность оперативной памяти», запущенного в Docker-контейнерах и KVM-виртуальных машинах, от коэффициента перекрытия в случае полных ресурсов

Рассмотрим отдельно зависимость значения величины разности производительности приложения в Docker-контейнере и в виртуальной машине KVM от коэффициента перекрытия на рисунке 5.16 на участке от 0.0 до 6.0.

По результатам данного эксперимента видно, что в случае доступа ко всем ресурсам системы у виртуальных сущностей пропускная способность оперативной памяти больше у контейнеров, чем у виртуальных машин, причем при всех значениях коэффициента перекрытия.

5.5. Выводы

По результатам экспериментального исследования были сделаны следующие выводы, что:

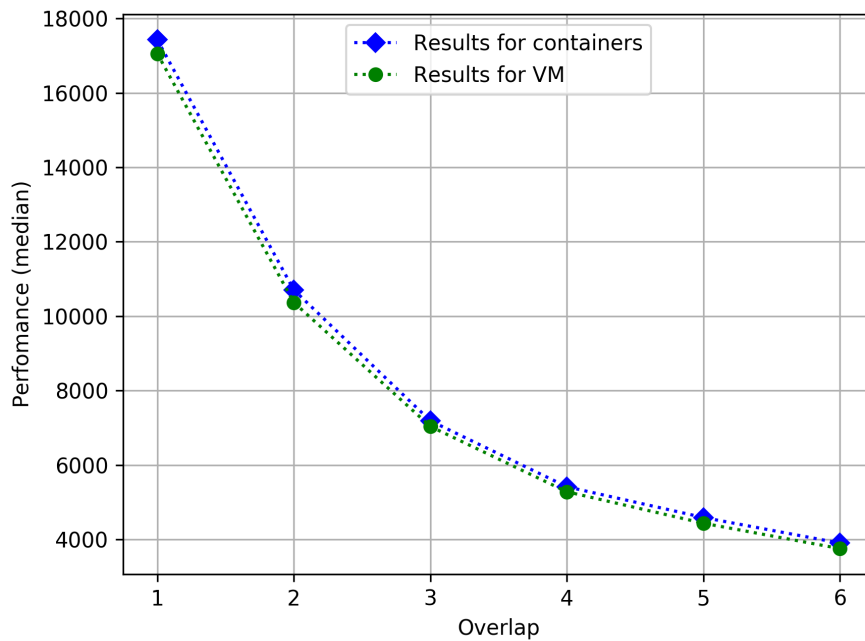


Рис. 5.15. Зависимость медианного значения производительности приложения «Пропускная способность оперативной памяти», запущенного в Docker-контейнерах и KVM-виртуальных машинах, от коэффициента перекрытия в случае полных ресурсов

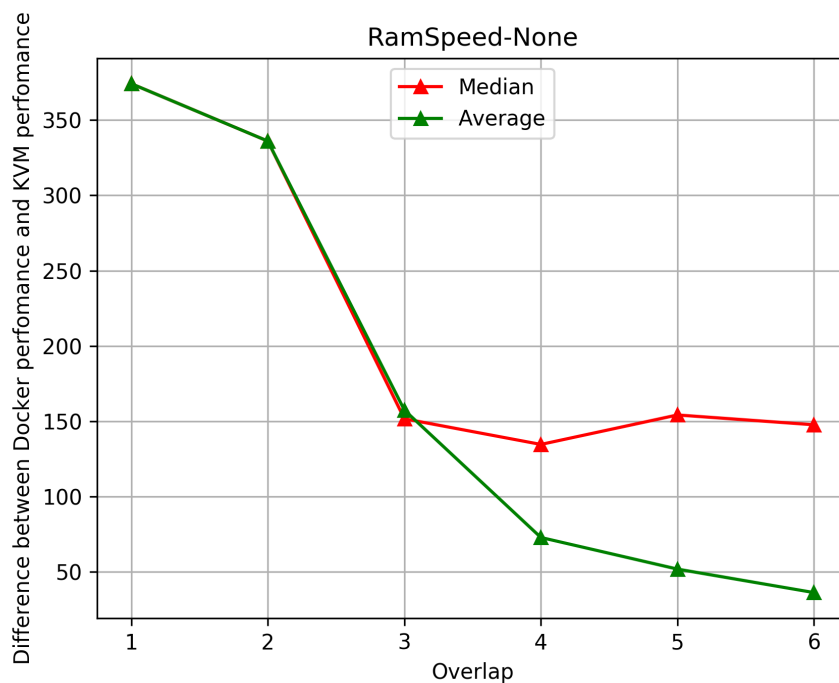


Рис. 5.16. Зависимость разности производительности приложения «Пропускная способность ОП», запущенного в Docker-контейнерах и KVM-виртуальных машинах, от коэффициента перекрытия в случае неполных ресурсов

1. В случае, когда запущен только один экземпляр виртуальной сущности, производительность приложений в Docker-контейнере выше, чем производительность приложений в виртуальной машине под управлением KVM.
2. Деградация производительности приложений в Docker-контейнере происходит либо медленнее, либо приблизительно так же, как деградация производительности приложений в виртуальных машинах под управлением KVM, в зависимости от приложения.
3. В операционной системе может быть запущено значительно больше Docker-контейнеров, чем виртуальных машин под управлением гипервизора KVM. Однако стоит отметить, что ограничение на коэффициент перекрытия для виртуальных машин под управлением KVM в 6.0 скорее всего возможно изменить при компиляции ядра.

Глава 6.

Заключение

В данной работе исследовалась производительность виртуальных машин и контейнеров в облачных системах, а так же деградация производительности контейнеров и виртуальных машин при увеличении коэффициента перекрытия, то есть ситуации, когда несколько виртуальных сущностей используют одни и те же физические ресурсы.

Была выдвинута гипотеза, что производительность контейнеров выше производительности виртуальных машин. При этом при увеличении коэффициента перекрытия деградация производительности контейнеров происходит медленнее, чем у виртуальных машин.

По результатам обзора, проведенного в главе 2, было получено, что наиболее популярные технологии виртуализации в облачных системах - это аппаратная виртуализация и контейнерная виртуализация. Две этих технологии обладают своими достоинствами и недостатками. Так, виртуальные машины вынуждены поддерживать гостевое ядро операционной системы, на что тратятся ресурсы хостовой операционной системы. Контейнеры используют ядро хостовой операционной системы, экономя ресурсы, но не могут запускать приложения, требующих модификации ядра, без модификации ядра хостовой операционной системы.

В обзоре существующих решений по сравнению производительности

виртуальных машин и контейнеров в главе 3 был рассмотрен ряд работ. Часть работ была посвящена тому, что влияние виртуальных машин друг на друга в рамках одной хостовой системы является сложным процессом, а потому не существует точных математических моделей предсказания производительности виртуальных машин и ее деградации. Далее, в части работ сравнивалась производительность контейнеров и виртуальных машин с помощью запуска в них приложений, измеряющих производительность, так называемых бенчмарков. Из этого было заключено, что производительность виртуальной сущности - это производительность приложения, запущенного в этой сущности, то есть производительность может сильно зависеть от самого приложения. Так же было заключено, что необходимо исследовать производительность виртуальных сущностей в двух ситуациях: когда им доступны все ресурсы системы и когда доступна лишь часть.

В работах сравнивались разные технологии виртуализации, при этом контейнеры показывали большую или эквивалентную виртуальным машинам производительность. Однако практически полностью отсутствовало исследование темы деградации производительности виртуальных сущностей при увеличении коэффициента перекрытия.

В главе 4 рассматривалась реализация системы, позволяющей сравнить производительность приложений в контейнерах и в виртуальных машинах. Данная система может быть использована в реальных облачных системах при необходимости исследовать поведение приложений при увеличении коэффициента перекрытия в различных виртуальных сущностях и их стратегиях размещения.

В главе 5 рассматривались проведенные с помощью разработанной системы эксперименты. По результатам экспериментов были сделаны выводы, что контейнеры обладают более высокой производительностью, чем виртуальные машины. Однако то, насколько контейнеры произво-

дительнее виртуальных машин, зависит от приложения, запускаемого в них. Производительность некоторых приложений практически одинакова и в контейнерах, и в виртуальных машинах. При увеличении коэффициента перекрытия производительность приложений в контейнерах либо остается выше производительности в виртуальных машинах, либо становится приблизительно эквивалентной.

В итоге можно заключить, что контейнеры более производительны, чем виртуальные машины. Таким образом, использование контейнеров для приложений, не требующих модификации ядра, более предпочтительно. При этом при увеличении количества экземпляров контейнеры остаются производительнее виртуальных машин, либо их производительности становятся эквивалентны.

Литература

- [1] Hyungro Lee. Virtualization Basics: Understanding Techniques and Fundamentals. // School of Informatics and Computing, Indiana University, 2014.
- [2] Homepage of VMWare ESXi. <http://www.vmware.com/ru/products/esxi-and-esx.html> (дата обращения 01.09.2017)
- [3] Hasan Fayyad-Kazan, Luc Perneel, Martin Timmerman. Full and Para-Virtualization with Xen: A Performance Comparison. // Journal of Emerging Trends in Computing and Information Sciences, 2013.
- [4] Homepage of Xen. <https://www.xenproject.org> (дата обращения 01.09.2017)
- [5] Homepage of VirtualBox. <https://www.virtualbox.org> (дата обращения 01.09.2017)
- [6] Homepage of KVM. <https://www.linux-kvm.org> (дата обращения 01.09.2017)
- [7] Homepage of LXC. URL: <https://linuxcontainers.org/ru/> (дата обращения 01.09.2017)
- [8] Homepage of OpenVZ. URL: <https://openvz.org/> (дата обращения 31.10.2017)

- [9] Homepage of Docker. URL: <https://docs.docker.com/> (дата обращения 01.09.2017)
- [10] Homepage of Rocket. URL: <https://coreos.com/rkt/docs/latest/> (дата обращения 31.10.2017)
- [11] Homepage of DrawBridge. URL: <http://research.microsoft.com/en-us/projects/drawbridge/> (дата обращения 31.10.2017)
- [12] Homepage of runC. URL: <https://runc.io/> (дата обращения 30.04.2017)
- [13] Lucas Chaufournier, Prateek Sharma, Prashant Shenoy. Containers and Virtual Machines at Scale: A Comparative Study. // Middleware '16 Proceedings of the 17th International Middleware Conference, 2016.
- [14] Михеев Павел. Разработка и реализация системы масштабирования виртуальных сетевых функций с помощью легковесной виртуализации. // МГУ, 2017.
- [15] Younggyun Koh, Rob Knauerhase [и др.]. An Analysis of Performance Interference Effects in Virtual Environments. // Performance Analysis of Systems & Software IEEE International Symposium on, 2007.
- [16] George Kousiouris, Tommaso Cucinotta, Theodora Varvarigoua. The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks. // The Journal of Systems and Software, 2011.
- [17] Anton Beloglazov, Rajkumar Buyya. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in Cloud data centers. // Wiley Online Library, 2011.

- [18] Haikun Liu, Hai Jin, Cheng-Zhong Xu, Xiaofei Liao. Performance and energy modeling for live migration of virtual machines. // Springer US, 2013.
- [19] Wes Felter, Alexandre Ferreira, Ram Rajamony, Juan Rubio. An Updated Performance Comparison of Virtual Machines and Linux Containers. // 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2015.
- [20] Rabindra K. Barik, Rakesh K. Lenka [и др.]. Performance Analysis of Virtual Machines and Containers in Cloud Computing. // International Conference on Computing, Communication and Automation (ICCCA2016), 2016.
- [21] Homepage of Phoronix-test-suite. URL: <http://www.phoronix-test-suite.com/> (дата обращения 01.03.2018)