Тема работы:

Облачный компилятор на основе микросервисной архитектуры и оркестрации контейнеров

Состав коллектива:

1. Пазников Алексей Александрович, к.т.н., с.н.с. СПбГЭТУ «ЛЭТИ», руководитель

2. Хейдари Сайед Моид, аспирант. СПбГЭТУ «ЛЭТИ», исполнитель

Информация о гранте:

РНФ, проект № 22-21-00686 «Алгоритмы и программные средства оптимизации выполнения параллельных программ в модели удаленного доступа к памяти», руководитель – Пазников А.А., 2022-2023

Научное содержание работы:

1. Постановка задачи:

1. Разработать облачный компилятор с функциональностью управления учетными записями, командной строкой (CLI) и оптимизацией.
2. Предложить архитектурное решение облачного компилятора, обеспечивающее масштабируемость, надежность и безопасность.
3. Обработать большое количество запросов на компиляцию одновременно и эффективно распределить нагрузку на серверы.
4. Исследовать новые возможности для интеллектуальных советов и обнаружения ошибок в компиляторах.
5. Провести экспериментальное исследование, демонстрирующее возможность компилировать большое количество запросов одновременно на кластере.

Мы стремимся создать облачный компилятор, который обладает высокой производительностью, гибкостью и эффективностью обработки запросов на компиляцию.

2. Современное состояние проблемы:

В работе [1] предложен подход к развертыванию компилятора в частном облаке, находящемся внутри внутренней сети компании. Авторы предложили архитектуру, которая получает код от клиента, компилирует его и создает исполняемый файл с прямой ссылкой. Однако отсутствие возможности хранения исполняемых файлов на сервере третьей стороны и отсутствие выбора пользователем места хранения исполняемых файлов или истории компиляции может вызвать проблемы безопасности и конфиденциальности.

Авторы работы [2] предложили архитектуру компилятора и модель развертывания. Их подход основан на использовании одного монолитного компилятора на сервере без разделения на модули. Авторы предложили алгоритм, который распределяет запросы на сервера по приоритету. При отсутствии свободных серверов пользователь ожидает их освобождения. В подходе отсутствует механизм вертикального масштабирования, и серверы статически выделяются для компилятора. Такой подход полностью монолитный и не предусматривает разделение компилятора на модули.

В работах [3, 4] была представлена теория об использовании общего пула настраиваемых вычислительных ресурсов в частном облаке на основе Ubuntu Enterprise Cloud (UEC). Модель предлагает предоставление услуг ограниченному числу клиентов и распределение ресурсов в гетерогенной среде. Авторы предоставили ресурсы облачной вычислительной системы (CCS) с многоступенчатыми устройствами.

В работе [5] М. Пабита, Т. Селвакумар и С. П. Деви также предложили компилятор в частном облаке на основе Linux-окружения. Хотя компилятор работает в облачной инфраструктуре, он по-прежнему использует тот же монолитный подход без масштабируемости и высокой доступности. Кроме того, он доступен только для ограниченного числа пользователей и не предоставляет системы управления репликацией. Полная остановка системы происходит, если сервис компилятора перестает отвечать по любой причине.

Авторы работы [6] представили ключевые факторы, которые следует учитывать при разработке методов автомасштабирования. Они предложили метод настройки автомасштабирования контейнеризованных приложений при наличии предсказуемой нагрузки. В своей работе они также рассмотрели различные интервалы адаптации вместо фиксированного периода, предложенного Kubernetes. Авторы также предложили алгоритм с постоянной сохранения для автомасштабирования.

Авторы работы [7] исследовали некоторые компиляторы и оценили сгенерированный ими код с точки зрения архитектуры набора команд (ISA). Они рассмотрели несколько параметров сгенерированного кода, включая динамическое число инструкций, производительность, размер сгенерированного кода, энергопотребление и время выполнения. Выбор компилятора для разработки системы является очень важным для достижения наивысшей производительности.

1. Aamir, N.A.; Patil, S.; Navada, A.; Peshave, A.; Borole, V. Online C/C++ compiler using cloud computing. In Proceedings of the International Conference on Multimedia Technology, Hangzhou, China, 26-28 July 2011; pp. 3591-3594.
2. Chandan, B.; Anirban, K.; Rana, D. SaaS Oriented Generic Cloud Compiler. Procedia Technol. 2013, 10, 253-261.
3. Zhang, C.; Green, R.; Alam, M. Reliability and Utilization Evaluation of a Cloud Computing System Allowing Partial Failures. In Proceedings of the IEEE 7th International Conference on Cloud Computing, Anchorage, AK, USA, 27 June-2 July 2014; pp. 936-937.
4. Boyer, F.; Etchevers, X.; de plama, N.; Tao, X. Poster: A Declarative Approach for Updating Distributed Microservices. In Proceedings of the IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion), New York, NY, USA, 27 May-3 June 2018; pp. 392-393.
5. Pabitha, M.; Selvakumar, T.; Punitha, D.S. An Effective C, C++, PHP, Perl, Ruby, Python Compiler using Cloud Computing. Int. J. Comput. Appl. 2013, 69, 20-25.
6. Taherizadeh, S.; Grobelnik, M. Key influencing factors of the Kubernetes auto-scaler for computing-intensive microservice-native cloud-based applications. J. Adv. Eng. Softw. 2020, 140, 102734.
7. Park, C.; Han, M.; Lee, H.; Kim, S.W. Performance comparison of GCC and LLVM on the EISC processor. In Proceedings of the International Conference on Electronics, Information and Communications (ICEIC), Kota Kinabalu, Malaysia, 15-18 January 2014; pp. 1-2.

3. Подробное описание работы, включая используемые алгоритмы:

П В предложенном подходе мы представляем фазы компиляции в виде отдельных микросервисов и включаем другие микросервисы, такие как связующий модуль, ассемблер, советник по коду, контроллер базы данных и менеджер пользователей.

Каждая фаза компиляции выполняется отдельно как независимый сервис. Выход каждого микросервиса передается в микросервис Kafka (платформа распределенного потокового событийного вещания). Упомянутая архитектура сосредоточена на технологии

оркестрации контейнеров Google Kubernetes. Kubernetes - это проект с открытым исходным кодом, предоставляющий оркестрацию контейнеров для создания многоузловых облачных кластеров. С помощью этой технологии мы развертываем микросервисы как независимые контейнеры, используя Docker в качестве платформы для контейнеризации. С помощью этой модели мы можем иметь сотни или даже тысячи репликаций контейнеров (подов).

Каждый микросервис может иметь много копий одного пода через систему под названием "набор реплик". Если один под прекращает работу, есть другие репликации этого же пода, чтобы обрабатывать новые входящие запросы. И другой сервис под названием "Kubelet" создает новый под, чтобы заменить остановившийся. Эти механизмы помогают системе повысить доступность всего оркестра и снизить время простоя.

В монолитной модели текущих компиляторов приложение тесно связано, и все уровни приложения доступны пользователю как узкое место. Таким образом, нет необходимости во взаимной связи между сервисами

В случае архитектуры микросервисов нам нужно разделить приложение на несколько сервисов, и каждый сервис имеет свое собственное окружение хранения. В нашей архитектуре мы используем брокер Kafka на основе модели "производитель-потребитель" в качестве технологии обмена сообщениями между микросервисами, чтобы каждый сервис имел полную связь с другими сервисами

Вместо передачи данных между различными модулями компилятора в общей памяти мы передаем их через сетевой протокол (TCP/UDP или HTTP/HTTPS). Для каждого сценария обмена сообщениями у нас есть определенный шаблон сообщений.

Кроме того, мы разделяем front-end и back-end компилятора на два разных пространства имен в облачной среде. Каждое пространство имен имеет свою собственную ответственность, и результаты первого пространства имен передаются в следующее пространство имен через брокер Kafka. Внутренне доступный сервис с типом IP кластера (сервис, доступный внутри) сопоставлен с определенным IP и предназначен для группы репликаций подов. Каждое развертывание может обслуживать большое количество пользователей.

В нашей архитектуре микросервисов взаимодействие между сервисами играет важную роль в производительности. Поэтому в соответствии с нашими требованиями мы должны выбрать правильный подход к взаимодействию между сервисами.

Мы используем Nginx в качестве обратного прокси при реализации Kubernetes. Ingress выступает в роли балансировщика нагрузки на переднем плане кластера. Первым назначением входящих запросов будет наш шлюз API, который принимает запросы, проходит через уровень безопасности и распределяет их между микросервисами.

В данной статье мы используем LLVM в качестве набора модульных и многократно используемых технологий компилятора и инструментов для нашей реализации и экспериментов. Для разработки такой системы нам необходимо иметь передний слой для каждого инструмента LLVM (инфраструктура компилятора в виде набора модульных и многократно используемых технологий компилятора и инструментов), который будет предоставлять услуги конечному пользователю в протоколе HTTP. Предоставляя API для каждой функции, слой шлюза нашей реализации микросервисов делает командный запрос на соответствующий инструмент LLVM в фоновом режиме и отправляет результат в качестве HTTP-ответа запрашивающему сервису.

LLVM предоставляет несколько инструментов для каждой фазы компиляции. Для этой цели мы можем собирать каждую часть LLVM с соответствующими объектными файлами и общими библиотеками

4. Полученные результаты

В данном исследовании значительная часть общей производительности облачного сервера зависит от скорости хранения и скорости чтения-записи. Для проведения экспериментов с одним главным узлом и двумя рабочими узлами мы использовали тестировочный инструмент Fio, предназначенный для оценки производительности ввода-вывода и проведения стресс-тестов на различных платформах

Кроме тестирования с помощью инструмента Fio, мы также оценивали время компиляции программы умножения матрицы 8 × 8, написанной на C++ с использованием компилятора LLVM/Clang. Мы сравнивали время компиляции для разного числа рабочих узлов, присоединенных к главному узлу в кластере. Согласно эксперименту, мы компилировали код с оптимизацией уровня 3 и активированным советником по коду. Затем мы сравнивали время, затраченное на отправку кода на сервер и получение результата. Как показано на графике, увеличение числа рабочих узлов приводит к соответствующему уменьшению времени компиляции.

Эксперимент проводился с 10 повторениями для каждого развертывания в кластере. Результаты представлены на рисунке 1.

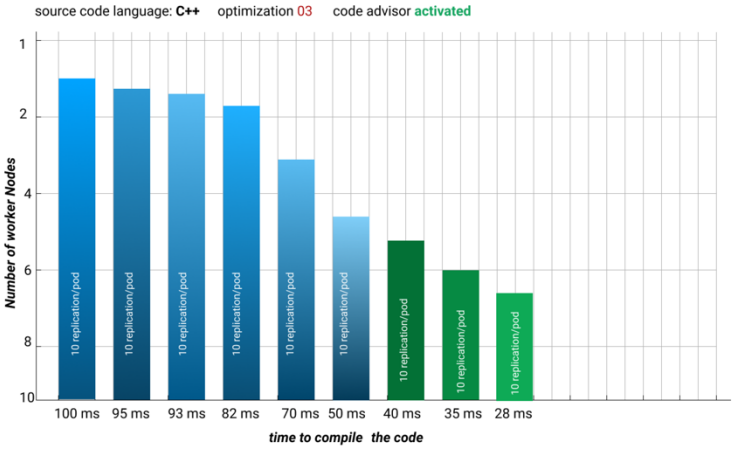

Рис. 1

Другой экспериментальный результат представлен на рисунке 2. Увеличение числа рабочих узлов в кластере приводит к росту количества одновременных запросов и числа одновременных пользователей, которых сервер может обрабатывать. Уменьшается время простоя сервера, а скорость компиляции увеличивается.

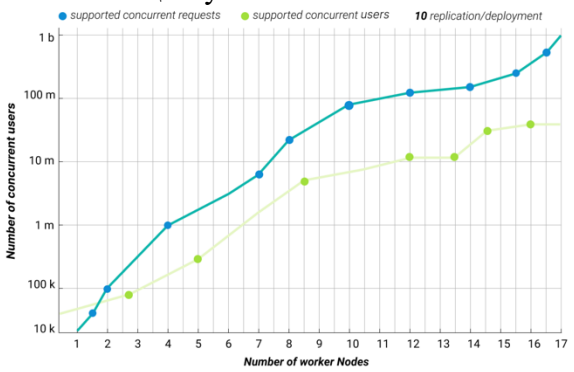

Рис. 2

5. Эффект от использования кластера в достижении целей работы:

Использование кластера при облачной компиляции позволяет достичь следующих эффектов:

1. Ускорение процесса компиляции: Распределение фаз компиляции на отдельные серверы позволяет параллельно обрабатывать большое количество запросов, что приводит к сокращению времени, необходимого для компиляции проектов.

2. Масштабируемость: Кластерная архитектура позволяет гибко масштабировать систему, чтобы справиться с повышенной нагрузкой и обрабатывать большое количество запросов одновременно.

3. Надежность: Использование кластера обеспечивает высокую доступность и отказоустойчивость системы компиляции. В случае сбоя одного сервера, другие серверы в кластере могут продолжить обработку запросов.

В целом, использование кластера в облачной компиляции повышает эффективность и производительность процесса компиляции, улучшает доступность и обеспечивает безопасность и удобство использования для пользователей.

Перечень публикаций, содержащих результаты работы

Heidari S. M., Paznikov A. A. Multipurpose Cloud-Based Compiler Based on Microservice Architecture and Container Orchestration // Symmetry. – 2022. – V. 14. – №. 9. – P. 1818.