УДК 004.657

А.С. Гавриленко, А.Г. Савенко

savenko@bsuir.by

Институт информационных технологий Белорусского государственного университета информатики и радиоэлектроники, г. Минск, Республика Беларусь

РАСПРЕДЕЛЕНИЕ НАГРУЗКИ ПРИ ПОСТРОЕНИИ ОТЧЁТОВ И ЗАПРОСОВ С БОЛЬШИМ ОБЪЁМОМ ДАННЫХ

При увеличении запросов в небольших и средних по объёмам обрабатываемой информации приложениях или сайтах, возникает вопрос о расширении тех или иных возможностей для ускорения отображаемых или загружаемых данных.

Рассмотрим проект, имеющий мобильное, web- и desktop-приложения. Все они подключаются к определенному API для осуществления взаимодействие по схеме клиент-сервер. В какой-то момент единственный воркер (процесс который выполняется в фоновом режиме) работающий на сервере начинает отключаться, вследствие чего, возникает ошибка. Данные события характеризуются тем, что API не хватает мощностей проекта, и даже после очередной перезагрузки воркера, работоспособность инструмента не будет восстановлена. Решением, в данном случае, может являться распределение нагрузки между нодами (сервер, или несколько серверов) расположенных на компьютерах, которые перенаправляют всю нагрузку от одного сервера на другие, тем самым упрощая и ускоряя работу единственного сервера.

Предлагается следующая схема перераспределения нагрузки. Для API, которое изначально выполняло функции получения информации, её обработке и выдаче, оставляется только первая. Получая какой-либо запрос (GET, POST, PUT, DELETE), API отдает параметры запроса в очередь RabbitMQ (брокер сообщений). Rabbit должен состоять из следующих частей: exchange (коммутатор), подписчиков, сообщений, имеющих хэш, записываемый в Redis (быстрая база данных по типу ключ-значение), который выступает в роли ключа и параметров запроса, выступающих в роли значения. К данным параметрам добавляется также статус, в котором находится текущий процесс, (request, process, succeed). В сообщениях также передается routing_key – ключ назначения. В данном случае ключ служит для перенаправления на нужный контроллер, в котором метод осуществляет логику. Запустив очередной воркер, осуществляется подписка на коммутатор, который выполняет распределение нагрузки между подписчиками следующим образом: если подписчик занят в текущий момент, коммутатор передаст сообщение тому, кто свободен или будет ждать того, кто быстрее всего освободится. Таким образом, подписчики выполняют все действия, которые ранее выполняло единственное API. Используя любую страницу, запрос, действие или функцию проекта, осуществляется распределение на воркер с наименьшей нагрузкой. Воркер, который выполнил определенный алгоритм действий, передает ответ в конкретную очередь, предназначенную для выполнения алгоритмов, и которая указана в сообщении. Параллельно всем действиям происходит подписка на очередь ответа (reply-to), в которую передается ответ алгоритма, заданного в воркере. И в случае какого-либо результата – передается ответ клиенту.

Проанализируем данную схему на предмет быстродействия. Данные для обработки, будут поступать под разнообразные задачи. Может наступить момент, когда воркеры загружены большими длительными задачами и мешают выполнению мелких задач. Для того чтобы это не происходило, необходимо разграничить воркеры. Для разграничения воркеров, нужно выявить мелкие задачи и задачи с большими объемами данных, тем самым разделив воркеры на несколько групп (в том числе группу воркеров обрабатывающих отчеты). Этими действиями можно добиться максимального ускорения системы, т.к. воркеры, будут разграничены, а мелкие задачи, не буду ждать воркеров, обрабатывающих большие данные. Однако, если ограничиться равным количеством воркеров, это не повысит, т.к. некоторые воркеры будут простаивать впустую. Для исключения такой ситуации, нужно распределить количество воркеров: для мелких задач необходимо выделить небольшое количество воркеров (т.к. задачи быстро выполняются), а для более сложных задач следует выделить большее количество воркеров (т.к. они обрабатываются долго). Если для схемы будет использоваться только один клиент, а результат при множественных одинаковых запросах обрабатывается каждый раз, то использование серверных мощностей будет не совсем рационально. Чтобы не загружать сервера на которых работают воркеры можно использовать сервер Redis, как средство для хранения кэша обработанных результатов. Поиск текущего хэша в Redis производится быстрее, чем очередная обработка такого же результата с последующей его обработкой на воркерах. Для осуществления кэширования, требуется результат выполнения воркера отправить в Redis, где результат будет значением, а хэш, который был сгенерирован ранее, будет ключом.