Примечания

1. В качестве проекта для создания benchmark для обоснования почему какой-либо фреймворк или СУБД были выбраны, может быть использован любой крупный проект на усмотрение студента – ЛР по курсу Web, курсовая по БД или сетям и т.д.
2. Также допустимо написать отдельное приложение, которое может сравнивать например различные СУБД на основании запросов из ранее упомянутых проектов, либо сравнивает два микро-фреймворка для одного языка программирования

Лабораторная работа № 3

Задание

Составить набор сценариев (benchmark) для оценки производительности фреймворка для проекта
Примеры часто рассматриваемых сценариев для оценки производительности можно найти например в сравнении разных популярных web-фреймворков:

- https://github.com/the-benchmarker/website
- https://github.com/TechEmpower/FrameworkBenchmarks

Так же требуется сравнить хотя бы с одной альтернативой выбранному фреймворку

Требования

1. Производительность (не важно – по памяти на сценарий, по времени обработки запроса на сервере, по задержке из-за сериализации объектов, по количеству полных gc-пауз если это jvm, по времени warm-up периода при старте приложения) оценивается статистической величиной – требуется провести хотя бы 100 испытаний и собрать статистику
2. Испытаний будет много, надо добиться равных условий на каждую попытку, то есть следует использовать отдельный докер-образ на каждый прогон, но с одной и той же конфигурацией
3. Измерение времени (задержек) выполнения операций всегда должно включать в себя:
    a. график измеряемой величины во времени
    b. распределение величины по перцентилям по числу запросов
    c. гистограмму распределения
    d. требуемые перцентили: 0.5, 0.75, 0.9, 0.95. 0.99
4. Следует проверить хотя бы 2-3 параметра: если benchmark для web-фреймворка, то оценивать скорость сериализации объектов + время обработки тяжелых и средних запросов на бэкенде + одновременный логин большого (но допустимого) количества пользователей; если подразумевается непрерывность работы или выбор “самого доступного на данный момент” узла – оценивать время переключения на новый узел в случае загруженности ранее использованного, фактическое время ожидания заявки / пользователя / запроса в очереди; если это математика – время непосредственно на вычисления; если

какой-то сложный случай (и в очереди постоять, и посчитать что-то) – то разбить на cpu-bound и io-bound этапы и оценивать их отдельно

5. Требуется снимать данные об утилизации ресурсов разными компонентами – CPU, RAM, чтение / запись на диски (если используется GPU тоже учесть в отчете); в отчете также надо указывать информацию в формате “компонент – min/max/medium RAM, min/max/medium CPU, …” – итог можно сохранить в json (или другой формат с иерархической структурой); также требуется составить графики утилизации этих ресурсов; рекомендуется использовать Prometheus (https://prometheus.io/) или аналог для сбора статистики и отрисовки графиков
6. Ожидается такой способ использования benchmark:
    a. поднимается докер образ с выбранным объектом оценки
    b. запускается набор тестов для оценки важных параметров
    c. собирается статистика с такого прогона (на основании лога, либо через условный attach каких-то артефактов к шагам сценария и их анализом, ключевое – данные должны быть обработаны автоматически либо сгруппированы в .csv, на который можно натравить тот же excel)
    d. выполняем a.- c. шаги 100 раз
    e. собирается окончательная статистика по всем прогонам и формируется отчет
    f. проверяется альтернативный объект тем же способом и формируется отчет

Пожелание:

Попытаться подумать и сделать вывод об альтернативных фреймворках или даже языке реализации в целом на основании оценки производительности и того, как может измениться производительность в случае масштабирования проекта