Архитектура NoSQL СУБД для типичного высоконагруженного приложения

А. Ю. Платов, email: platoff@mail.ru

Нижегородский государственный архитектурно-строительный университет

А. С. Прокопенко, email: a.prokopenko@internet.ru

Московский государственный технический университет имени Н. Э. Баумана

Аннотация. *Предложена архитектура СУБД на основе NoSQL, удовлетворяющего типичным требованиям, приемлемым для подавляющего числа организаций, использующих высоконагруженные приложения: простота использования, независимость объёма данных от оперативной памяти, скорость работы. Обосновано применение конкретных решений, реализующих базовый функционал, описана модель классов СУБД.*

Ключевые слова: *высоконагруженные приложения, NoSQL, базы данных в памяти, базы данных типа ключ-значение, C++.*

Введение

За последнее десятилетие отмечается значительный рост потребности в использовании высоконагруженных приложений (highload-систем). Расширение сетевой аудитории и сопутствующий объём трафика, рост объёмов передаваемых и обрабатываемых данных заставляет искать проектные решения, позволяющие максимально использовать возможности сетевой и вычислительной аппаратуры [1]. В связи с этим особое значение при проектировании highload-систем приобретают вопросы структуры хранения данных и архитектуры связей между компонентами системы.

Одним из распространённых подходов для управления данными при реализации highload-систем является использование NoSQL. Для простой структуры данных наиболее подходящий тип NoSQL СУБД, является модель «ключ-значение», которая ориентирована на обеспечение максимальной горизонтальной масштабируемости (добавление вычислительных мощностей увеличивает производительность).

На рынке уже давно существует ряд решений в области СУБД в памяти типа «ключ-значение». Наиболее известные из них это СУБД с открытым исходным кодом «Redis», «Tarantool» и «Memcached», а также проприетарные, к которым относится, например, «CouchBase».

Особенностью СУБД «Redis» является то, что база данных хранится в оперативной памяти, что ограничивает объём данных.

В СУБД «Tarantool», помимо внутреннего языка запросов, поддерживается SQL, а также возможна реализация бизнес-логики при работе с данными на языке Lua. При этом можно выбирать ядро приложения для хранения данных: работающее как традиционная in-memory СУБД, или как СУБД, использующая жесткий диск в сочетании с оперативной памятью. Второй режим позволяет работать с данными, чей объём в 10-100 раз больше доступного объёма оперативной памяти [2].

«Memcached» – это СУБД с максимально простым интерфейсом, которая чаще всего используется для кэширования данных или хранения сессий, поскольку система не делает обращений к жесткому диску, и данные в случае аварии могут быть утеряны.

СУБД CouchBase, в основе которой лежит Memcached, является проприетарным решением для крупных компаний (предоставляется поддержка и другие преимущества, важные для бизнеса). Кроме схемы типа «ключ-значение» позволяет создавать также документно-ориентированные базы данных. Эта СУБД очень популярна на рынке, несмотря на меньшую производительность.

Все эти СУБД реализованы на C или C++ и предоставляют интерфейс для разных языков программирования.

Каждое из описанных решений характеризуется своими «плюсами» и «минусами», позволяющими разработчикам ориентироваться на тот или иной набор требований. Ниже предлагается решение, удовлетворяющее следующему принципиальному набору требований:

- простота интерфейса;
- возможность использования в приложениях на языке C++;
- поддержка нескольких режимов ядра СУБД (все данные в памяти; часть данных в памяти, часть на жестком диске; все данные на жестком диске; перенаправление части запросов на обращение к жесткому диску);
- классические требования к СУБД: создание резервных копий, поддержка языка запросов, авторизации, поддержка ACID (атомарность согласованность, изолированность, стойкость), поддержка нескольких клиентов одновременно.

Этот набор требований представляется наиболее общим и потому приемлемым для очень большого числа организаций или предприятий, нуждающихся в поддержке и развитии высоконагруженных веб-приложений.

1. Разработка моделей

Очевидная модель работы СУБД – это клиент-сервер: клиент отправляет запросы к серверу, который его обрабатывает и отдает соответствующий ответ. Поэтому необходимо разработать архитектуру двух приложений, а также их методы взаимодействия друг с другом.

Наиболее удобный формат для подобной документации – UML-диаграмма [3]. В её основе лежит специально разработанный графический язык, который описывает архитектуру приложения в терминах объектно-ориентированного программирования (ООП): классы, методы, их взаимодействие друг с другом.

На рисунках 1, 2 представлены UML-диаграммы классов разрабатываемого проекта. Поскольку модель работы СУБД – клиент-серверная, то на диаграмме изображены две главные части проекта – клиент (Client) и сервер (DataBase) (выделены прямоугольниками).

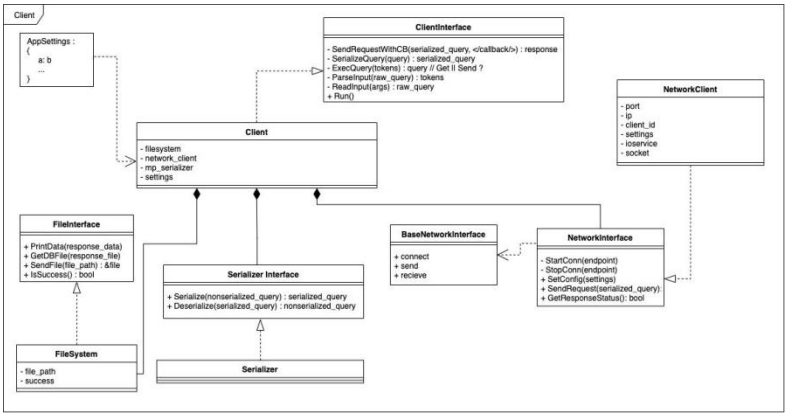

Рис. 1. UML-диаграмма классов клиентской части приложения

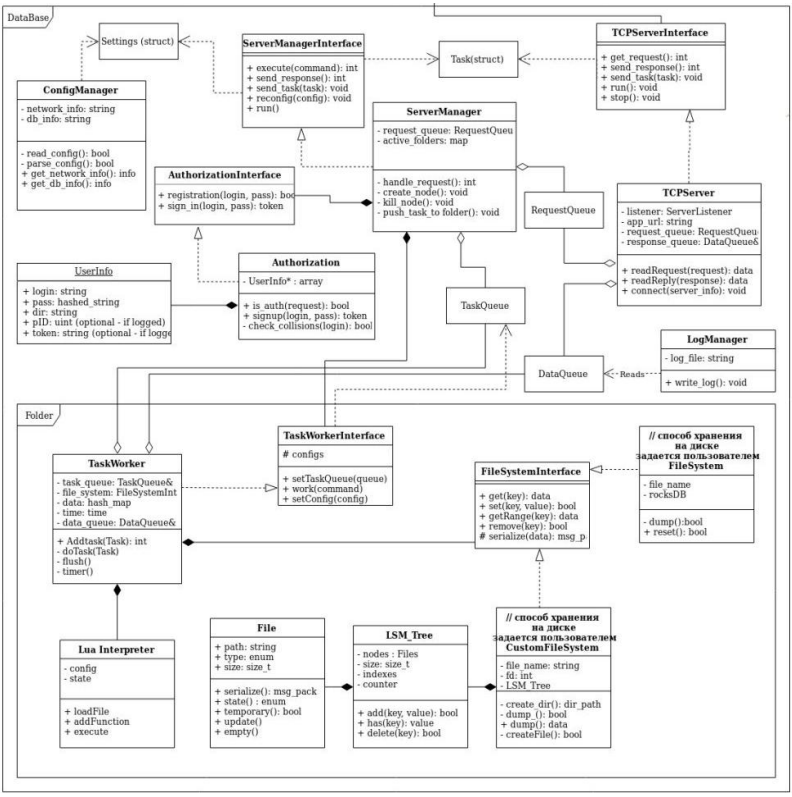

Рис. 2. UML-диаграмма классов серверной части приложения

Клиент и сервер взаимодействуют друг с другом через классы NetworkInterface (представляет программный интерфейс приложения, его функционал) и TCPServerInterface (принимает запросы клиентов, и отправляет им данные).

Ядро СУБД описывается классом ServerManager, который взаимодействует с остальными модулями приложения: работа с сетью, авторизация. ServerManager оперирует массивом баз данных, с которыми в данный момент работают клиенты. В этом классе происходит работа с запросами пользователей – разбор запросов, определение того, что нужно сделать с данными. После разбора запроса, на его основе создается задача на исполнение, и она поступает в очередь задач соответствующей базы данных. Экземпляр такой базы данных описывает модуль Folder. Он содержит очередь задач в классе

TaskWorker, которые поступают от клиентов. В классе TaskWorker происходит непосредственно выполнение задачи.

Журналирование запросов устанавливается в настройках, чтобы в случае аварийного завершения программы данные не пострадали. Класс LogManager может просматривать очередь задач, и в зависимости от настроек может осуществляться журналирование запросов.

В случае, если запрос является скриптом для извлечения данных, то DataWorker подготавливает и передает его классу LuaInterpreter, который затем исполняется интерпретатором языка Lua.

Вся работа с жестким диском и оперативной памятью происходит в классе FileSystem. Он осуществляет сброс данных на диск, а также загрузку этих данных в оперативную память.

После выполнения запроса его результат передается в очередь выполненных задач, с которой работает класс ServerManager. После изъятия из очереди результата запроса, формируется ответ от сервера на его основе и отправляется клиенту, используя класс TCPServer.

Перед тем, как приступать к реализации описанной схемы, требуется разработать модели представления данных, с которыми будет оперировать конечный пользователь СУБД. Зная модели и термины, их определяющие, можно создать скриптовый язык, наличие которого необходимо для работы с СУБД конечному пользователю.

В качестве базы данных было введено понятие «folder» (папка). У каждого пользователя может несколько папок. Пользователь может производить различные операции с каждой из них: добавлять/удалять/обновлять элементы, искать по ключам, делать копию данных. Простейший сценарий работы представлен на листинге 1. Элемент базы данных – это пара «ключ-значение». Важным аспектом при реализации являются поддерживаемые типы данных в СУБД. Для начального этапа было принято решение использовать в качестве типа ключей – строки, а для значений – бинарное представление. В будущем поддерживаемые типы для значений могут быть расширены на числа и строки.

2. Реализация архитектуры

Для реализации был выбран следующий базовый сценарий работы:

– авторизация пользователя;
– создание «папки», определяющей базу данных;
– работа с папкой;
– разрыв соединения.

Исходя из этого сценария СУБД должна обладать следующим функционалом:

1. авторизация;

2. создание пользователя;
3. смена пользователя;
4. создание баз данных;
5. смена баз данных;
6. удаление баз данных;
7. журналирование;
8. работа с элементами «папки» – ключами и их значениями (создание/изменение/удаление/поиск);
9. создание резервных копий.

Важнейшим фактором в работе СУБД в памяти является скорость её работы [4]. С другой стороны, разработка с использованием высокоуровневых языков программирования, в которых возможно использование ООП парадигмы, ускоряет время, затрачиваемое на написание программы. Учитывая эти факторы, а также существующие библиотеки, способные упростить разработку, языком программирования для проекта был выбран C++, отвечающий заданным критериям. C++ позволяет использовать преимущества низкоуровневого программирования для обеспечения скорости работы в сочетании с удобным синтаксисом, ООП и библиотеками Boost.

В качестве протокола общения между клиентом и сервером был выбран TCP, поскольку он часто используется в аналогичных проектах. Это обуславливается тем, что чем более высокоуровневым является протокол передачи данных (по модели ISO/OSI), тем больше в нем функций. Если функции высокоуровневого протокола не нужны, то его использование – это трата ресурсов. TCP подходит к нашим требованиям.

Конечный пользователь взаимодействует с сервером базы данных, используя программный клиент и скриптовый язык запросов. В случае реляционных СУБД это обычно SQL, в случае NoSQL СУБД он может отличаться у каждой реализации. В нашем случае, был разработан простой скриптовый язык запросов, покрывающий все сценарии работы с СУБД. Для его исполнения в клиент встроен интерпретатор этого языка, который отправляет разобранный запрос на сервер.

Основная особенность работы данной СУБД – при соединении с базой данных (папкой) создается поток, в котором происходит обработка запросов к этой папке. Таким образом оптимальное количество работающих параллельно папок ограничивается числом ядер процессора системы, на которой работает сервер.

Чтобы обеспечить работу с объемом данных, превышающим физически доступную оперативную память, был разработан класс, поддерживающий различные библиотеки для работы с жестким диском.

Хорошо себя зарекомендовало использование LSM-дерева (log-structured merge tree) для хранения и доступа к данным типа «ключ-значение» [5], вместо более распространённого B-дерева [6]. Поэтому для работы с жестким диском была выбрана СУБД RocksDB, основанная на этой структуре данных. В будущем она может быть заменена на разработанную под собственные нужды систему на основе такого подхода.

```
> create user_name    # создание пользователя
pass: *****           # ввод пароля
OK
> connect user_name 1 # подключение от пользователя user_name к базе данных '1'
pass: *****           # ввод пароля пользователя
OK
> set key value       # задание значения
OK
> get key             # получение значения
value
> disconnect          # отключение от базы данных '1'
OK
> kill 1              # удаление базы данных '1'
pass: *****
OK
>exit                 # выход из интерпретатора
```

Рис. 3. Простейший сценарий работы с разрабатываемой СУБД

Поскольку, данные для одной базы данных обрабатываются в одном потоке, то это обеспечивает персистентность данных, так как данные не могут быть изменены извне. В будущем, при добавлении поддержки транзакций эта функциональность может быть очень полезна.

Заключение

Предложенное решение по управлению данными на основе NoSQL использует наработки аналогов и лидеров на рынке, а также предоставляет простой интерфейс и конфигурирование, что может помочь при внедрении данной технологии.

Это решение в силу набора требований, положенных при проектировании, может быть полезным для широкого круга растущих компаний, работающих с большими данными и с большим числом запросов.

Несмотря на работающий прототип, продукт требует доработок, прежде чем сможет быть подвергнут нагрузочному тестированию и станет возможным провести полноценное сравнение данной работы с лидерами индустрии.

Список литературы

1. Клеппман, М. Высоконагруженные приложения. Программирование, масштабирование, поддержка / М. Клеппман. – СПб.: Питер, 2018. – 640 с.

2. Документация к серверу приложений и СУБД Tarantool [Электронный ресурс]. – Режим доступа : https://docs.tarantool.io/ru/doc/~latest/Tarantool-ru.pdf

3. Описание Unified Modeling Language [Электронный ресурс]. – Режим доступа : https://www.uml.org/what-is-uml.htm

4. Kabakus A.T., Kara R. / A.T. Kabakus, R. Kara // A performance evaluation of in-memory databases. Journal of King Saud University – Computer and Information Sciences, Volume 29, Issue 4, 2017. – P. 520-525, ISSN 1319-1578, https://doi.org/10.1016/j.jksuci.2016.06.007

5. Luo C., Carey M.J. / C. Luo, M.J. Carey // LSM-based storage techniques: a survey. The VLDB Journal 29, 2020. – P. 393–418 [Электронный ресурс]. – Режим доступа : https://doi.org/10.1007/s00778-019-00555-y

6. Graefe G., Kuno H. Modern B-tree techniques / G. Graefe, H. Kuno // IEEE 27th International Conference on Data Engineering. – Hannover, 2011. – P. 1370-1373, doi: 10.1109/ICDE.2011.5767956