УДК 004.75

СПОСОБЫ МАСШТАБИРОВАНИЯ ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ WEB-ПРИЛОЖЕНИЙ

[1]ГуриновичА.А., [1]СкудняковЮ.А., [2]Гурский Н.Н.
[1]Белорусский государственный университет информатики и радиоэлектроники», г. Минск
[2] Белорусский национальный технический университет, г. Минск

В настоящее время в сфере разработки web-приложений существует множество проблем, связанных с недостаточной производительностью. Данные в больших web-приложениях на стороне сервера (или базы данных) могут занимать большие объёмы данных (например, данные об учащихся в международном масштабе) [1].

Данная работа посвящена способам масштабирования web-приложений. В первую очередь, организацию обновления визуальных данных в web-клиенте можно выполнить с помощью обычных HTTP-запросов по требованию пользователя. Например, в поисковом сервисе компании Google [2] используется именно такой метод: нажимая на кнопку поиска, пользователь отправляет HTTP-запрос на сервер, после чего получает ответ и обновляет информацию на странице. Во-вторых, организацию обновления визуальных данных в web-клиенте можно выполнить с помощью HTTP-поллинга. В этом случае используется тот же простой HTTP-запрос данных, как и в первом случае, однако он повторяется с некоторым интервалом времени, таким образом, web-клиент имеет возможность показывать информацию в реальном времени. Данный метод прост в реализации и может использоваться при разработке простого web-интерфейса сетевого устройства для мониторинга небольшого количества данных, например, мониторинг одной таблицы параметров сетевого устройства в реальном времени. Наконец, обновление визуальных данных можно реализовать с помощью WebSocket. В этом случае web-клиент открывает соединение с сервером и подписывается на необходимые топики («темы») данных, после чего сервер отправляет данные по подписке в тот момент, когда посчитает нужным, например, когда данные изменились в базе данных. Примером web-приложения может стать любое программное средство с обновлением данных в реальном времени, так как WebSocket является самым эффективным с точки зрения производительности и нагрузки на сервер.Для обработки больших данных выгоднее всего использовать WebSocket, так как размер потока сообщений между web-клиентом и сервером получается наименьшим и наиболее обоснованным, поскольку метод обновления только после запроса пользователя не подходит для создания приложения с данными, обновляющимися в реальном времени, а поллинг не подходит

для постоянной загрузки данных, когда их становится очень много и, к тому же, при использовании поллинга web-клиент производит запросы на сервер всегда, даже тогда, когда данные в базе данных не изменились. Явным недостатком при использовании WebSocket становится множество различных так называемых «обёрток» над протоколом, без единой стандартизации. В таком случае необходимо выбрать такой протокол, который будет поддерживаться и на стороне web-клиента, и на стороне сервера. Несмотря на крайне ограниченное количество готовых решений для реализации передачи данных по данному протоколу, существует реализация протокола с открытым стандартом WAMP, с помощью которого можно относительно просто реализовать передачу и прием данных через WebSocket. Чтобы решить проблему большого потока сообщений с сервера на web-клиент, например, в случае мониторинга большой сети, необходимо каким-то образом свести обработку каждого пришедшего сообщения к минимальному количеству операций, таким образом, уменьшив общее время обработки сообщений за определенный промежуток времени, что позволяет быстрее освобождать ресурсы для выполнения других задач (например, таких как построение DOM) в однопоточном web-клиенте. Буферизация данных позволяет эффективно решить данную проблему. Такая буферизация работает наподобие буферизации в процессоре, а именно при использовании буферизации web-клиент не обрабатывает пришедшие с сервера сообщения сразу, а помещает их в буфер, причем не в массив, а переносит все данные в одно сообщение, чтобы его можно было удобно в дальнейшем применить к имеющимся данным, и всего один раз за заданный промежуток времени.

Представим, что разрабатываемое web-приложение может содержать как малое количество данных в базе данных, так и очень большое количество данных. Например, приложение мониторинга сети в реальном времени. Если сетевых узлов мало (допустим, их количество может быть от 1 до 50), то буферизация сыграет небольшую роль, так как современные компьютеры имеют достаточные вычислительные ресурсы, чтобы успевать обработать телеметрию такого количества устройств (разумеется, если каждое сообщение не влечёт за собой, например, пересчет и перестроение графа топологической сети). Следовательно, нужно каким-то образом минимизировать влияние буферизации на визуальное отображение web-интерфейса пользователя, когда сетевых узлов, а точнее, сообщений с сервера мало (приходят реже, чем заданный ранее интервал буферизации), и постепенно увеличивать влияние буферизации на скорость обновления данных в интерфейсе для того, чтобы вычислительные операции не переполняли стек процесса и не вызывали визуальное затормаживание интерфейса, если сообщения начинают приходить чаще. Для этого предлагается обновлять таймер применения данных минимум до того момента, когда ни одно сообщение не придёт за заданный ранее интервал,

а максимум до того момента, когда значение суммы временных интервалов (учитывая обновления таймеров) достигнет какой-нибудь критической отметки для форсированного обновления. Например, если задать минимальный интервал обновления равным 200 мс, то в том случае, если в течение 200 мс придёт только одно сообщение, то данные из этого сообщения применятся к данным в уже существующем web-хранилище сразу после данного интервала, иначе, таймер обновится и будет ожидать сообщения, если оно вновь успеет прийти с сервера – таймер обновится и так далее. Так будет происходить до тех пор, пока сумма такого времени не достигнет критической отметки, равной, например, 1 секунде, и в этом случае данные, сформированные из всех пришедших за 1 секунду сообщений, добавятся к уже существующим данным в хранилище. Таким образом, при большом потоке данных обновление данных в web-интерфейсе будет происходить не чаще, чем 1 раз в секунду, а при малом потоке данных может происходить от 1 раза в 200 мс до 1 раза в 1 секунду. Рассмотрим реализацию данного алгоритма на языке JavaScript стандарта EcmaScript 2015. Класс с помощью данного алгоритма обрабатывает сообщение о событиях, произошедших в большой сети, поэтому, потребовалась буферизация, чтобы свести задержки операций к минимуму. Внешние обработчики сообщений WebSocket используют метод, передавая в него сообщения с обновлениями данных о событиях, произошедших в сети. Учитывая, что константа, определенная как Constants.DEFAULT_UPDATE_TIMEOUT, равна 200 мс, получаем реализацию буферизации. Технологии web-программирования позволяют реализовать web-приложения, удовлетворяющие как отсутствию чрезмерных вычислений и оптимизационных операций при малом потоке данных, так и отсутствию видимых задержек вычислений при большом потоке данных.

Различные задачи разработок требуют использования разных технологий, методов и подходов к обработке информации и оптимизации данного процесса, однако в объёмных web-приложениях остаётся выгодным использование протокола WebSocket для организации обновления данных в web-клиенте и буферизации для оптимизации этого обновления.

Литература

1. Пример использования AWS: Kaplan. [Электронный ресурс]. – Режим доступа: https://aws.amazon.com/ru/solutions/case-studies/kaplan/ – Дата доступа: 10.09.2019.

2. GoogleSearch. [Электронный ресурс].– Режим доступа: https:// www. google.com – Дата доступа: 10.09.2019.