Кэширование сегодня является неотъемлемой частью любого Web-проекта, не обязательно высоконагруженного. Для каждого ресурса критичной для пользователя является такая характеристика, как время отклика сервера. Увеличение времени отклика сервера приводит к оттоку посетителей. Следовательно, необходимо минимизировать время отклика: для этого необходимо уменьшать время, требуемое на формирование ответа пользователю, а ответ пользователю требует получить данные из каких-то внешних ресурсов (backend). Этими ресурсами могут быть как базы данных, так и любые другие относительно медленные источники данных (например, удаленный файловый сервер, на котором мы уточняем количество свободного места). Для генерации одной страницы достаточно сложного ресурса нам может потребоваться совершить десятки подобных обращений. Многие из них будут быстрыми: 20 мс и меньше, однако всегда существует некоторое небольшое количество запросов, время вычисления которых может исчисляться секундами или минутами (даже в самой оптимизированной системе один могут быть, хотя их количество должно быть минимально). Если сложить всё то время, которое мы затратим на ожидание результатов запросов (если же мы будем выполнять запросы параллельно, то возьмем время вычисления самого долгого запроса), мы получим неудовлетворительное время отклика.  
  
Решением этой задачи является кэширование: мы помещаем результат вычислений в некоторое хранилище (например, memcached), которое обладает отличными характеристиками по времени доступа к информации. Теперь вместо обращений к медленным, сложным и тяжелым backend’ам нам достаточно выполнить запрос к быстрому кэш

Принцип локальности

Кэш или подход кэширования мы встречаем повсюду в электронных устройствах, архитектуре программного обеспечения: кэш ЦП (первого и второго уровня), буферы жесткого диска, кэш операционной системы, буфер в автомагнитоле. Чем же определяется такой успех кэширования? Ответ лежит в принципе локальности: программе, устройству свойственно в определенный промежуток времени работать с некоторым подмножеством данных из общего набора. В случае оперативной памяти это означает, что если программа работает с данными, находящимися по адресу 100, то с большей степенью вероятности следующее обращение будет по адресу 101, 102 и т.п., а не по адресу 10000, например. То же самое с жестким диском: его буфер наполняется данными из областей, соседних по отношению к последним прочитанным секторам, если бы наши программы работали в один момент времени не с некоторым относительно небольшим набором файлов, а со всем содержимым жесткого диска, буферы были бы бессмысленны. Буфер автомагнитолы совершает упреждающее чтение с диска следующих минут музыки, потому что мы, скорее всего, будем слушать музыкальный файл последовательно, чем перескакивать по набору музыки и т.п.  
  
В случае web-проектов успех кэширования определяется тем, что на сайте есть всегда наиболее популярные страницы, некоторые данные используются на всех или почти на всех страницах, то есть существуют некоторые выборки, которые оказываются затребованы гораздо чаще других. Мы заменяем несколько обращений к backend’у на одно обращения для построения кэша, а затем все последующие обращения будет делать через быстро работающий кэш.  
  
Кэш всегда лучше, чем исходный источник данных: кэш ЦП на порядки быстрее оперативной памяти, однако мы не можем сделать оперативную память такой же быстрой, как кэш – это экономически неэффективно и технически сложно. Буфер жесткого диска удовлетворяет запросы за данными на порядки быстрее самого жесткого диска, однако буфер не обладает свойством запоминать данные при отключении питания – в этом смысле он хуже самого устройства. Аналогичная ситуация и с кэшированием в Web’е: кэш быстрее и эффективнее, чем backend, однако он обычно в случае перезапуска или падения сервера не может сохранить данные, а также не обладает логикой по вычислению каких-либо результатов: он умеет возвращать лишь то, что мы ранее в него положили.