MEMCACHED – РАСПРЕДЕЛЕННАЯ СИСТЕМА КЭШИРОВАНИЯ ПАМЯТИ

Николай СКУРТУ

Департамент Программной Инженерии и Автоматики, Группа TI-195, Факультет Вычислительной Техники и Микроэлектроники, Технический Университет Молдовы, Кишинёв, Республика Молдова

Автор корреспондент: Николай СКУРТУ, e-mail: scurtu.nicolai@isa.utm.md

Научный руководитель: Дориан САРАНЧУК, DISA, FCIM, UTM

Аннотация. *В данной статье представлены общие представления о методике кэширования, применяемой для баз данных, на примере одной из систем – Memcached. Поскольку Memcached – это относительно простая, долго существующая и устоявшаяся система, на её примере легко показать основаные достоинства, и что немаловажно, недостатки данного подхода к кэшированию в оперативной памяти.*

Ключевые слова*: Memcached, кэш, LRU, клиент-серверная архитекура, оперативная память, база данных*

Введение

При работе с базами данных неизбежно возникают ситуации, когда некоторые из запросов к серверу обслуживаются особенно долго по сравнению с остальными. Это может происходить из-за того, что запрос написан неэффективно, индексы используются неудачно, либо по другим причинам. Все эти ситуации не являются предметом интереса для данной статьи, но бывают ситуации, где даже, если учесть все вышеперечисленные аспекты, все равно время выполнения запроса оставляет желать лучшего.

Например, в следующем SQL-запросе обрабатывается относительно большой объём данных [1]:

```
1 SELECT *
2 FROM hugetable
3 WHERE timestamp > lastweek
4 ORDER BY timestamp ASC LIMIT 50000;
```

Конечно, для начала необходимо подумать, нельзя ли снизить время выполнения таких длительных запросов, учитывая особенность конкретной предметной области. Если, например, выяснится, что в 90% случаях пользователю нашей базы данных требуется всего 10 первых записей, но не в коем-случае не 50 000, то, очевидно, можно создать нужные индексы, понизить лимит запрашиваемых записей до 10 и сделать дополнительную опцию для запроса бóльшего количества записей.

Тем не менее, остается вопрос, а что же делать, если количество запрашиваемых данных в запросе оптимизировано, но из-за большого количества пользователей один и тот же запрос повторяется очень часто. Зачастую для таких ситуаций целесообразно использовать кэш.

Кэш – это промежуточный буфер с быстрым доступом к нему, содержащий информацию, которая может быть запрошена с наибольшей вероятностью. Доступ к данным в кэше осуществляется быстрее, чем выборка исходных данных из более медленной памяти или удалённого источника, однако её объём существенно ограничен по сравнению с хранилищем исходных данных [2].

Memcached – это система, которая позволяет осуществлять кэширование результатов уже выполненых базой данных запросов в оперативной памяти.

Принцип работы системы Memcached

Memcached - это распределенная система кэширования памяти общего назначения. Memcached основана на клиент-серверной архитектуре. Memcached-сервер предоставляет услуги по кэшированию данных, а клиент предоставляет эти данные в форме пар ключ-значение. Важно то, что значение в данном случае – просто строка, то есть Memcached ничего не знает о формате используемых вами данных: он просто хранит строки, их интерпретация – задача клиента. Теперь, перед тем как в очередной раз потребуется сделать запрос к базе данных, сначала нужно обратиться к Memcached-серверу и узнать, не хранятся ли эти данные в памяти Memcached-сервера, и если хранятся, то получить их оттуда, нежели запрашивать их у базы данных. В результате такого двуступенчатого подхода (сначала заглянуть в кэш, и если данных там не окажется, заглянуть в базу данных) можно добиться прироста производительности и снижения нагрузки на базу данных. Действительно, извлечь данные, которые хранятся в оперативной памяти в виде пар ключ-значение, намного проще, чем анализировать SQL-запрос и, в худшем случае, загружать эти данные с диска.

Рисунок 1. Успешный запрос к Memcached-серверу

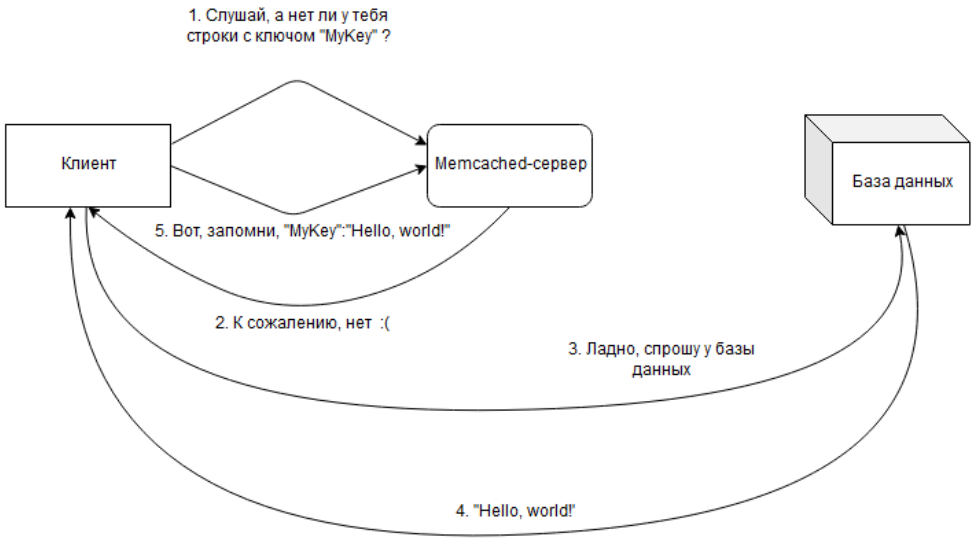

Рисунок 2. Неудачный запрос к Memcached-серверу

Важно также понимать, что Memcached – это инструмент программиста: именно программист решает где, когда, и как будут запрашиваться данные и сохраняться. Memcached – это не автоматическая прослойка, которая магическим образом ускоряет выполнение всех запросов к БД. Чтобы Memcached действительно ускорил работу какой-то программы, нужно

чтобы эта программа сама взаимодействовала с Memcached-сервером. Это отчётливо видно на Рис.2. Клиент, после того как получил отрицательный ответ на шаге 2, сам, явно, решил сохранить пару ключ-значение в памяти Memcached-сервера (шаг 5). Это говорит о том, что программа вообще-то не обязана использовать кэш. Если из тех или иных соображений часть данных в любом случае должна быть запрошена напрямую из базы данных, то, благодаря ненавязчивости Memcached, у программиста будет такая возможность. Это демонстрирует определенную гибкость при использовании данного инструмента. Тем не менее, гибкость такого рода присутствует не во всех аспектах Memcached.

Архитектура Memcached

В прошлом параграфе подразумивалось, что в системе присутствует всего один Memcached-сервер. К слову, это совершенно не обязан быть отдельный физический узел: это можеть быть отдельный процесс, запущенный на той же машине, что и веб-сервер и\или база данных. В данном случае лишь указывалось логическое расположения частей системы, а не физическое. В данном параграфе будет раскрыто слово «распределенная» в описании системы Memcached. «Распределенная» в данном контексте означает, что Memcached-сервер не обязан быть один – их может быть несколько. Самое приятное тут, что от этого особенно ничего не меняется. Дело состоит в том, что Memcached-серверы друг о друге ничего не знают. Это достигается за счет того, что клиент по ключу определяет к какому серверу стоит обратиться и каждый ключ обслуживается всего одним сервером. Информация не реплицируется, то есть серверы не хранят одну и ту же информацию, а, наоборот, каждый сервер ответственен за какой-то уникальный набор ключей. В таком подходе требуется всего лишь перечислить адреса всех серверов в клиенте, чтобы он мог обращаться к ним по необходимости.

Самое время вспомнить, что мы живем не в идеальном мире и компьютеры как и все хоть сколько-нибудь сложные механизмы иногда выходят из строя. Нужно задуматься, а что же будет, если такое случится с одним из (или если вам черная кошка перешла дорогу, то сразу с несколькими) Memcached-серверов. Несомненно, весь кэш, который хранился на этом сервере будет недоступен, следовательно, любые обращения к этим данным со стороны клиента будут неудачными. Получается, что обращения к кэшу, в данном случае, будут только тормозить систему. Дело усугубляется тем, что раз один из серверов недоступен, то алгоритм, по которому клиент выбирает сервер для ключа, следует бы подкорректировать, чтобы он не порождал запросы к недоступному серверу, а перераспределил нагрузку между оставшимися серверами. Вопрос теперь в том, а насколько хорошим будет это перераспределение: перераспределятся ли только ключи, за которые отвечал ныне недоступный сервер или сразу все ключи. Очевидно, нет смысла «передвигать» и так хорошо работающий кэш. Здесь определяющую роль играет выбор Memcached-клиента, в качестве которого может выступать, например, библиотека для языка C. Хорошие клиенты реализуют так называемый Consistent Hashing. Это алгоритм, который как раз устойчив к убавлению или добавлению серверов Memcached.

Остается теперь ответить на вопрос, что же будет делать Memcached, когда оперативная память, выделенная ему, будет заканчиваться. Если требуется сохранить какую-то информацию, но для неё нет свободной памяти, то Memcached будет искать «жертву», чтобы вместо нее записать новую информацию. Эту «жертву» Memcached ищет по алгоритму LRU (Least Recently Used). Суть этого алгоритма в том, чтобы найти участок памяти, который дольше всего не использовался (является наименее востребованным), и заменить его содержимое новыми данными. Данный алгоритм полагается на то, что клиент запрашивает какую-то часть данных с большей частотой, а остальную часть запрашивает гораздо реже. Таким образом, исключение менее востребованных данных не повлечет за собой рост частоты кэш-промахов (или, по крайней мере, будет её минимизировать).

Основные недостатки системы Memcached

Как уже было сказано, Memcached – это инструмент программиста, при этом довольно простой. Это с одной стороны достоинство, а с другой стороны, во многих ситуциях, недостаток. Memcached не предоставляет никаких типов данных, тем самым не позволяя получать или обновлять данные частично. Например, профиль пользователя может состоять из большого набора полей, и для обновления всего одного поля не нужно перезаписывать всю структуру данных целиком. С другой стороны, если использовать Memcached для статического веб-сайта, на котором информация обновляется крайне редко, то такой подход позволяет хранить HTML-код в виде одной строки, целиком. В данном случае Memcached позволяет избежать накладных расходов с хранением и анализом информации, связанной с типами данных.

Другой недостаток заключается в жестком ограничении на политику замещения памяти, а именно такая политика ровно одна – LRU, о которой было рассказано выше. LRU – это не единственно возможный подход по замещению памяти. Например, можно исключать случайно попавшиеся ключи, если у нас есть дополнительные предположения относительно нашей системы (равномерное распределение запросов по ключам), и сэкономить на накладных расходах, связанных с алгоритмом LRU. Даже сам алгоритм LRU допускает несколько разных подходов в реализации: можно использовать приближение к этому алгоритму, которое дает достойный результат, но требует меньше ресурсов.

Еще один недостаток связан с тем, что сохранять данные на диске Memcached без сторонних инструментов не позволяет. Хотя, учитывая то, что Memcached предназначен для одной конкретной цели – кэширования, то можно сказать, что он и не должен предоставлять лишних функций, которые напрямую не связаны с самим кэшированием.

Заключение

Memcached – это специализированный инструмент. Он выполняет свою задачу по эффективному кэшированию и делает это быстро и просто [3]. Главный его недостаток в контектсе одной задачи оказывается его главным достоинством в контексте другой задачи. Если задача достаточно хорошо оптимизируется тем набором услуг, который предлагает Memcached, то нет необходимости его заменять на более сложный инструмент. Ведь, более сложный инструмент не только предлагает больше опций, но также и больше багов. Известны случаи, когда лишний функционал библиотеки, о котором не подозревало большинство ее пользователей, оказывался мишенью для атак из-за некоторых уязвимостей. Напротив, если задача сама по себе предполагает использование тех опций, которыми не располагает Memcached, то стоит поискать альтернативы, например, довольно популярное на данный момент решение это Redis [4]. Redis – это намного больше, чем просто кэш в оперативной памяти, соответственно там есть и выбор из несколько политик замещения, множество типов данных и т.д [5].

Тем не менее, мир не ограничен только уже существующими системами. Иногда требуется реализовать свою собственную систему, а для этого очень полезно бывает взглянуть на опыт написания схожих систем.

Библиография

1. Memcached Github Wiki. – [Электронный ресурс] [дата обращения 02.03.2023], Доступно: https://github.com/memcached/memcached/wiki
2. Memcached Wikipedia. – [Электронный ресурс] [дата обращения 02.03.2023], Доступно: https://en.wikipedia.org/wiki/Memcached
3. VIJAY CHIDAMBARAM, DEEPAK RAMAMURTHI Performance Analysis of Memcached. [Электронный ресурс] [дата обращения 02.03.2023], Доступно: http://pages.cs.wisc.edu/~vijayc/papers/memcached.pdf
4. Using Redis as an LRU cache. – [Электронный ресурс] [дата обращения 02.03.2023], Доступно: https://redis.io/topics/lru-cache
5. Stackoverflow Memcached vs Redis. – [Электронный ресурс] [дата обращения 02.03.2023], Доступно: https://stackoverflow.com/questions/10558465/memcached-vs-redis