Курсовая работа

Исследование особенностей функционирования серверных архитектур на основе PHP и NodeJS

Дерюгин Максим Сергеевич

Научный руководитель:

Быстрицкий Н.Д.

Аннотация

На сегодняшний день сервисы, измеряющие свою аудиторию миллионами пользователей, обрабатывают огромное кол-во запросов в секунду, особенно если это приложение в режиме реального времени. Так как приоритетными задачами в таких приложениях являются скорость отклика приложения и качество отправляемой пользователю информации, такие сервисы вынуждены выделять достаточно мощные (и даже с запасом) вычислительные системы, способные обработать такой поток данных - сервера. Чтобы вычислительная система могла справиться с получаемым потоком данных, принято прибегать к масштабированию системы (увеличение мощности системы путем увеличения кол-ва вычислительных и других устройств) и к оптимизации серверной логики. Так как масштабирование не может длиться бесконечно, и чем больше вычислительная система, тем она дороже и сложнее в обслуживании, то оптимизация серверной логики приложения является приоритетной задачей.

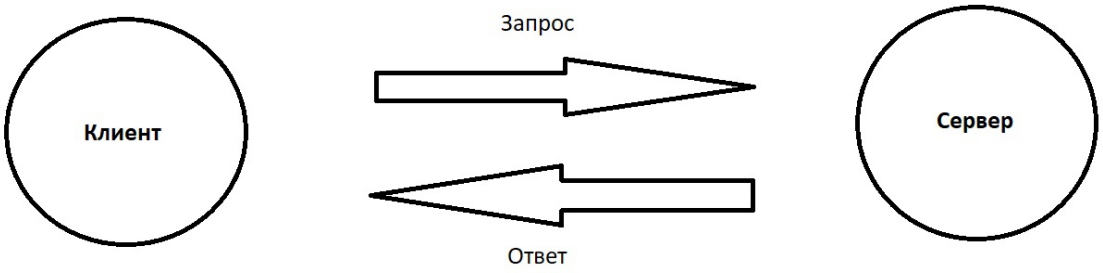
В данной работе приводится исследование особенностей функционирвоания сервера и клиент-серверного взаимодействия. Также рассматриваются два широко используемых языка программирования для реализации серверной архитектуры. В ходе исследования строится два эквивалетных сервера и сравнивается показатели скорости обработки запросов каждым из них.

Содержание

1 Исследование вопроса

1.1 Краткое описание предмета исследования

Серверной архитектурой называется програмное обеспечение, расположенное (как правило) на выделенной вычислительной машине, взаимодействующее с клиентским приложением посредством сетевых протоколов.
Клиент-серверное взаимодействие является одной из широко-используемых моделей функционирования приложения, в основе которой лежит разделение логических частей работы между клиентскими приложениями (заказчиками услуг) и серверными приложенями (исполнителями услуг). Причем все вычисления происходят именно на стороне сервера, клиенсткая же часть, принимает, обрабатывает данные от сервера, отображает их в корректном виде и реагирует на действия пользователя. Клиент-серверное приложение подразумевает выполнение ресурсоемких операций на стороне сервера и выдачу клиентской части готового результата.



1.2 Рассматриваемые функциональные характеристики при исследовании серверной архитектуры

Основными функциями серверной логики являются:

1. Вычисления;
2. Запросы к базам данных (далее БД);
3. Обработка данных (поиск, сортировка, шифрование и т.д.);
4. Расширение клиентского функционала (например, eventEmmiter);
5. Синхронизация клиентских сессий (например, месседжер);

Функционал серверной архитектуры строится индивидуально исходя из заданных целей приложения и возможности рабочей станции.

1.3 Выбор популярных инструментов для реализации серверной архитектуры

Для построения серверной архитектуры в настоящий момент могут быть использованы такие языки, как: PHP, Java, Python, JavaScript, C# и другие. Проводимое исследование основывается на сравнительном анализе серверных архитектур на PHP и JavaScript. Выбор по отношению к PHP обосновывается следующим:

1. У более чем 75% приложений в интернете серверная логика используют PHP (рис.1);
2. Самые популярные[1] CMS[2] написаны с помощью PHP;
3. Самый популярный[3] фреймворк для написания серверной архитектуры Laravel написан на PHP;

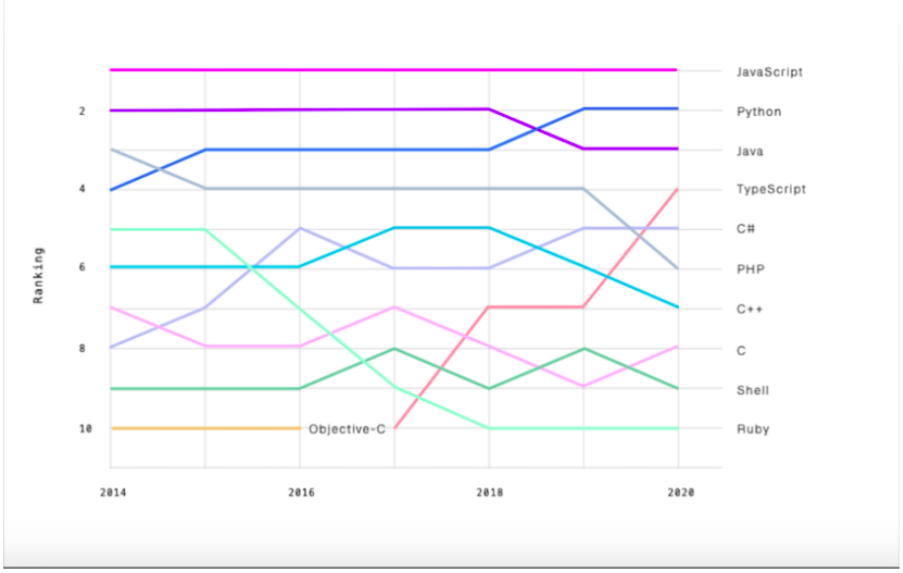
	2010 1 Jan	2011 1 Jan	2012 1 Jan	2013 1 Jan	2014 1 Jan	2015 1 Jan	2016 1 Jan	2017 1 Jan	2018 1 Jan	2019 1 Jan	2020 1 Jan	2021 1 Jan	2021 26 Apr
PHP	72.5%	74.8%	76.6%	77.7%	80.3%	80.6%	80.0%	80.0%	80.2%	78.9%	78.9%	79.1%	79.2%
ASP.NET	24.4%	23.2%	21.4%	19.9%	17.8%	16.7%	15.6%	14.8%	13.5%	11.8%	10.6%	9.3%	8.7%
Ruby	0.5%	0.5%	0.6%	0.5%	0.6%	0.9%	1.1%	1.3%	1.6%	2.4%	3.0%	4.3%	4.6%
Java	4.0%	3.8%	3.9%	4.0%	2.6%	2.8%	3.1%	3.3%	3.4%	4.0%	3.7%	3.2%	3.5%
Scala						0.2%	0.2%	0.3%	0.5%	1.2%	1.6%	1.8%	1.9%
static files							1.5%	1.5%	1.6%	2.1%	1.8%	1.6%	1.5%
Python	0.3%	1.0%	1.3%	1.5%	1.7%	1.6%	1.7%	1.6%	1.3%	1.1%	1.3%	1.4%	1.4%
JavaScript			<0.1%	<0.1%	0.1%	0.1%	0.2%	0.3%	0.4%	0.7%	0.8%	1.2%	1.3%
ColdFusion		1.3%	1.2%	1.1%	0.8%	0.7%	0.7%	0.6%	0.6%	0.5%	0.5%	0.3%	0.3%
Perl		1.1%	1.0%	0.8%	0.6%	0.5%	0.5%	0.4%	0.3%	0.3%	0.2%	0.2%	0.1%
Erlang						0.1%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%
Miva Script							0.1%	<0.1%	<0.1%	<0.1%	<0.1%	<0.1%	<0.1%

Рис. 1: Исследование используемых языков при написании серверной архитектуры по верссии w3techs.com

Хотя JavaScript (далее JS) изначально создавался как язык для клиентской части, выбор обосновывается следующим:

1. Один из самых популярных языков программирования на данный момент (рис.2)

[1]По версии портала w3techs.com https://w3techs.com/technologies/overview/content_management
[2]Content Management System - система управления содержимым
[3]На основе статистики числа загрузок по данным информационного ресурса Github.com



Рейтинг самых популярных языков программирования

Рис. 2: Анализ популярности языков программирования в декабре 2020 года по версии Github.com

2. Наличие среды Node.js для написания серверной архитектуры и самого большого пакетного менеджера npm (рис.3)

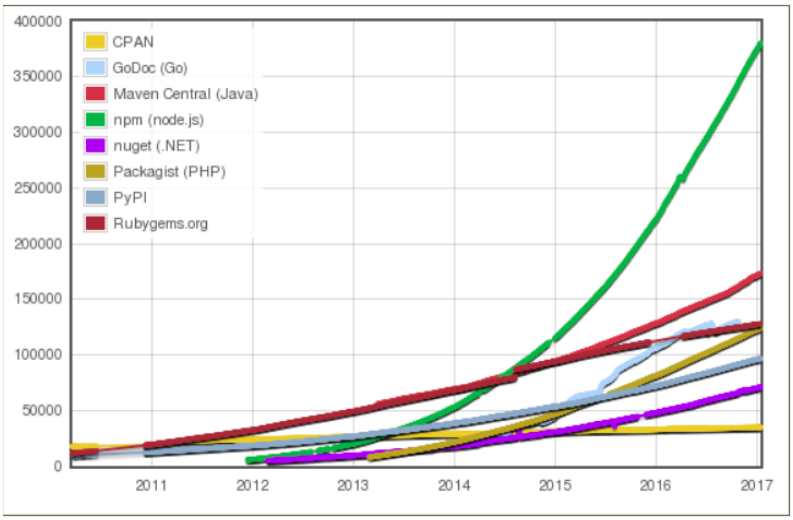


Рис. 3: Рейтинг пакетных менеджеров по кол-ву пакетов по версии OpenNET.ru

3. Наличие строго типизированной надстойки в виде TypeScript [1]

4. Возможность писать логику как серверной части, так и клиентской на одном языке

[1] https://www.typescriptlang.org/

2 Постановка задачи

2.1 Изучение основных производимых операций на сервере

Выше был перечисленен список многих операций, производимых на стороне сервера. Эти операции происходят именно на сервере по нескольким причинам:

1. Безопасность. Данные не выходят за пределы контролируемой зоны, так как их обработка происходит в одном месте. К тому же, логика клиентских скриптов может быть просмотрена из консоли разработчика в браузере;

2. Скорость. Отклик клиентского приложения зависит от мощности вычислительной машины, на которой запущен клиент (браузер). Это не позволяет способствовать высокой скорости обработки больших данных;

3. Централизация. Все приложения, работающие в реальном времени, требуют мгновенной реакции на действие одной клиентской сессии в других клиентских сессиях;

В любом приложении приоритетными задачами являются скорость и корректность выполнения запрашиваемой пользователем операции. Они достигаются путем оптимизации логики приложения, масштабирования и правильного выбора вспомогательных инструментов.
Вычисления и обработка данных высокопроизводительной системы (приложений) требуют подбор таких характеристик алгоритмов, которые оптимально задействовуют ресурсы системы. Для этого необходимо оценить сложность используемых алгоритмов, а так же объем занимаемых ими ресурсов и сопоставить это с возможностями системы исполнения.

2.2 Анализ построения серверной архитектуры

Построение серверной архитектуры включает в себя следующие основные понятия:

1. Настройка веб-сервера;

2. Описание серверной логики приложения;

Во многих случаях первый пункт можно не реализовывать, так как существуют практические готовые решения, такие как Apache[1], Nginx [2], Node.js веб-сервер[3]. Второй же пункт включает в себя такие понятия, как:

1. Обработка запросов, маршрутизация приложения;
2. Запросы к файловой системе/БД;
3. Передача данных с сервера на клиент;

Обработка запросов и маршрутизация - это возможность сервера реагировать на действия пользователя (и не только пользователя). Структурированный запрос на сервер, как правило, является композицией URL[4], метод запроса HTTP (GET, POST и др.) и тела запроса (необязательный параметр, наличие зависит от метода запроса).

Организация запросов к файловой системе и БД является одним из важнейших аспектов построения сервера. Необходимо придерживаться четкой логики при выборе между блокирующим и неблокирующим запросом, так как иначе могут возникнуть либо неоправданные замедления в работе, либо хаотичное поведение программы.

И хотя за непосредственную передачу данных на клиент при браузерном взаимодействии отвечает веб-сервер, при разработке серверной архитектуры требуется корректная организация посылемых данных. То есть, стоит при необходимости посылать данные малыми кусками и не включать в ответ ненужную информацию.

2.3 Анализ выбранных для исследования языков программирования для написания серверной логики

Как уже говорилось, для исследования были выбраны языки программирования JS и PHP. Их сходства:

1. Интерпретируемые, скриптовые языки;
2. Веб-ориентируемые инструменты;

Их различия:

[1]https://httpd.apache.org/

[2]https://nginx.org/

[3]https://nodejs.org/en/

[4]Universal Resource Locator - универсальный указатель местоположения документа в Сети интернет

1. на JS можно писать логику как для клиентской части (в контексте выполнения браузера), как и для серверной части (в контексте выполнения веб-сервера Node.js), в то время как код PHP предназначен только для серверной части (в контексте выполнения веб-сервера Apache/Nginx)

2. Отсутствие в PHP событийно-ориентированного подхода программирования, присущего JS

2.4 Принципы реализации эквивалентных по логике серверных архитектур

Для проведения исследования предлагается построить 2 максимально приближенных по логике серверных архитектуры, одну на PHP, другую на JS. Эта логика представляет собой REST-сервер[1], т.е. сервис для взаимодействия с другими веб-приложениями[2] (продолжение идеи взаимодействия распределенных систем в сети). В ходе исследвования будут осуществляться различные запросы к этим REST-серверам и производится соответствующие замеры.
Функционал REST-серверов включает в себя следующие возможности:

1. Запрос списка пользователей (считывание с файла, запись в файл и отправка данных с сервера на клиент);

2. Сортировка списка пользователей по определенному критерию;

3. Алгоритм большой сложности для замера скорости исполнения эквивалентных кодов на разных языках;

2.5 Определение необходимых критериев сравнения

Критерии сравнения строятся в соответствии с каждой функциональной возможностью REST-сервера. Сравнения осуществляются на основе времени обработки того или иного запроса (из трех предложенных выше запросов). Результаты сравнений предоставляются на отдельной клиентской странице.
Для сравнения составляются наборы пользователей разных размеров, имитирующих данные об абстрактных пользователях - БД сервера. Каждый пользователь включает в себя следующие характеристики:

[1]REST - архитектурный стиль взаимодействия компонентов распределённого приложения в сети
[2]Приложения, работающие в Интернете

1. Полное имя (char);
2. Пол (char);
3. Дата рождения (char);
4. Возраст (int);
5. Условный рейтинг (float);

Такой модели достаточно для сравнения работы каждого из рассматриваемых языков программирования с каждым элементарным типом. Размеры наборов растут с множителем 2: 290, 580, ..., 75000, 150000, 300000. Максимальный размер набора составляет 300 000 пользователей. Такой лимит был выбран на основе типового объема пользовательского списка информационного ресурса(системы) в сети Интернет. Для более точного результата можно увеличить это значения, однако не гарантируется уникальность элементов набора. Набор генерируется автоматически при исполнении скрипта generate.sh. Он необходим для работы тестов.
Клиентское приложение представляет из себя страницу с несколькими окнами, в которых присутвуют кнопки для отправки запроса и таблица, в которую заносятся результаты тестов.

3 Тестирование

3.1 Описание логики тестирования, ПО и машины, на которой проходит исследование

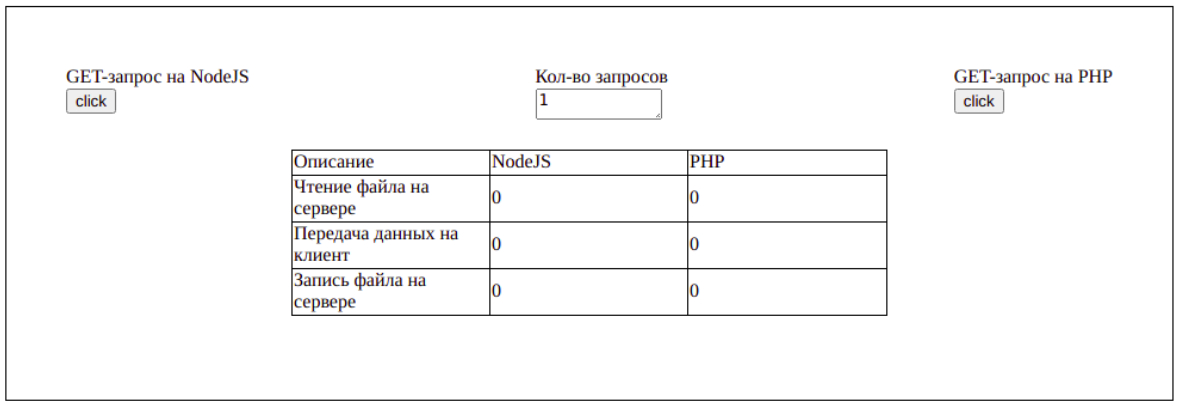
Исследвоание происходит на локальных хостах (с целью исключения фактора нестабильного подключения через Интернет).
Локальные хосты будут запущены на вычислительной машине со следующеми характеристиками:

1. Модель: Acer predator Helios 300 G3-572-78VX;
2. Технические характеристики:
   (a) Процессор: intel core I7-7700HQ, 2.8 - 3.8 ГГц, 4 физ. ядра, 8 лог. ядер;
   (b) Оперативная память: 2 x DDR4 8ГБ 2400 МГц/с;

(c) ОС: Ubuntu 20.04 LTS;

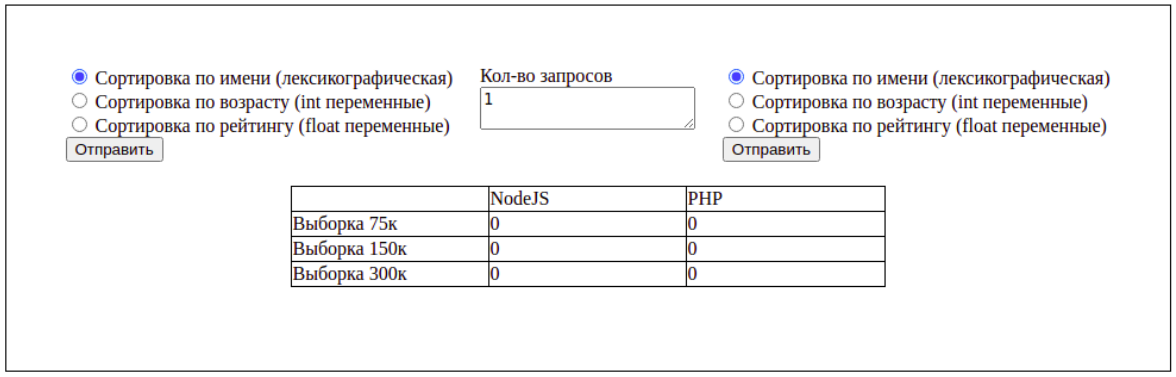
Тест представляет взаимодействие 2 REST-серверов и 1 клиентское приложение. Каждый тест содержит запрос к серверу и сравнение затраченного на обработку времени каждым сервером.

Первый из тестов является сравнением скорости чтения, записи и передачи данных с сервера на клиент:



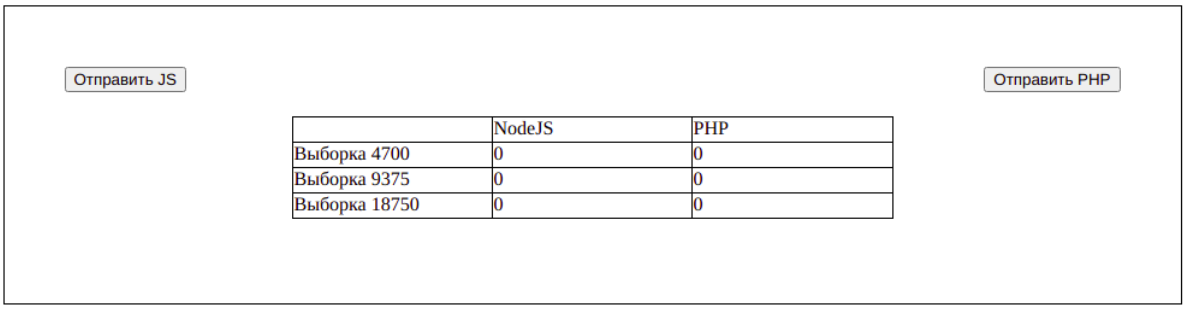
Клиент запрашивает от сервера файл с пользователями (300 000 пользователей) и передает его и время прочтения и записи на клиент. На клиенте же фиксируется общее время выполнения запроса, затем из него вычитается время чтения и записи и результат считаем временем отправки файла на клиент. Для большей точности вычислений можно указать большее число запросов и полученный результат поделить на это кол-во.

Второй тест содержит сравнение по скорости сортировок различных типов данных:



Клиент указывает тип сортировки и отправляет запрос на сервер. Сервер поочередно сортирует файлы размером 75 000, 150 000 и 300 000 пользователей и возвращает затраченное на сортировку время. Для рациональности и общности исследования взяты типовые используемые сортировки, имеющие приемлимые характеристики по времени вычисления. Тем самым сравниваются встроенные решения от разработчиков языка, которые предполагаются ниболее оптимальными. Как и в предыдущем тесте, для лучшей точности вычислений можно указать бОльшее число запросов.

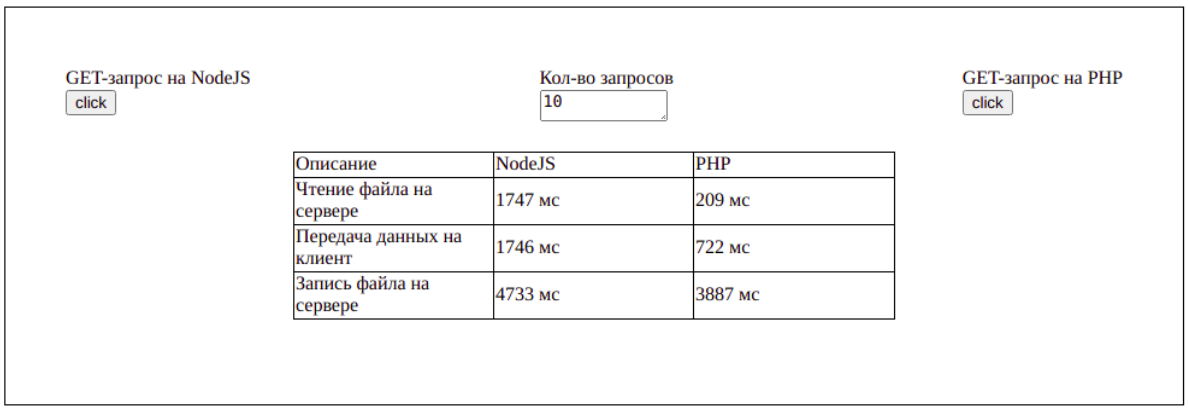
Третий тест содержит сравнение времени исполнения алгоритмов большой сложности. В качестве примера был выбран алгоритм сортировки пузырьком. Его сложность составляет $O(n^2)$.



Суть теста заключается в сравнении скорости исполнения одного и того же набора команд разными языками программирования с одинаковыми входными данными.

3.2 Результаты тестов

Результаты первого теста следующие:

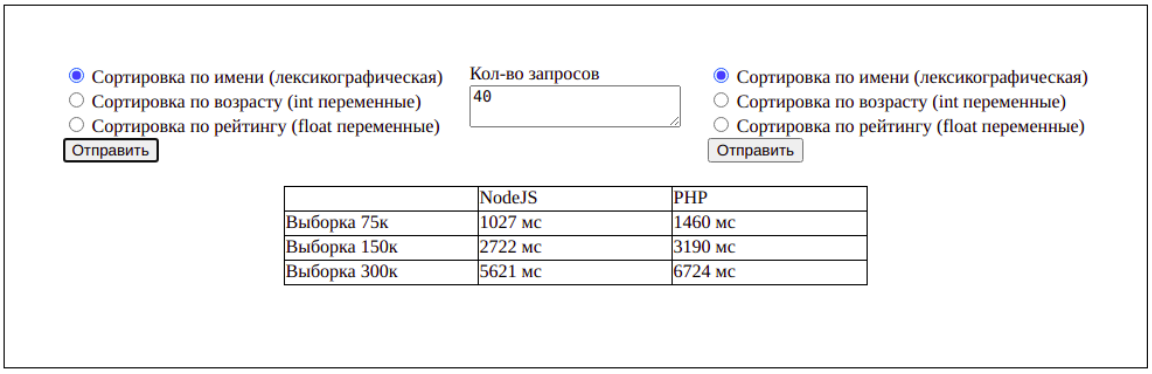


Из привиденных результатов выполнения тестов можно сделать вывод, что PHP намного быстрее считывает файлы (приблизительно в 7.37 раз быстрее) и передает

данные по протоколу на клиент (приблизительно в 2.55 раз быстрее), чем JS.

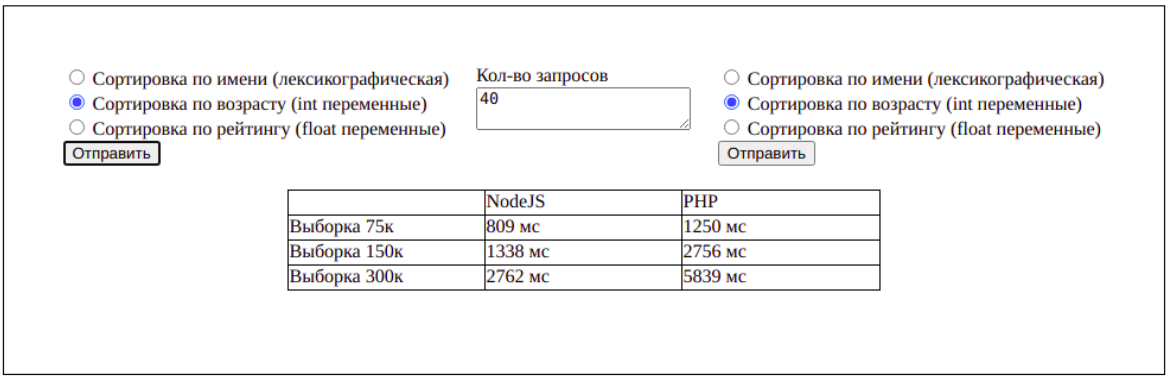
Однако результаты встроенных сортировок показывают, что JS быстрее PHP:

1. В среднем на 26% быстрее при сортировке строк (char):



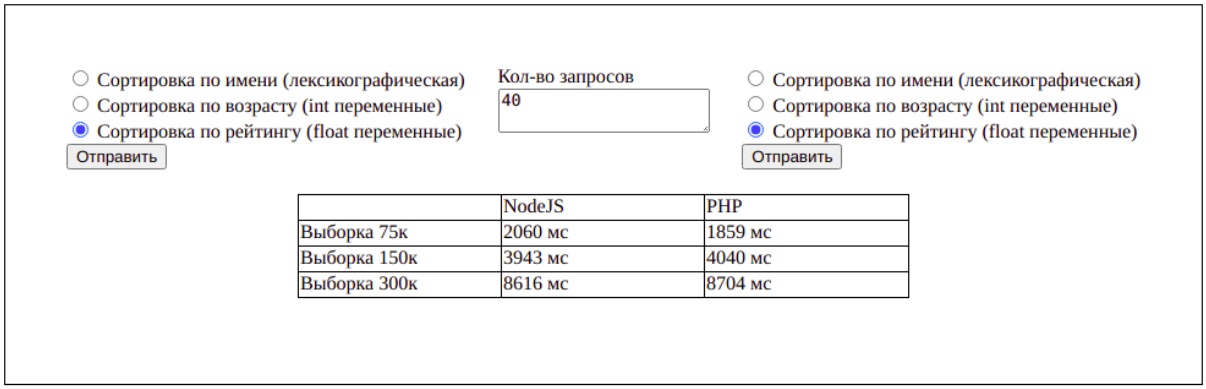
	NodeJS	PHP
Выборка 75к	1027 мс	1460 мс
Выборка 150к	2722 мс	3190 мс
Выборка 300к	5621 мс	6724 мс

2. В среднем на 90% быстрее при сортировке целых чисел: (int)



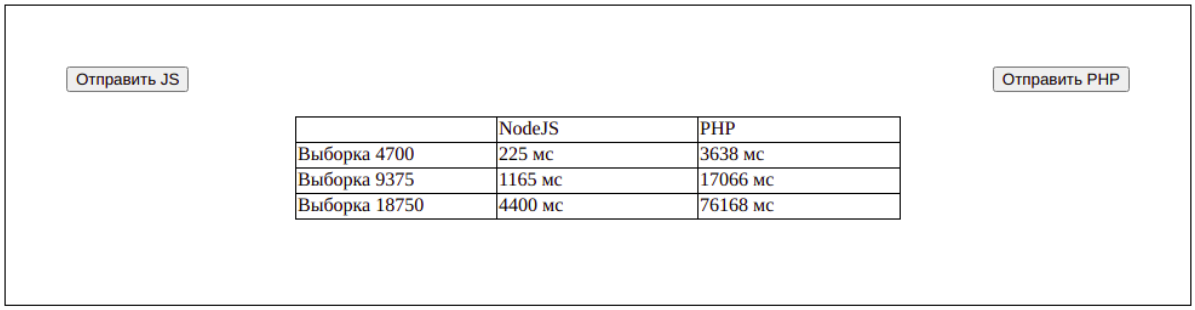
	NodeJS	PHP
Выборка 75к	809 мс	1250 мс
Выборка 150к	1338 мс	2756 мс
Выборка 300к	2762 мс	5839 мс

3. При float (с точностью 4 знаков после запятой) скорости сортировок совпадают:



	NodeJS	PHP
Выборка 75к	2060 мс	1859 мс
Выборка 150к	3943 мс	4040 мс
Выборка 300к	8616 мс	8704 мс

Результата третьего теста:



	NodeJS	PHP
Выборка 4700	225 мс	3638 мс
Выборка 9375	1165 мс	17066 мс
Выборка 18750	4400 мс	76168 мс

Третий тест показывает, что JS в среднем приблизительно в 16 раз быстрее PHP при такой сортирвке (а именно сортировке по возрасту (тип int) массива объектов с пятью свойствами).

4 Выводы

4.1 Анализ полученного результата работы

Из результатов теста можно сделать следующие выводы:

1. Работа с файловой системой в PHP значительно быстрее, чем в JS. Значит, при выборе языка для написания файлового менеджер (или другой программы, активно взаимодействующей с файловой системой) стоит выбирать PHP;

2. Встроенные сортировки и особенно скорость исполнения значительно быстрее у JS. Значит, при написании приложения, работающего в режиме реального времени стоит отдать предпочтения JS;

4.2 Перспектива дальнейшего исследования

Так как с каждым днем появляется все больше приложений, работающих в реальном времени в сети Интернет и требующих мгновенного отклика, это исследование несет большой вклад для текущей обстановки в сфере IT. Как раннее говорилось, масштабирование высоконагруженных систем не может производиться бесконечно, поскольку сильно растут финансовые затраты на непосредственное масштабирование и поддержку действующей системы. Поэтому эта работа позволяет определиться с языком программирования в зависимости от поставленных задач.

Список литературы

1. https://learn.javascript.ru/ - учебник по JS
2. https://developer.mozilla.org/ru/docs/Web/JavaScript - документация по JS от разработчиков языка
3. "PHP7 в подлиннике"Д.Котеров и И.Симдянов
4. https://www.php.net/manual/ru/index.php - документация по PHP от разработчиков языка