Введение

Геномные конвейеры обычно основаны на комбинации нескольких компонентов исследовательского программного обеспечения сторонних производителей. Эти приложения, как правило, представляют собой академические прототипы, которые часто трудно установить, настроить и развернуть. Программа, реализованная в данной среде, обычно имеет множество неявных зависимостей от программ, библиотек и других компонентов, присутствующих в этой среде. Как следствие, вычислительный рабочий процесс, построенный в одной среде, имеет мало шансов на корректную работу в другой среде без значительные усилия (Garijo et al., 2013). В прошлом технология виртуализации машин предлагалась в качестве ответа на этот вопрос (Howe, 2012; Gent, 2013). Однако такой подход имеет ряд существенных недостатков. Образы виртуальных машин имеют большой размер (обычно несколько гигабайт), поскольку для них требуется полная копия файлов операционной системы (ОС). Даже для того, чтобы добавить или изменить всего один файл, необходимо собрать и развернуть полную копию виртуальной машины. Более того, трудно, если не невозможно, повторно использовать компоненты программного обеспечения или данные внутри виртуальной машины. Их содержание, как правило, непрозрачно, т.е. систематически не описано и не доступно с помощью стандартного инструмента/протокола (Хинсен, 2014).

Технология контейнеров Docker (http://www.docker.com ) в последнее время привлекает все большее внимание научного сообщества, поскольку позволяет запускать приложения в изолированном, автономном пакете, который может эффективно распространяться и выполняться переносимым образом на широком спектре вычислительных платформ (Gerlach et al., 2014; Boettiger, 2015).

Наиболее очевидным преимуществом такого подхода является замена утомительной установки многочисленных компонентов программного обеспечения со сложными зависимостями простой загрузкой единого предварительно созданного готового к запуску образа, содержащего все программное обеспечение и требуемую конфигурацию.

Еще одним преимуществом Docker является то, что он запускает каждый процесс в изолированном контейнере, который создается, начиная с неизменяемого образа. Это предотвращает конфликты с другими установленными программами в среде хостинга и гарантирует выполнение каждого процесса в предсказуемой конфигурации системы, которая не может изменяться с течением времени из-за неправильно настроенного программного обеспечения, системных обновлений или ошибок программирования. Запуск контейнера занимает всего долю секунды, и многие экземпляры могут запускаться в одной и той же среде хостинга. Это возможно, потому что он запускается как изолированный процесс в пользовательском пространстве хост-операционной системы, совместно используя ядро с другими контейнерами.

Исследование IBM Research показывает, что контейнеры Docker приводят к незначительным накладным расходам на производительность процессора и памяти и что приложения, работающие в контейнере работает одинаково или лучше по сравнению с традиционной технологией виртуальных машин во всех тестах (Фелтерет и др., 2014).

Список литературы 2

Облачные вычисления переживают стремительное распространение. Чтобы воспользоваться его преимуществами, например масштабируемостью, эластичностью, гибкостью и экономичностью, все больше приложений переносится из частных инфраструктур в облачные центры обработки данных. Благодаря правильному распределению физических ресурсов в облаках, например центрального процессора, памяти, хранилища и сетевых ресурсов, между различными облачными приложениями вычислительные ресурсы используются эффективно, а затраты на развертывание и эксплуатацию приложений снижаются. Тем не менее, экономическая эффективность развертывания приложений в облаках является одной из самых больших проблем для поставщики услуг и операторы облачных вычислений. Для достижения этой цели были широко изучены вопросы распределения ресурсов и управления ими для приложений, развернутых в рамках различных архитектур или парадигм.

Для поддержки эластичных вычислений в облаке с многоуровневой структурой ресурсы можно масштабировать, регулируя объем физических ресурсов , назначенных каждому развернутому приложению (Liu et al., 2015; Wei et al., 2010). Существует два способа масштабирования ресурсов. Один из них заключается в добавлении дополнительных виртуальных машин (ВМ) (Jiang et al., 2013), которые называются горизонтальное масштабирование. Другой способ заключается в выделении большего количества ресурсов развернутым виртуальным машинам (Shi et al., 2016), что называется вертикальным масштабированием.

Однако развертывание горизонтального масштабирования может занять десятки секунд Виртуальные машины или разбудить физическую машину (PM). Для вертикального масштабирования используются вспомогательные методы, например частота динамического напряжения Масштабирование (DVFS) требует дополнительной поддержки как со стороны хост-операционной системы (OS), так и гостевой ОС, что неизбежно приведет к дополнительной задержке (Baccarelli et al., 2015; Shojafar et al., 2016).

Изучаются различные архитектуры, парадигмы для ускорения процедур и оптимизации затрат на разработку и развертывание приложений в облаках. Появляющаяся виртуализация на основе контейнеров, Docker, в частности, становится альтернативой для развертывания операционной системы приложений, а также поддерживающих библиотек, в то время как традиционные виртуальные машины на базе гипервизора предлагают абстракцию только на аппаратном уровне.

Тем временем архитектура микросервисов (Мартин и Джеймс) разъединяет сложные приложения превращаются в легкие и слабо связанные компоненты. Каждый компонент независимо выполняет микросервис и может быть заменен или обновлен без участия других компонентов. Появление контейнерной и микросервисной архитектуры дает возможность повысить масштабируемость и эластичность разработки приложений . Тем не менее, из-за присущих им различий существующие исследования и механизмы распределения ресурсов для гипервизора на основе Виртуальная машина и монолитная архитектура не могут быть непосредственно применены для использования ресурсов распределение с использованием контейнеров Docker и приложений на основе микросервисов.

Была проделана определенная работа по использованию контейнеров для практического развертывания приложений на основе микросервисов. Чжоу и др. (2018) изучали проблему планирования микросервисов с целью получения максимального дохода от развертывания микросервисов. Guerrero et al. (2018) разработали генетические подходы для определения объема ресурсов, выделяемых каждому микросервису, и способов эффективного масштабирования при изменении рабочей нагрузки. Фацио и др. (2016) обобщили трудности планирования и управления ресурсами при развертывании микросервисов с динамическими запросами пользователей и разнородными настройками в облаках. Однако в большинстве существующих работ не учитывались особенности контейнеров, например, динамическое масштабирование ресурсов с высокой степенью детализации, наслоение изображений и повторное использование библиотек, что повлияло бы на эффективность использования ресурсов и стоимость развертывания приложений.

Мы обратили внимание на важность учета особенностей Docker контейнеры и разработал предварительный подход к динамическому распределению ресурсов для приложений в центрах обработки данных с использованием контейнера docker (Гуань и др., 2017). Тем не менее (Guan et al., 2017) сосредоточились на

особенностях контейнеров Docker, но предположили, что приложения развертываются в монолитной архитектуре, так что контейнеры для одного приложения являются однородными. Ограничения монолитной архитектуры,

такие как неэффективность обновления и доставки, заставляют нас изучить вопрос о распределении ресурсов для развертывания приложений на основе микросервисов.

Мы исследуем как возможности, так и проблемы при развертывании

облачных приложений с использованием контейнеров Docker в рамках микросервисной архитектуры. В частности, мы рассматриваем следующие уникальные возможности для

развертывания приложений с контейнерами Docker в рамках микросервисной

архитектуры:

∙ Каждое приложение может состоять из разнородных микросервисов, каждый

из которых требует разного объема ресурсов и поддерживающих библиотек.

∙ Количество и вместимость контейнеров определяются адаптивно

, исходя не только из требований приложений, но и из имеющихся ресурсов в PMs.

∙ Стоимость развертывания контейнеров связана с доступными вспомогательными

библиотеками в PMs и необходимыми библиотеками приложений.

∙ Функции управления ресурсами и выполнения приложений разделены, и управление ресурсами осуществляется распределенным

образом.

Учитывая вышеупомянутые особенности, мы стремимся оптимизировать

развертывание приложений на основе микросервисов с помощью контейнеров Docker путем минимизации затрат на развертывание приложений, а также эксплуатационных расходов. Решая проблему развертывания приложений, мы

цель - ответить на следующие вопросы: 1) Где разместить контейнерные микросервисы для каждого приложения, чтобы свести к минимуму затраты на развертывание и

эксплуатацию? 2) Какой объем ресурсов,

включая вычислительные ресурсы и сетевые ресурсы, был бы

назначен каждому контейнеру, не вступая в противоречие с ограничениями ресурсов и удовлетворяя требованиям пользователей? Кроме того, ожидается, что решение для развертывания приложений, применимое в крупномасштабных центрах обработки данных с изменяющимся объемом рабочих нагрузок,

будет масштабируемым и адаптивным к изменениям.

Чтобы решить проблему развертывания приложений на практике,

мы сначала предлагаем новый фреймворк под названием ADMD, в котором ресурс, назначенный каждому приложению, может быть адаптивно настроен контроллером

микросервиса для приложения распределенным образом.

Затем мы разрабатываем экономичный и масштабируемый алгоритм для каждого контроллера микросервиса, чтобы определить развертывание исполняемых контейнеров и назначение задач. Основной вклад этого документа

кратко изложен ниже:

∙ Мы математически моделируем проблему развертывания приложений на основе микросервисов, чтобы минимизировать общие затраты, учитывая особенности

Принимается во внимание контейнер Docker.

∙ Для решения задачи представлена масштабируемая платформа, которая могла

бы динамически корректировать объем ресурсов, выделяемых каждому приложению, в зависимости от его требований и статуса.

∙ Что касается трудностей, связанных с проблемой развертывания приложения, мы разбиваем исходную проблему на подзадачи небольшого размера, которые могли бы быть независимо решены в предлагаемой платформе распределенным образом.

Остальная часть этого документа организована следующим образом. В разделе 2 кратко описывается виртуализация на основе контейнеров и архитектура микросервисов.

В разделе 3 кратко рассматриваются связанные технологии, включая соответствующие исследования, конкретно в контексте распределения ресурсов в облачных

центрах обработки данных. В разделе 4 мы формулируем стоимость развертывания приложений с использованием микросервисных приложений и контейнеров Docker в качестве задачи оптимизации. Из-за сложности этой задачи оптимизации мы

разложите его на подзадачи и решайте их с помощью масштабируемой структуры и эффективного для коммуникации алгоритма. В разделе 5 демонстрируется платформа развертывания приложений на основе микросервисов, а

в разделе 6 подробно описываются алгоритмы распределения ресурсов.

Оценки приведены в разделе 7, а заключительные замечания

кратко изложены в разделе 8.

Литература 3

Описание

При проектировании роботов широко используется Фреймворк ROS. В качестве операционной системы использую преимущественно Linux Ubuntu (реже Windows или MacOS). Стоит отметить, что существует разная версионность фреймворка ROS. К примеру, ROS 1 Noetic должна соответствовать версия Linux Ubuntu от 18.0 до 20.04, а ROS 2 Humble должна соответствовать версия 22.04. Что делать в случаях, когда необходимо взаимодействовать сразу с несколькими версиями ROS? Можно, конечно, иметь множество компьютеров или виртуальных машин с разными операционными системами, однако такой подход не рациональный, ведь хотелось бы иметь централизованную систему для быстрого и удобного доступа к нужному проекту, независимости от операционной системы. Проблему может решить Docker.

Контейнеризация является отличной альтернативой аппаратной виртуализации. Все процессы в ней протекают на уровне операционной системы, что позволяет существенно экономить ресурсы и увеличивать эффективность работы с приложениями. Одним из наиболее популярных инструментов для программной виртуализации является Docker — автоматизированное средство управления виртуальными контейнерами. Он решает множество задач, связанных с созданием контейнеров, размещением в них приложений, управлением процессами, а также тестированием ПО и его отдельных компонентов.

Docker (Докер) — программное обеспечение с открытым исходным кодом, применяемое для разработки, тестирования, доставки и запуска веб-приложений в средах с поддержкой контейнеризации. Он нужен для более эффективного использование системы и ресурсов, быстрого развертывания готовых программных продуктов, а также для их масштабирования и переноса в другие среды с гарантированным сохранением стабильной работы.

Docker-контейнеры работают в разных средах: локальном центре обработки информации, облаке, персональных компьютерах и т. д.

Преимущества использования Docker:

1. Минимальное потребление ресурсов — контейнеры не виртуализируют всю операционную систему (ОС), а используют ядро хоста и изолируют программу на уровне процесса. Последний потребляет намного меньше ресурсов локального компьютера, чем виртуальная машина.
2. Скоростное развертывание — вспомогательные компоненты можно не устанавливать, а использовать уже готовые docker-образы (шаблоны). Например, не имеет смысла постоянно устанавливать и настраивать Linux Ubuntu. Достаточно 1 раз ее инсталлировать, создать образ и постоянно использовать, лишь обновляя версию при необходимости.
3. Удобное скрытие процессов — для каждого контейнера можно использовать разные методы обработки данных, скрывая фоновые процессы.
4. Работа с небезопасным кодом — технология изоляции контейнеров позволяет запускать любой код без вреда для ОС.
5. Простое масштабирование — любой проект можно расширить, внедрив новые контейнеры.
6. Удобный запуск — приложение, находящееся внутри контейнера, можно запустить на любом docker-хосте.
7. Оптимизация файловой системы — образ состоит из слоев, которые позволяют очень эффективно использовать файловую систему.

Наброски действий:

подготовить докер, скачать икслаунч, установить WSL.

Странички на имиджы контейнеров рос: osrf/ros (хороший), ros (базовый).

Установить (docker pull osrf/ros:humble-desktop-full) в командной строке

Можно настроить докер файл