Масштабируемая подсистема диалогов

Цель:

В результате выполнения ДЗ создан базовый скелет микросервиса, который будет развиваться в дальнейших ДЗ.

Проект находится тут:

https://github.com/maleykovichdim/my_chat_websocket_sharding

Запуск: docker-compose up --build

localhost:8086

Чат сервер реализован на языке golang, за основу взят gorilla web-socket server.

Папка: /chat

Запуск: ./start.sh

Чат клиент

создан для проверки работы сервера

Он отсылает и принимает сообщения

Папка: /chat_client

Запуск: go run main.go

Четыре Mysql server-а, запускаемые в docker-compose,

шардируют таблицу сообщений message.

```
mysql> show columns from message;
+--------------+-------------+------+-----+-------------------+----------------+
| Field        | Type        | Null | Key | Default           | Extra          |
+--------------+-------------+------+-----+-------------------+----------------+
| id           | int(11)     | NO   | PRI | NULL              | auto_increment |
| id_author    | int(11)     | NO   |     | NULL              |                |
| id_recipient | int(11)     | NO   |     | NULL              |                |
| text         | varchar(255)| NO   |     | NULL              |                |
| time         | timestamp   | NO   |     | CURRENT_TIMESTAMP |                |
+--------------+-------------+------+-----+-------------------+----------------+
5 rows in set (0.00 sec)
```

Запуск: docker-compose up --build

Id message на данный момент в каждой таблице независимо.

По факту было использовано консистентное хэширование по указанному ниже выведенному(вычисляемому) id диалога.

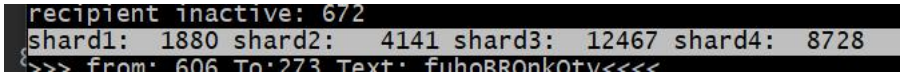
Основные идеи:

Группировать сообщения стоит по идентификатору чата и времени написания сообщения(Чтобы учесть эффект “Леди Гаги”). В данной версии шардирования использовался искусственно выведенный идентификатор чата:

```
var idChat string
if AUTHOR ID < RECIPIENT ID  {
        idChat = AUTHOR ID + "_" + RECIPIENT ID
} else {
        idChat = RECIPIENT ID + "_" + AUTHOR ID
}
```

Совокупность id автора и получателя монотонная последовательность, поэтому используем под капотом hashed ключ шардирования. Время отправки сообщений тоже монотонная последовательность.И можно было бы дополнительно распределять сообщения в каждой дополнительно созданной группе шардов по границе-порогу таймстампа.

Пример заполнения шардов сообщениями по рандомному потоку сообщений:



Так как само шардирование осуществляется на уровне приложения , то оно соответственно допускает возможность решардинга без даунтайма:

Функция GetShardDbBody возвращает результатом указатель на holder базы данных shard-a.

Функцию можно модифицировать для добавления особых условий.

Также можно добавлять новые базы данных в “кольцо” или удалять из контекста переставшие работать.

В консистентном хэшировании при добавлении новых нод вы перетасовываете только небольшую часть ключей.

При отключении шарда он должен быть удален из кольца и сообщения распределятся в другие шарды. Также если шард перегружен, то стоит добавлять новые шарды рядом с перегруженным и позаботиться о уже заполненных шардах на которых действовали старые правила, определяемые старым набором шардов, - возможно одним из решений является переход на новые шарды в группах с граничной меткой времени или включение правил привязанных к таймстампу, если запросы это позволят...