МОУ ”Лицей №26”

Разработка распределенной системы онлайн-судьи

Разработал: Артём Башкирев
Преподаватель: Комиссарова Елена Сергеевна

Подольск
2024

Содержание

1 Введение

1.1 Мотивация

Нынешние платформы для проведения соревнований по спортивному программированию, также используемые для вступительных экзаменов, могут уходить в простой в периоды высокой пользовательской нагрузки, что приводит к значительным прерываниям пользователей и потенциально влияет на достоверность оценок. Такие сбои могут привести к потере времени на соревновании, усилению беспокойства и упущенным возможностям для участников.

1.2 Цели и задачи

Цель 1: Разработать систему онлайн-судьи.
Задачи:

- Определить техническую архитектуру системы, включая конкретные технологии, которые будут использоваться.
- Разработать и внедрить различные компоненты системы.
- Провести интеграцию различных компонентов в целостную систему и протестировать в различных условиях.

Цель 2: Оценить производительность и масштабируемость системы.
Задачи:

- Разработать способы измерения производительности и масштабируемости.
- Провести эксперименты по сравнению производительности с существующими системами онлайн-судейства.
- Проанализировать результаты экспериментов и определить области для улучшения.

1.3 Целевая аудитория

Пользователи

- Получают преимущества от бесперебойной работы и быстрого получения результатов.
- Организаторы получают повышенную надежность и поддержку большого количества участников.

Разработчики и администраторы

- Ценители технологий с открытым исходным кодом, модульной архитектуры и распределенных технологий.
- Используют хорошо документированный код и ресурсы для легкого развертывания и настройки.

2 Техническая архитектура

2.1 Обзор проектирования системы

Подход к микросервисам:
Платформа использует архитектуру микросервисов, в которой отдельные службы выполняют отдельные функции:

- **Служба судьи:** Получает материалы, проверяет их формат, выбирает подходящий контейнер и инициирует распределенное выполнение.
- **Служба прогонщиков:** Выполняет тестовые примеры в изолированных контейнерах и генерирует вердикты.
- **Агрегатор вердиктов:** Собирает вердикты от прогонщиков, выполняет дополнительный анализ и сохраняет окончательные результаты.
- **Сервис пользовательского интерфейса:** Взаимодействует с пользователями, обрабатывает аутентификацию, отображает статус отправки и предоставляет обратную связь.
- **Сервис управления данными:** Управляет доступом и взаимодействием с распределенными базами данных (Redis, MySQL, MongoDB).

Каждый сервис работает независимо, взаимодействуя с другими через определенные API. Это позволяет обеспечить:

- **Масштабируемость:** Отдельные сервисы можно масштабировать горизонтально, добавляя новые экземпляры в зависимости от спроса.
- **Управляемость:** Маленькие, сфокусированные сервисы легче разрабатывать, тестировать и обновлять.
- **Устойчивость к сбоям:** Сбои в работе сервисов изолированы, что минимизирует влияние на всю систему.

Механизмы связи: Система использует различные механизмы связи в зависимости от контекста:

- **RESTful API:** Используется для обмена структурированными данными между службами, например для получения представлений или хранения вердиктов.
- **Websockets:** Смогут обеспечить двунаправленную связь между клиентами и сервером в реальном времени, предоставляя оперативные обновления статуса подачи.
- **Redis Pub/Sub:** Обеспечивает асинхронную передачу вердиктов между сервисами: прогонщик публикует вердикт по завершении работы, а агрегатор вердиктов подписывается на его получение.

Эти механизмы обеспечивают эффективную и масштабируемую связь в распределенной системе.
Дополнительные соображения:

- **Обнаружение сервисов:** Механизмы Kubernetes могут быть использованы для динамического распределенного исполнения сервисов.
- **Управление конфигурацией:** Такие инструменты, как Ansible, позволяют управлять конфигурацией распределенных сервисов.
- **Логгинг и мониторинг:** Распределенные сервисы протоколирования, такие как Atlassian Statuspage, обеспечивают централизованное понимание поведения системы и потенциальных проблем.

2.2 Контайнеризация

Система предполагает использование контайнеров. Это обеспечивает:

- **Согласованность:** Тесты и код выполняются на любой системе, независимо её расположения.
- **Изоляцию:** Потенциально деструктивный для системы прогонщика код предполагает невозможность выхода за пределы песочницы.
- **Скорость:** Предварительно созданные образы контайнеров исключают длительную сборку и направлены на быстрое получение результатов.

2.2.1 Выделенный реестр контайнеров

Предполагаемый Container Registry для Docker предполагает автоматическую сборку и обновления образов с инструментами CI/CD. Ключевыми в представимом решении станут системы контроля версий утилит сборки и языков программирования внутри контайнеров чтобы обеспечить честность соревнований.

3 Взаимосвязь архитектуры

3.1 Высокоуровневый обзор

Высокоуровневое архитектурное представление системы сфокусировано на взаимодействии с пользователями и потоке данных, связанных с данными, отправляемыми пользователями.

Управление пользователями:

- Пользователи регистрируются в системе, предоставляя основную информацию, такую как имя пользователя и электронная почта. Дополнительные сведения о профиле, такие как имя, местоположение и принадлежность, фиксируются в отдельной таблице.
- Система сможет различать роли пользователей с помощью флагов (is_staff), что позволяет использовать механизмы гранулярного контроля доступа.

Обработка решений:

- Пользователи отправляют данные по установленным каналам, и система записывает ссылку на исходный код и временную метку отправки для каждого экземпляра.
- Центральная система очередей, работающая на базе Celery, эффективно управляет обработкой поступающих решений, обеспечивая оптимальное использование ресурсов.
- Предоставленные данные надежно хранятся в облачном хранилище, используя его масштабируем и гибкость для работы с различными форматами данных.

Сохранение данных:

- Данные о пользователях, включая информацию о профиле и историю активности, хранятся в базе данных MySQL. Эта реляционная структура базы данных позволяет эффективно выполнять запросы и управлять структурированными данными.
- Данные об отправке решений хранятся в базе данных MongoDB, что позволяет использовать ее масштабируемость и гибкость при работе с потенциально большими и неоднородными форматами данных.

3.2 Обзор пути обработки решений

3.2.1 Выбор контайнера и распределенное выполнение

Перед выполнением система применяет стратегии тщательного выбора контейнеров (в первой версии производится силами пользователя):

- **Подбор языка и фреймворка:** система использует специальный реестр контейнеров, в котором хранятся готовые образы для популярных языков программирования и фреймворков. На основе идентификации языка и фреймворка представленного кода из реестра выбирается наиболее подходящий контейнер.
- **Управление версиями:** Реестр поддерживает несколько версий контейнеров для каждого языка/фреймворка, что позволяет организаторам определять желаемую среду выполнения для конкретных соревнований или упражнений. Такая гибкость позволяет удовлетворить различные требования и предпочтения.
- **Соображения безопасности:** Механизмы ”песочницы” в контейнерах дополнительно повышают безопасность, ограничивая доступ к файловой системе и сетевое взаимодействие.

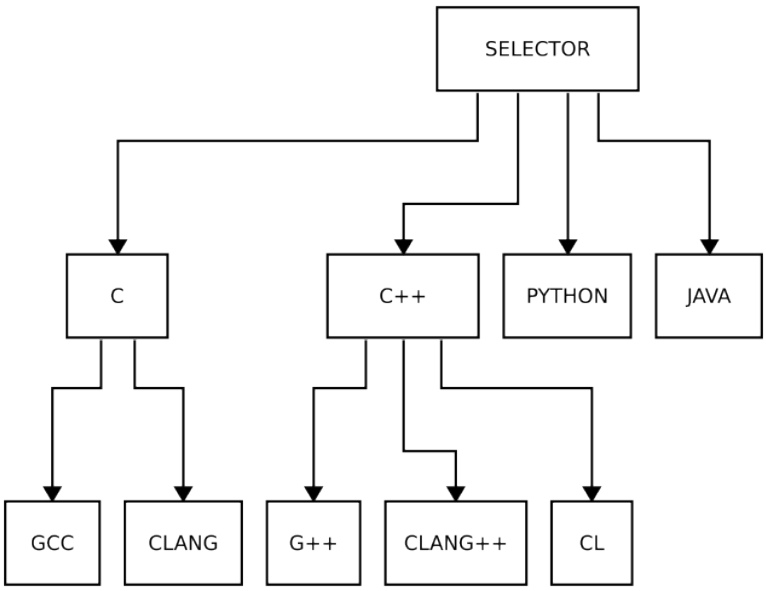

Рис. 1: Схема выбора контайнера исполнения

3.2.2 Выполнение тестов и возврат вердиктов

После выбора оптимального контейнера начинается фаза выполнения:

- **Подготовка контейнера:** Эфемерный экземпляр контейнера запускается с использованием выбранного образа. В среду контейнера вводятся необходимые параметры конфигурации и данные тестовых примеров.
- **Изолированное выполнение:** Представленный код выполняется в контейнере, полностью изолированном от базовой хост-системы. Это гарантирует справедливость и устраняет проблемы совместимости, связанные с различными программными средами.
- **Управление ресурсами:** Система использует методы управления ресурсами для обеспечения эффективного распределения вычислительных ресурсов между запущенными контейнерами. Это позволяет оптимизировать производительность системы и предотвратить нехватку ресурсов, особенно в периоды пиковых нагрузок.
- **Мониторинг выполнения:** *(Отложено до v2)* На протяжении всего этапа выполнения система отслеживает использование ресурсов и состояние контейнеров. Любые аномалии или истощение ресурсов вызывают корректирующие действия, обеспечивая плавное и бесперебойное выполнение.

3.2.3 Агрегация и хранение вердиктов

Поскольку представления выполняются в изолированных контейнерах, формирование вердиктов приобретает первостепенное значение.

Распределенная генерация вердиктов: Каждый прогонщик, выполнив представленный образец кода с тестовыми примерами, генерирует вердикт. Этот вердикт содержит результат

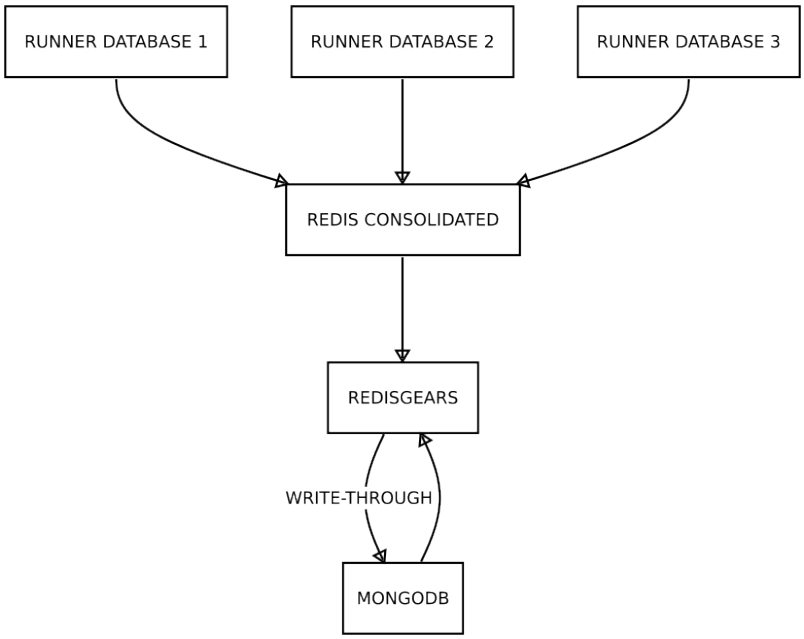

Рис. 2: Схема агрегации вердиктов

(пройден, не пройден, ошибка), а также другие важные данные, такие как время выполнения и использование ресурсов. Система использует метод pub/sub для эффективной передачи вердиктов:

- **Прогонщики как Publishers**: После генерации вердикта каждый исполнитель публикует его в выделенном канале Redis. Данный способ не требует дополнительной инфраструктуры.
- **Агрегатор вердиктов в качестве подписчика**: Redis Consolidated (см. Рис. 3) подписывается на каналы прогонщиков. В режиме реального времени он получает и собирает вердикты, опубликованные отдельными исполнителями.

MongoDB для сохранения вердикта:

Чтобы обеспечить возможность анализа, подробную информацию и возможность пересмотра результатов после соревнований, система использует MongoDB для хранения вердиктов:

- **Структурированное хранилище:** Окончательный вердикт, а также связанные с ним метаданные (идентификатор заявки, временная метка, вердикты отдельных участников) хранятся в коллекции MongoDB.
- **Масштабируемость и гибкость:** Документоориентированность MongoDB способствует эффективному хранению разнообразных данных о вердиктах, а ее масштабируемость позволяет без труда обрабатывать большие объемы вердиктов.
- **Доступ в будущем:** Это постоянное хранилище позволяет организаторам и пользователям ретроспективно получать доступ к вердиктам, анализировать тенденции и, при необходимости, проводить переоценку прошлых представлений.

4 Оценка

Метрики скорости агрегации, задержки и времени отклика имеют первостепенное значение для судьи по коду, существенно влияя на удобство использования и справедливость системы.
Задачи оценки:

- Количественно оценить способность системы справляться с различной нагрузкой при сохранении приемлемого времени отклика и задержки.
- Выявить и устранить потенциальные узкие места в производительности для проактивной оптимизации.

Целевые функции:

- **Скорость агрегации вердиктов:** измерение времени, необходимого агрегатору вердиктов для эффективного сбора и консолидации вердиктов от отдельных исполнителей. Эта метрика напрямую влияет на общее время ожидания пользователями результатов. (Возможно только для разрабатываемой платформы)
- **Задержка обратной связи в реальном времени:** оценка задержки между завершением отправки и предоставлением обратной связи пользователям (если применимо). Минимизация этого времени имеет решающее значение для подобных систем.
- **Время отклика системы:** оценка общей скорости реакции системы на действия пользователей. Сюда входят такие действия, как загрузка заявок, просмотр результатов и взаимодействие с платформой. Быстрое время отклика способствует плавному и бесперебойному взаимодействи с пользователем.

Количественные показатели для разрабатываемой системы:

- **Скорость агрегирования вердиктов:**
  - Среднее и максимальное время агрегации при различных моделируемых пользовательских нагрузках.
  - Анализ распределения времени агрегации для выявления потенциальных выбросов или изменений производительности.
- **Задержка обратной связи в реальном времени:**
  - Средняя и максимальная задержка между завершением отправки и получения вердиктов.
  - Измерение задержки в различных сетевых условиях и географических регионах.
- **Время отклика системы:**
  - Среднее и максимальное время отклика для типичных действий пользователя при разных уровнях нагрузки.
  - Разбивка времени отклика по конкретным компонентам системы (например, интерфейсная часть, серверная часть, база данных) для целевой оптимизации.

Методика нагрузочного тестирования:
Чтобы создать реалистичные сценарии и точно оценить производительность в различных условиях, планируется использовать комбинацию подходов:

- **Инструменты тестирования контролируемой нагрузки:** Использование инструментов Apache JMeter, Locust и K6 для имитации одновременных пользователей, отправляющих решения и получающих вердикты.

- **Контролируемые эксперименты:** Проведение экспериментов, стратегически изменяя количество одновременных пользователей, частоту отправки и сложность компиляции и прогона, чтобы проанализировать их влияние на показатели производительности.
- **Реальное использование:** Привлечение реальных пользователей к контролируемому бета-тестированию.

5 Заключение

5.1 Вывод

Система в своей начальной форме в виде версии 0.0.1 является примером того, как можно исследовать новые концепции. Хотя она еще не охватывает весь спектр предполагаемых функций, эта начальная итерация служит пилотным исследованием распределенных систем судейства кода. В контексте академического проекта разработка способствовала получению обширного опыта, углубляясь в тонкости распределенных архитектур, контайнеризации и их реализации.

Хотя на начальном этапе реализации не удалось получить исчерпывающие количественные данные, проект заложил прочный фундамент для дальнейшего развития и оценки. Несмотря на эти трудности, были получены ценные сведения, которые заложили основу для создания надежной и эффективной платформы для оценки кода.
Полученный опыт в жизненном цикле проекта:

- **Важность детального планирования:** Опыт показал необходимость тщательного планировани сценариев нагрузочного тестирования и контрольных показателей производительности с самого начала.
- **Устранение узких мест в инфраструктуре:** Начальное выявление потенциальных узких мест в производительности во время разработки позволило выбрать архитектурный дизайн для обеспечения масштабируемости и быстродействия.
- **Важность тестирования в реальных условиях:** Проект подчеркнул важность включения тестирования в реальных условиях, когда это возможно.

6 Приложение

6.1 Литература

- Gaurav Sen - System Design: Online Judge for coding contests
- Titus Winters, Tom Manshreck, Hyrum Wright - Software Engineering at Google: Lessons Learned from Programming Over Time
- Martin Fowler - Patterns of Enterprise Application Architecture

Список иллюстраций