МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Пермский государственный национальный
исследовательский университет

ОТЧЕТ
по дисциплине «Технологии разработки распределенных приложений»
по теме «Разработка распределенного приложения для локальной сети»

Работу выполнили
студенты гр. ЗАОФИТ-4-2016-нб
Ташлыков Данил,
Байрамов Давид,
Редкачев Александр
5 июля2020 г.

Проверил к.ф.-м.н.,
доцент кафедры ПМИ
Деменев Алексей
Геннадьевич
«___» _____ 2020 г.

Пермь, 2020

1 Постановка задачи

Цель: изучение методов коммуникации процессов в сети, а также средств динамического конфигурирования распределенных приложений.

Проверяемые компетенции: способность работы с информацией из различных источников, включая сетевые ресурсы сети Интернет, для решения профессиональных задач; способность применять на практике теоретические основы и общие принципы разработки распределенных систем; уверенное знание теоретических и практических основ построения распределенных баз данных; способность использовать на практике стандарты сетевого взаимодействия компонент распределенной системы.

Задание выполняется в группе (не более трех студентов) или индивидуально. Каждый студент отчитывается по каждому пункту задания индивидуально.

Требования к выполнению работы:

Приложение должно обеспечивать параллельную работу нескольких клиентов и серверов. Дополнительное требование: возможность запуска нескольких серверов на одном компьютере.

Клиентские приложения должны автоматически находить серверы для обслуживания и выполнения заданных функций.

Серверы системы могут выполнять различные функции.

При разрыве сеанса приложения должны автоматически восстанавливать свою работоспособность.

Для хранения данных и доступа к ним применить ADO и/или ADO.NET (или их аналоги).

Приложения должны поддерживать возможность взаимодействия в различных режимах.

Для организации взаимодействия нужно использовать различные средства коммуникации (именованные каналы, мейлслоты, сокеты, MSMQ, .Net Remoting, web-сервисы, WCF-сервисы), сравнив их возможности.

В программе **необходимо** использовать открытые данные. Это могут быть государственные открытые данные любой страны (например, с порталов и http://data.gov.ru/,), иные общедоступные машиночитаемые данные.

По окончании выполнения задания каждая группа студентов должна подготовить отчет и доклад на 7-10 минут. Выступление студентов сопровождается показом презентации, отражающей основные позиции разработки.

Отчет по выполнению задания должен включать:

1. Общее описание приложения. Постановка задачи, введение в предметную область.
2. Архитектура системы. Обоснование выбора данного типа архитектуры распределенного приложения. Алгоритм работы приложения в целом.
3. Архитектура каждого из логических компонент системы (серверы, клиенты, диспетчеры). Подходы к реализации. Алгоритмы работы. Многопоточность, обоснование.
4. Методы коммуникаций компонентов системы (клиент→сервер, сервер→клиент и т.д.). Обоснование выбора этих методов коммуникации.
5. Способ передачи данных (синхронная / асинхронная, однонаправленная / двунаправленная и т.д.). Обоснование.
6. Структура передаваемых данных. Вид протоколов, обоснование выбора.
7. Отказоустойчивость системы. Как система поведет себя, если «исчезнет» один или несколько ее компонент. Что произойдет с системой, если «исчезнувший» компонент будет восстановлен на другом узле сети.
8. Работа с базой данных (если используется). Обоснование.
9. Исходный код приложений с комментариями.

Максимальное количество баллов, которые студент может получить за выполнение работы равно тридцати. Распределение баллов за выполнение работы представлено в следующей таблице:

Критерий оценивания	Оценка
Распределенное приложение спроектировано с учетом особенностей предметной области. Выбрана наиболее подходящая модель распределенной системы. Если используется распределенная база данных, то тиражирование данных имеет подходящий для данной предметной области механизм и архитектуру.	3
Архитектура системы является оптимальной для заданных при разработке критериев. В отчете присутствует обоснование выбора данного типа архитектуры.	3
Приложение обеспечивает параллельную работу нескольких клиентов и серверов, в том числе на одном компьютере. Серверы распределенной системы выполняют различные функции.	2
Приложение является масштабируемым, позволяет добавлять новых участников взаимодействия без переписывания кода и перезапуска приложений.	2
Существует возможность динамического реконфигурирования системы.	4
Для организации взаимодействия компонент распределенной системы используется не менее четырех различных средств коммуникации. В отчете присутствует четкое обоснование выбора средств взаимодействия для каждого конкретного случая.	4
Система является отказоустойчивой. В случае если один и/или несколько компонент системы аварийно завершают свою работу.	2
Распределенное приложение продолжает работать и в случае, если после аварийного завершения некоторого компонента, он восстановлен на другом узле вычислительной сети.	3
Отчет содержит подробное описание архитектуры каждого компонента	3

распределенного приложения.	
В отчете описана структура передаваемых данных, формат сообщений и вид протокола, используемого для этого.	2
В отчете представлено описание способа передачи сообщений при коммуникации компонентов распределенной системы с обоснованием.	2

2 Описание приложения. Введение в предметную область

Предметная область – измерение времени доступа к ресурсам.

В настоящее время все больше и больше передается данных по сети, так же из-за увеличения мощностей появляется возможность передавать больше данных за короткое время, поэтому и количество данных, передаваемых по сети “Интернет” - растет.

Поэтому вопрос об скорости отклика для пользователя становится все серьезнее.

На первый взгляд - разница отклика между, например, Францией и Россией - небольшая. Но как только в ход пойдут тяжеловесные страницы, разница будет огромна.

При проектировании и разработке продукта важно, чтобы разница в отклике для любых пользователей, была минимальна.

Разработанное распределенное приложение было создано для тестирования разницы откликов на интересующие вас апи.

При обращении к нему, вы должны описать запрос, который требуется сделать для замера отклика.

Вам же вернется информация об времени отклика от бота и данные, которые он получил. Для дальнейшей работы эти данные с всей остальной информацией приложение сохранило в базу данных.

3 Архитектура системы

На рисунке 1 изображена архитектура рассматриваемой распределенной системы.

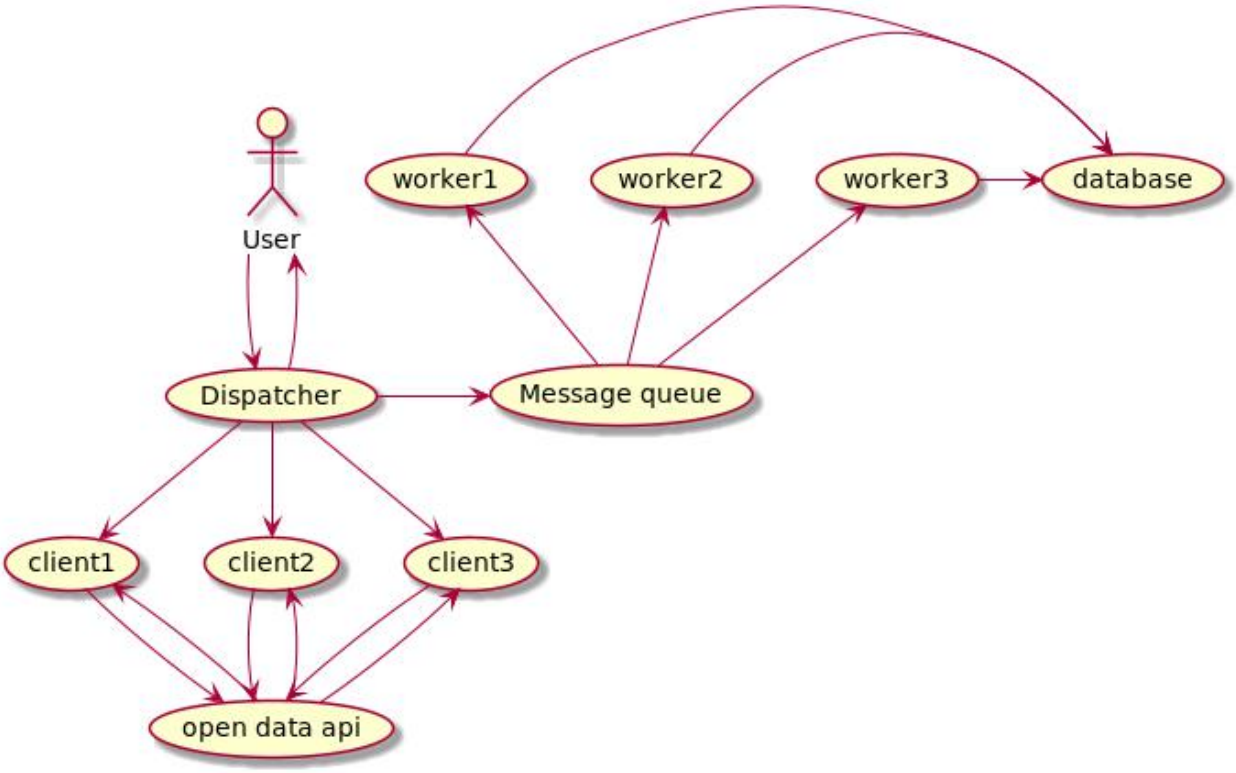


Рисунок 1 – Архитектура распределенной системы

Алгоритм работы данной распределенной системы состоит из следующих шагов:

1. Клиент при включении ищут своего диспетчера и подключаются к нему по websocket

2. Пользователь посылает POST запрос на REST-API диспетчера, в теле запроса указывает параметры, которые требуются для запроса клиентами в формате json.

3. Диспетчер обрабатывает пользовательский запрос, формирует читаемый для клиентов запрос и отправляет каждому в установленное websocket соединение.

4. Клиенты приняв команду от диспетчера, выполняют заданный запрос на внешнее апи с открытыми данными и возвращают ответ с временем запроса диспетчеру через тот же websocket

5. Диспетчер, приняв ответ и дождавшись всех ответов посылает данные в очередь, для сохранения их в базу данных.

6. Слушатели очереди, исполнители, принимают запись и отправляют её в базу данных, используя протокол grpc

7. Так же, диспетчер формирует ответ для пользователя и отправляет его в ответе на пользовательский запрос.

Таким образом пользователь отправил запрос на сервер и в ответ получил все данные от ботов об времени отклика и данных, переданных от внешнего апи.

4 Архитектура клиента

Клиент представляет собой коммуникационный модуль и модуль логики. Модуль логики выполняет запрос на внешнее апи и обрабаывает его, а коммуникационный модуль через wss связывается с диспетчером и передает эту информацию. Саму схему архитектуры клиента можно увидеть на рисунке 2.

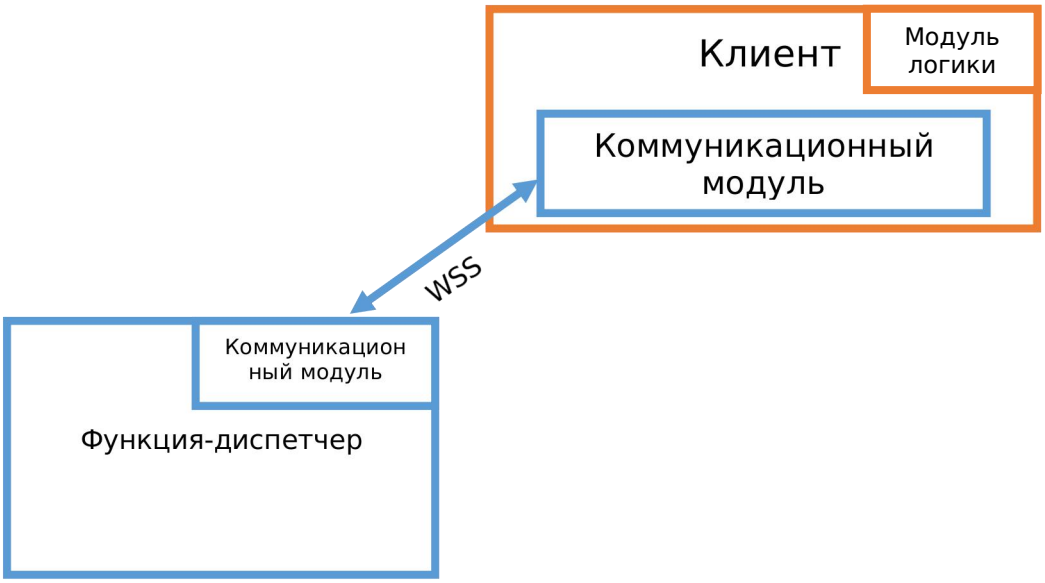


Рисунок 2 – Архитектура клиента

5 Архитектура сервера

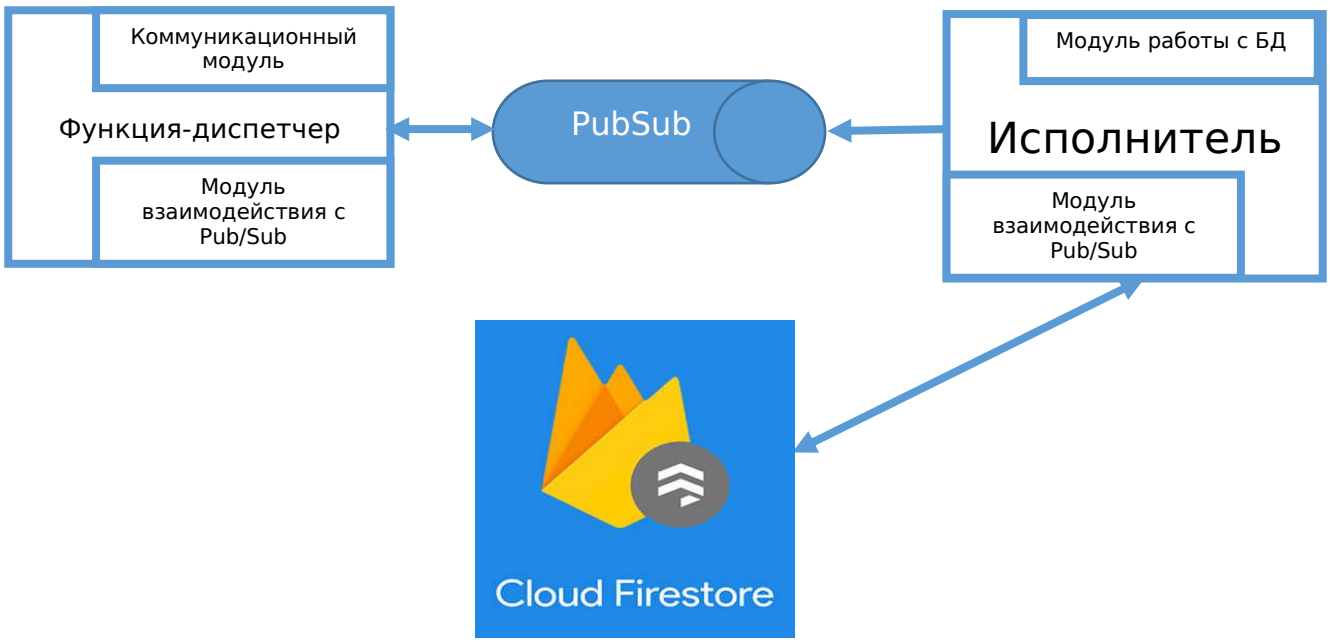
Серверная часть состоит из набора компонентов: диспетчер, очередь сообщений, исполнители и база данных.

Диспетчер содержит в себе 3 коммуникационных модуля,

Один отвечает за общение с пользователями через REST-APIm, один отвечает за обмени данными с клиентами (по протоколу wss, а второй отправляет данные, полученные от клиентов, в очередь сообщений Google Pub/Sub.

Затем, из этой очереди сообщения считывают исполнителями и записываются в базу данных Google Cloud Firestore, из которой они могут быть извлечены для дальнейшего анализа.

Архитектура сервера изображена на рисунке 3.



.

Рисунок 3 – Архитектура сервера

6 Google Cloud Firestore

Для хранения информации была использована облачная NoSQL база данных – Google Cloud FireStore. Взаимодействие БД с сервером происходит за счет gRPC. Схема взаимодействия представлена на рисунке 4.

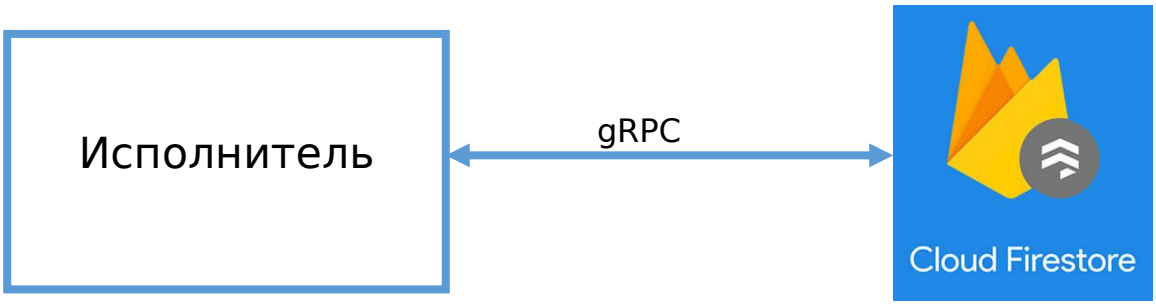


Рисунок 4 – Схема взаимодействия сервера с Cloud Firestore

7 Структура передаваемых данных и работа с БД

Сама БД не имеет жесткой структуры, т.к. используется NoSQL подход. Пример структуры передаваемых данных можно увидеть на рисунке 5.

Рисунок 5 - Пример записи в базе данных

Структура содержит информацию об боте, который её сгенерировал, об запросе, который был выполнен, об информации, которую этот запрос вернул и конечно же время обработки этого запроса и его статус код.

8 Отказоустойчивость

Отказоустойчивость системы гарантируется, прежде всего, делегированием ответственности за решение проблем, связанных с выходом из строя компонентов, на Google Cloud и те средства, которые используются данной платформой для защиты от неожиданного поведения системы. Рассмотрим подробнее отказоустойчивость каждого компонента.

Клиент гарантирует свою корректную работу за счет системы поллинга диспетчера, в случае отсутствия от того ответа, процесс повторяется каждые 10 секунд. При разрыве WS соединения попытки повторное подключения выполняются с тем же интервалом.

Так как диспетчер представляет собой экземпляр Cloud App Engine, то его работоспособность гарантируется платформой Google Cloud. При падении приложения, либо отсутствия связи, контейнер с приложением перезапускается на другом узле, доступном по тому же адресу, что и раньше. Сообщения, отправленные диспетчеру хранятся в Google Pub/Sub и регулируются механизмом подтверждения приема, описанным ниже.

В работе с очередью Google Pub/Sub, отказоустойчивость так же гарантируется платформой Google Cloud и ее встроенными средствами.

Лицензии

Среда разработки:

- Vim: GNU GPL.
- goLang: Creative Commons Arttribution 4.0

Языки программирования:

- Python: Python Software Foundation License (PSFL), GNU
- Go: MIT License

Используемые библиотеки:

- BSD License
- MIT License
- Creative Commons Arttribution 4.0

Используемые python библиотеки

1. Sys
2. Json
3. Pprint
4. Argparse
5. Requests

Данные библиотеки относятся к лицензии **PSFL**

Используемые go бибилиотеки

1. "cloud.google.com/go/firestore"
2. "github.com/jessevdk/go-flags"
3. "github.com/rs/zerolog/log"
4. "github.com/satori/go.uuid"
5. "github.com/danilkastar440/TRRP_LAST/pkg/pubsub"