Distributed Parser

Пчелка Анна, Шарабарин Михаил

Распределённая система для парсинга веб-страниц

Проект представляет собой распределённую систему для обработки и парсинга веб-страниц, включающую несколько взаимодействующих компонентов.

Основные компоненты

- Координатор: Управление процессом и распределение задач.
- Воркеры: Выполнение заданий по парсингу.
- База данных: Хранение данных о заданиях, результатах и состояниях.

ТЕХНОЛОГИИ:

- Контейнеризация: Docker
- Оркестрация: Kubernetes
- Хранилище данных: Apache Cassandra
- Графический интерфейс: Qt (C++)

АРХИТЕКТУРА РАСПРЕДЕЛЁННОЙ СИСТЕМЫ

Координатор

- Распределяет задачи между воркерами.
- Поддерживает связь с базой данных для получения и обновления статусов задач.

Воркеры

- Парсят веб-страницы на основе задач, полученных от координатора.
- Отправляют результаты в базу данных.

База данных (Apache Cassandra)

- Хранит задания на парсинг, их состояния и результаты.
- Масштабируемое хранилище с высокой доступностью.

Графический интерфейс

Позволяет пользователям регистрироваться, авторизоваться, отправлять задачи и отслеживать их статус.

ТЕХНОЛОГИИ И ИНСТРУМЕНТЫ

- **Docker:** Контейнеризация всех компонентов для унифицированной среды выполнения.
- **Kubernetes:** Управление масштабированием и развертыванием контейнеров.
- **Apache Cassandra:** Децентрализованная база данных для хранения данных с высокой производительностью.
- **Qt (C++):** Разработка кросс-платформенного графического интерфейса.

Особенности:

- Масштабируемость за счёт использования Kubernetes.
- Устойчивость к отказам благодаря Cassandra и распределённой архитектуре.

Пользовательский интерфейс

HWParser

Интерфейс позволяет пользователям:

- Регистрация и авторизация:
  - Ввод имени пользователя и пароля.
  - Сохранение токенов с поддержкой перезагрузки приложения.
- Отправка задач:
  - Ввод описания задачи и загрузка необходимых файлов.
  - Интеграция с API для передачи заданий координатору.
- Отслеживание статуса:
  - Просмотр состояния задач через интерфейс.

Особенности реализации:

- Разработан на основе Qt с использованием классов AuthManager и Profile.
- Локальное сохранение токена через QSettings.

Координатор

- Получает задания от интерфейса или базы данных.
- Делит задачи на подзадачи и отправляет их воркерам.
- Сохраняет прогресс выполнения в базу данных.

Воркеры

- Получают задания от координатора.
- Выполняют HTTP-запросы для загрузки страниц.
- Парсят контент и отправляют результаты обратно в базу данных.

Технические особенности:

- Обмен данными через REST API.
- Мониторинг и управление через Kubernetes.

БАЗА ДАННЫХ

Причины выбора Cassandra:

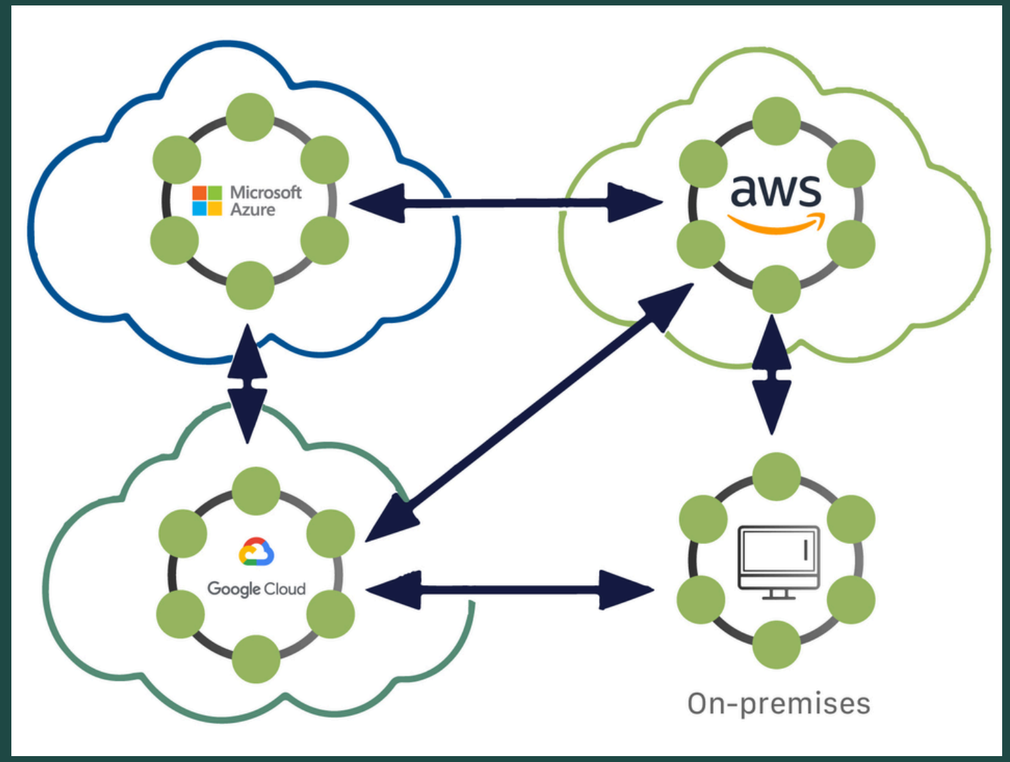
- Высокая доступность и масштабируемость.
- Поддержка распределённой архитектуры.

Типы данных, хранящихся в базе:

- Задания (описания, состояния).
- Результаты парсинга.
- Логи взаимодействий между координатором и воркерами.

Реализация:

- Кластеры Cassandra развернуты через Kubernetes.
- Доступ к данным осуществляется через REST API.



КОНТЕЙНЕРИЗАЦИЯ И ОРКЕСТРАЦИЯ

Контейнеризация с Docker:
Каждый компонент системы упакован в отдельный контейнер.
Унифицированная среда разработки и выполнения.
Оркестрация с Kubernetes:
Развёртывание координатора, воркеров и базы данных в виде подов.
Автоматическое масштабирование воркеров в зависимости от нагрузки.
Балансировка нагрузки и высокая отказоустойчивость.

РЕЗУЛЬТАТЫ И ТЕКУЩИЙ ПРОГРЕСС

Реализовано:

- Графический интерфейс для взаимодействия с системой.
- Логика работы с токенами (авторизация, регистрация).
- Базовая интеграция с REST API для отправки задач.

В процессе:

- Разработка координатора и воркеров для обработки заданий.
- Подключение базы данных Cassandra к основным компонентам.

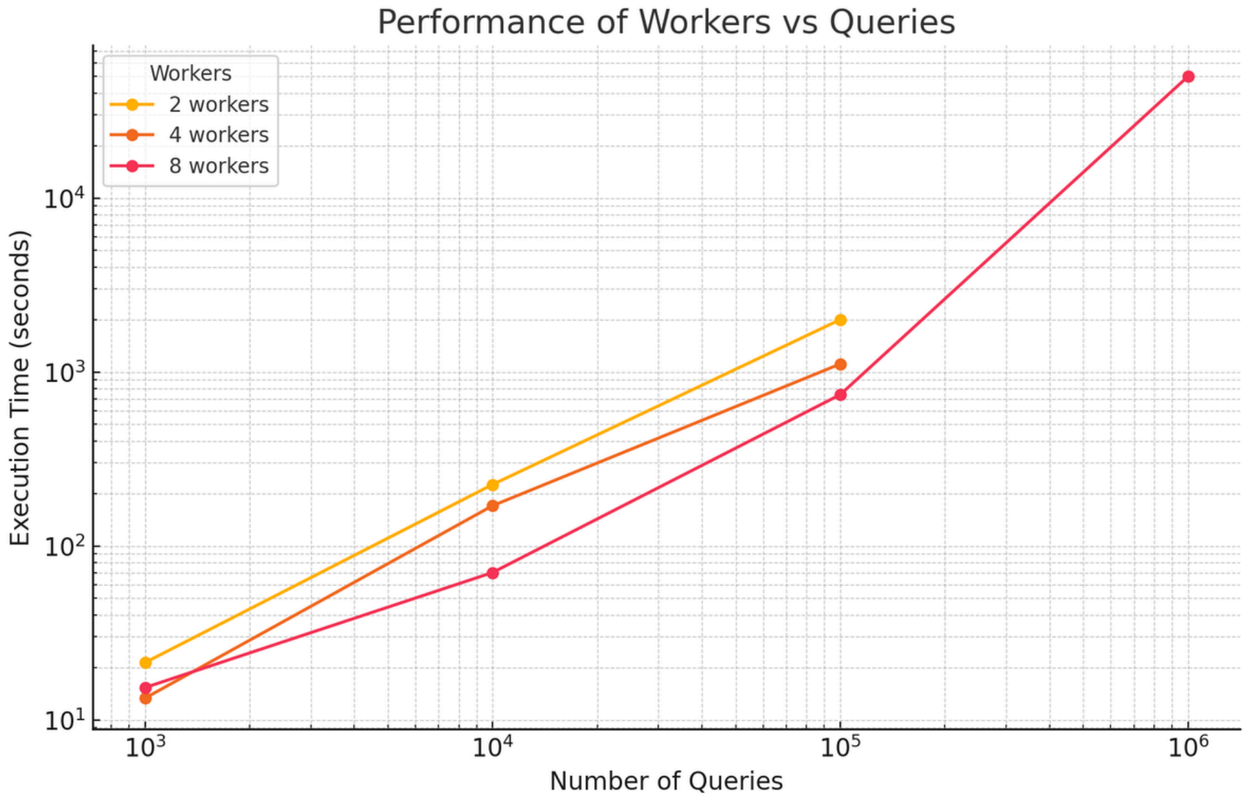
Планы:

- Масштабирование системы и нагрузочное тестирование.
- Расширение функциональности GUI для отображения статуса задач.

Таблица “Зависимость сокрости от количества воркеров”

Запросы/ Воркеры	**2**	**4**	**8**
10^3	21,32	13,31	15,32
10^4	225,43	170,32	70,32
10^5	>2000	18мин32с	12мин21с
10^6	+∞	+∞	50000с
10^7	+∞	+∞	+∞

Испытания проводились на ubuntu server на локальном компьютере в домашней сети 100 Mbps/sec

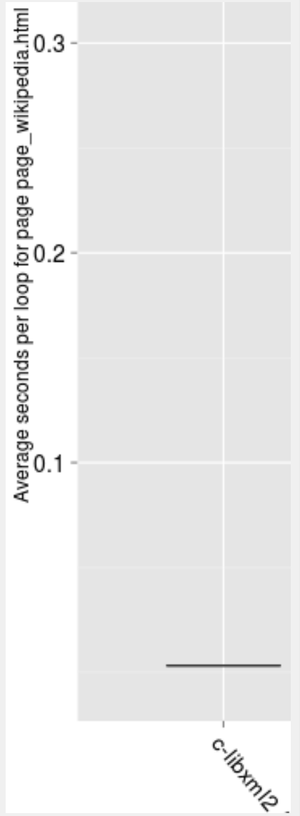


Потребление

Скорости

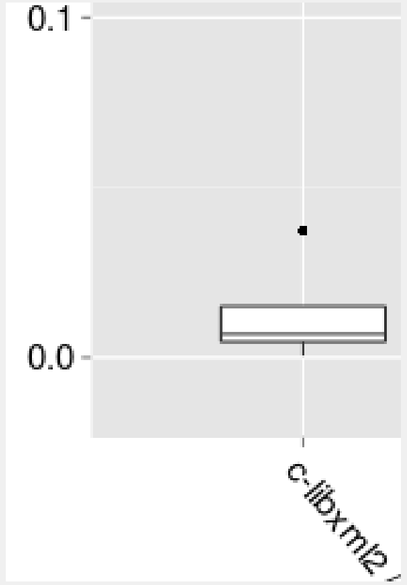
На графике изображена средняя скорость обработки одного сетевого запроса с парсингов кэшированной страницы wikipedia.html
Уточню, что файл не загружен окально, а хранится в кэше dns

```
> subset(res, (file=="page_google.html") & (loops==1000))[ c
    platform                parser   parser.s real.s user.s
6  c-libxml2 libxml2_html_parser.c   2.934295   2.93   2.92
30    erlang     mochiweb_html.erl  13.346997  13.51  13.34
14    nodejs     cheerio_parser.js   5.303000   5.37   5.36
38    nodejs  htmlparser_parser.js   6.686000   6.72   6.71
22    nodejs       jsdom_parser.js  98.288000  98.42  98.31
33      pypy      bsoup3_parser.py  40.779929  40.81  40.62
57      pypy      bsoup4_parser.py 434.215878 434.39 433.91
41      pypy    html5lib_parser.py 361.008080 361.25 360.46
```



Памяти

```
> subset(res, (file=="page_google.html") & (loops==1000))[ c
    platform                parser maximum.RSS
6  c-libxml2 libxml2_html_parser.c       2240
```



Потребление памяти измеренно на одном узле Воркер ноде. По сранению с другими библиотеками, очень круто получилось

```
> subset(res, (file=="page_google.html") & (loo
    platform                parser maximum.RSS
6  c-libxml2 libxml2_html_parser.c       2240
30    erlang     mochiweb_html.erl      21832
14    nodejs     cheerio_parser.js      49972
38    nodejs  htmlparser_parser.js      48740
22    nodejs       jsdom_parser.js     119256
33      pypy      bsoup3_parser.py      61756
57      pypy      bsoup4_parser.py    1701676
41      pypy    html5lib_parser.py    1741944
```

Resident set size (RSS) — это часть памяти (измеряемая в килобайтах), занимаемая процессом, которая находится в основной памяти (RAM).

Спасибо за внимание!