Вестник УГАТУ
Уфа : УГАТУ, 2012 Т. 16, № 3 (48). С. 143–152

УДК 004.65

Д. В. Павлов

РЕЛЯЦИОННАЯ РАСПРЕДЕЛЕННАЯ СИСТЕМА УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ С АВТОМАТИЧЕСКОЙ МАСШТАБИРУЕМОСТЬЮ

Излагается концепция построения новой реляционной распределенной системы управления базами данных с автоматической масштабируемостью. Для создания эффективной СУБД предлагается все данные хранить в оперативной памяти, применяя при этом новый метод индексации данных на основе трех линейных массивов (СТМ-индексация). Создаваемая СУБД наделяется свойствами автоматической масштабируемости и перераспределения данных. Для совместимости с существующими СУБД используется универсальный язык запросов SQL. Эффективность создаваемой СУБД проверялась экспериментально в сравнении с MySQL и на рассмотренном примере показала свою более высокую эффективность. *Система управления базами данных*; *метод индексации даны*; *реляционная*, *распределенная*, *СТМ-индексация*

ВВЕДЕНИЕ

Современные информационные системы работают с такими огромными потоками данных, что справиться с ними могут только много тысяч серверов. Настройка и администрирование этих серверов требуют огромных ресурсов: человеческих, денежных, энергетических и др. Разработка же программного обеспечения, которое хорошо распараллелится на эти сервера, занимает еще больше времени и ресурсов, и разрабатывать такое программное обеспечение приходится каждый раз сначала. Главной проблемой при этом является создание системы управления базой данных (СУБД).

Почти все интернет-гиганты: facebook.com, twitter.com, vkontakte.ru начинали с использования обычных, широко распространенных реляционных СУБД, например, таких как MySQL [1]. Затем с течением времени число пользователей данных сайтов стало таким, что использование парочки серверов с MySQL становилось невозможным и пришлось быстро увеличить количество серверов, а затем полностью перейти на какую-нибудь другую, уже более распределенную СУБД [2].

Сервис vkontakte.ru уже сейчас работает с 10000 серверов [3]. Очевидно, что администрирование такого массива серверов – это огромная проблема. Автоматизация этого процесса – задача, решение которой приносит огромную экономическую выгоду.

В настоящее время объемы оперативной памяти компьютеров возросли настолько, что для некоторых задач стало возможно хранить всю базу данных в оперативной памяти. Большинство основных СУБД спроектированы много лет назад, когда такое было невозможно, и поэтому они не оптимизированы для работы полностью в оперативной памяти.

Автором данной статьи, используя преимущества хранения информации в оперативной памяти, создается новая СУБД MFRDB – реляционная распределенная система управления базами данных с автоматической масштабируемостью, полностью функционирующая в оперативной памяти.

В данной СУБД используется новый метод индексации данных. Этот метод позволяет тратить на индексацию минимальные объемы памяти и поэтому позволяет существенно повысить эффективность создаваемой СУБД.

Созданная на сегодняшний день версия СУБД MFRDB реализует минимальную функциональность, позволяющую экспериментально проверять принятые решения и проводить экспериментальное сравнение возможностей СУБД MFRDB с возможностями широко применяющейся СУБД – MySQL.

1. КОНЦЕПЦИЯ СУБД MFRDB

При разработке новой СУБД MFRDB ставились следующие требования.

Администрирование СУБД должно отнимать минимум ресурсов. Идеальный вариант: запуск сервера, установка и запуск СУБД. И все. Данная сущность СУБД сама входит в распределенную сеть и дает знать о себе остальным сущностям распределенной системы.

Все данные хранятся в оперативной памяти. Скорость работы с оперативной памятью на порядок выше, чем с винчестером. В настоящее время не составляет проблем получить сер-

Контактная информация: victor.pavlov.1951@gmail.com

вер с 10–100 гигабайтами оперативной памяти. Таким образом, хранение всех данных в оперативной памяти стало возможным.

Индексы должны занимать минимум памяти. В связи с этим в качестве индексного метода выбирается не B-дерево, а собственная система индексирования, описанная в [4], которая занимает меньше оперативной памяти.

Автоматическое масштабирование. Система должна снимать с пользователя задачу изобретения способа увеличения пропускной способности системы. Все это должен сделать разработчик – добавление новых серверов в систему, и принятие мер для повышения производительности должно происходить автоматически.

Перераспределение данных. Распределенная система должна сама решать, какие данные хранить на каких серверах, сколько хранить копий и где их хранить. При этом система должна автоматически переносить данные с одного сервера на другой, чтобы запросы на получение данных проходили с максимальной скоростью.

Использование SQL. Подавляющее большинство новых разработок в сфере распределенных баз данных используют не реляционную модель хранения данных. И, как следствие, они предлагают новый (не SQL) интерфейс доступа к данным. Это вызывает множество проблем по переносу старых приложений на новую систему. Поэтому в MFRDB в качестве интерфейса доступа к данным выбран SQL.

2. СУЩЕСТВУЮЩИЕ СУБД И ИХ ОСОБЕННОСТИ

Рассмотрим коротко основные существующие системы управления базами данных.

MongoDB. MongoDB — документо-ориентированная система управления базами данных с открытым исходным кодом, не требующая описания схемы таблиц. Написана на языке C++ и распространяется в рамках лицензии AGPLv3. Есть репликация и репликационный лог. Возможна настройка системы мастер-слэйв: если мастер репликации перестал работать, среди слейвов через 10–20 секунд будет выбран новый. Реализован шардинг, работающий по диапазонам значений, а не по хешам, вычисляемым из значений. Диапазоны для каждого шарда задаются в конфигурационном файле вручную. Решардинг производится при помощи корректировки этих диапазонов. При этом записи на изменяемые шарды блокируются на время переноса. Индексы – некие деревья, похожие на B-tree. Индексы хранятся в оперативной памяти [5].

В MongoDB нет автоматического самонастраивающегося шардинга – масшатабируемости. Нет автоматического переноса связанных записей. Индексы хранятся в оперативной памяти, но, как и во многих других СУБД, в виде деревьев, занимающих много памяти.

MySQL. MySQL – свободная СУБД [6]. MySQL является решением для малых и средних приложений. Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа MyISAM, поддерживающие полнотекстовый поиск, так и таблицы InnoDB, поддерживающие транзакции на уровне отдельных записей.

MySQL является почти полной противоположностью предлагаемой в данной работе системы. В MySQL нет распределенности. Любой шардинг нужно делать на клиентской стороне. MySQL использует в качестве индексов B-деревья.

Oracle. Oracle Database или Oracle DBMS – объектно-реляционная СУБД. Oracle Database 11g – первая в мире база данных, разработанная специально для работы в сетях распределенных вычислений Grid, предназначенная для эффективного развертывания на базе различных типов оборудования, от небольших серверов до мощных симметричных многопроцессорных серверных систем, от отдельных кластеров до корпоративных распределенных вычислительных систем. СУБД предоставляет возможность автоматической настройки и управления, которая делает ее использование простым и экономически выгодным. Стоимость Oracle $41500 на 1 процессор.

Cassandra. В Cassandra реализована децентрализованность. Каждый сервер в системе идентичен, и поэтому в системе нет узкого места и единой точки падения системы. Cassandra используется такими гигантами как Facebook, Twitter, CloudKick, RackSpace, Reddit, Digg, Cisco, SimpleGeo. Самый большой кластер хранит 100 ТБ данных на 150 машинах.

В качестве индексов в Cassandra не используются B-деревья. Сделано это для того, чтобы при вставке новых записей не вносить сразу же изменения на жесткий диск, а сначала хранить часть изменений в оперативной памяти. За счет этого увеличивается скорость вставки новых записей. Но из-за этого уменьшается скорость чтения, так как для чтения приходится синхро-

низировать данные, находящиеся на жестком диске и в оперативной памяти [3].

Записи хранятся в сервере, определяемым ключом записи. Именно ключ записи определяет сервер, в котором будет хранится запись. При чтении записи по ключу легко вычислить сервер, на котором она записана, и получить запись. Записи с близкими ключами будут находится на 1 сервере. Таким образом, в Cassandra не реализовано хранение записей, близких друг к другу по смыслу [7].

По сути, в Cassandra все поисковые запросы идут по ключам, и таким образом вместо привычных индексов здесь необходимо дублировать информацию, создавая свои индексы путем создания дополнительных таблиц. На каждый индекс, который необходимо сделать, нужно создать новую таблицу и дублировать данные в нее.

Cassandra имеет общую черту с предлагаемой СУБД MFRDB системой: она хорошо масштабируется на вставку записей в систему и распределение нагрузки на множество серверов. Но данные в ней хранятся структурой «ключ-значение», не имеющей практически ничего общего с реляционной структурой. Поэтому там нет поддержки универсального языка запросов SQL.

3. ОТЛИЧИТЕЛЬНЫЕ ОСОБЕННОСТИ СУБД MFRDB

Разрабатываемая СУБД MFRDB имеет следующие отличительные особенности:

- в СУБД MFRDB в отличие от существующих СУБД MongoDB, MySQL, Cassandra реализован автоматический перенос связанных записей;
- в СУБД MFRDB применена новая система хранения индексов в отличие от деревьев, применяющихся в СУБД MongoDB, MySQL, Cassandra;
- в СУБД MFRDB реализована реляционная структура хранения данных, в отличие от применяющейся в СУБД Cassandra структурой «ключ-значение».

4. МОДЕЛЬ РАСПРЕДЕЛЕННОЙ СУБД MFRDB И ЕЕ ОСНОВНЫЕ ФУНКЦИИ

Используется реляционная модель, и поэтому каждая запись состоит из нескольких заранее определенных полей данных заранее определенного типа. Каждая запись имеет уникальное в пределах таблицы 64-битное целое поле ID – для того, чтобы унифицировать работу с данными по всей распределенной системе.

Каждая сущность MFRDB выполняет четыре роли сервера**:**

- *Хранение данных*. Непосредственно хранит сами записи.
- *Хранение информации о том, где хранятся данные*. Сервер хранит связь пары (Таблица, ID записи) и информации о сервере (IP-адрес, Порт), на котором хранится эта запись.
- *Роль индекса*. Хранение информации для быстрого поиска записей по индексу. В качестве индексного метода используется система трех массивов, описанная в [1].
- *Обработка SQL запроса*. Каждый сервер может получить SQL запрос от клиента или от другого сервера. Он должен его обработать (возможно, при этом послав запросы другим серверам) и вернуть ответ.

Каждая роль независима от другой. Возможно существование сервера, который вообще не выполняет некоторые роли, и это никак не должно отражаться на работе других ролей.

Все сервера общаются друг с другом. Они посылают друг другу сообщения о дееспособности – это нужно для поддержания актуальности информации о тех серверах, которые вышли из строя, для реорганизации работы распределенной системы. Если один из серверов обнаружил, что другой не работает, он должен сообщить об этом событии другим серверам.

Запросы от клиентского приложения посылаются на один из серверов СУБД случайным образом. Клиент может посылать запросы всегда только к одному серверу или каждый раз к разным. Это не должно влиять на получаемые результаты.

Каждый сервер хранит полученную конфигурацию всех остальных серверов и понимает свое место в этой схеме.

Для борьбы с ситуацией отключения некоторых серверов от распределенной системы у каждого сервера есть его полные копии на другом сервере. Количество копий должно регулироваться автоматически в зависимости от того, насколько часто требуются эти данные или насколько важны эти данные, насколько важно, чтобы они были постоянно доступны. Таким образом, при отключении одного из серверов данные, хранящиеся на нем, все равно будут доступны, так как существуют полные копии этого сервера.

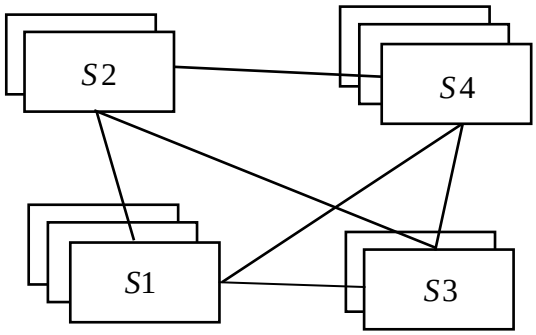

Рис. 1. Распределенная система из десяти серверов. Четыре разных сервера (*S*1, *S*2, *S*3, *S*4) и их копии

На рис. 1 изображена распределенная система из десяти серверов: четыре разных сервера (*S*1, S2, S3, S4) и их копии (две копии *S*1, одна копия *S*2, одна копия *S*3, две копии *S*4).

Клиент может обращаться к любому серверу в системе. Он может выбирать его случайным образом или работать только с одним из серверов. Клиент вообще может не знать о существовании других серверов, кроме того, с которым он работает. Все это никак не должно влиять на работоспособность системы.

Поиск записей. На каждом сервере хранится часть записей от всех записей БД. Каждый индекс будет разбит на большое количество маленьких индексов, и каждый индекс при поиске выдаст свой набор записей. Для поиска необходимо получить записи по всем диапазонам и еще раз отсортировать их вручную, если это необходимо.

Вставка записи. Вставка записи выполняется по следующему алгоритму:

Генерация *ID*, если он не задан в запросе на вставку.

Одновременно отправляем запрос вставки данных в копии случайного сервера *S_i*.

Одновременно отправляем запрос на вставку индексных данных в копии случайного сервера *S_k*.

На каждый запрос необходимо дождаться минимум *N* ответов.

Считаем, что запись вставлена.

При добавлении записи идентификатор должен генерироваться случайным образом — 64-битное целое число. Вероятность генерации повторного идентификатора в таком случае минимальна.

Число *N* – это минимальное количество ответов, которые мы получаем на наши запросы к другим серверам. *N* регулирует соотношение надежности системы и времени ее отклика.

Если число *N* велико, то есть уверенность, что записи вставились в большое число копий серверов, и, значит, данные не пропадут.

Если же *N* мало, то клиенту сообщается, что запись вставлена, но возможны ситуации, что запросы не дошли до некоторых серверов. Те же сервера, до которых запросы дошли, могут отключиться от сети, и вставленные данные окажутся недоступными.

5. АВТОМАТИЧЕСКОЕ МАСШТАБИРОВАНИЕ

Для повышения быстродействия и надежности СУБД MFRDB применяется ее автоматическое масштабирование:

- если какие-то записи часто запрашиваются на чтение, то увеличивается количество копий этого сервера;
- если происходит частая вставка, то увеличивается количество серверов.

Это связано с тем, что:

- при чтении выбираем случайную копию сервера, поэтому нагрузка снижается пропорционально количеству копий;
- если же основная нагрузка идет на запись, то нужно добавить не новую копию сервера, а новый сервер. Тогда часть нагрузки на запись перепадет на него и на его копии.

Полезно учитывать, какие записи часто считываются вместе, и размещать их на одном сервере. Например, лучше, чтобы книги одного автора хранились рядом и вместе с информацией об этом авторе. Чтобы оценки определенного человека хранились рядом и вместе с данными об этом человеке. В таком случае запрашивающий сервер (обрабатывающий SQL запрос) сделает меньше обращений. В идеале – одно к нужному серверу.

Тестирование автоматического масштабирования в СУБД MFRDB

Для тестирования функции автоматического масштабирования СУБД MFRDB проведен эксперимент 1.

Эксперимент 1. В первом эксперименте разрабатываемая СУБД тестировалась при вставке 1000 записей о книгах.

Записи имеют следующую структуру:

ID – уникальный идентификатор,

Caption – текстовое поле (строка максимальной длины 255 символов),

Year – год издания (целое положительное 32-битное число),

Language_ID – язык, на котором написана книга (целое положительное 32-битное число).

Выполнено ряд запусков при варьировании двумя параметрами:

- количеством клиентов, работающих с СУБД,
- количеством серверов в распределенной СУБД.

Результаты испытаний представлены в табл. 1 и 2.

Таблица 1

	Количество клиентов			
Количество серверов	11	22	33	44
1	62	62	62	62
2	37	35	35	34
3	27	26	26	25
4				19
5				16
6				14
7				12
8	15	13	12	11
9			11	10
10			10	9
11				9
12				8
13	11			7
14				7
15				7

В данном эксперименте на одном сервере было запущено одновременно несколько их версий СУБД, так что каждая из сущностей не полностью загружала процессор. Это позволило имитировать работу нескольких очень медленных реальных серверов.

В процессе экспериментов регистрировалось:

- **количество обработанных запросов в секунду** – общее число запросов (1000 штук), деленное на время выполнения их вставки;
- **затраты на 1 запрос** (Сервера × Секунды) – количество серверов, умноженное на время выполнения вставки и деленное на число запросов (1000 запросов).

Таблица 2

Количество серверов	Количество обработанных запросов в секунду	Затраты на 1 запрос (Сервера × Секунды)
1	16	0.064
2	29	0.064
3	40	0.075
4	52	0.076
5	62	0.080
6	71	0.084
7	83	0.084
8	90	0.088
9	100	0.090
10	111	0.090
11	111	0.099
12	125	0.096
13	142	0.091
14	142	0.098
15	142	0.105

Как видно из таблиц, с увеличением количества серверов уменьшается время выполнения запроса и увеличивается пропускная способность.

Анализ результатов тестирования автоматического масштабирования

1. Результаты эксперимента 1 свидетельствуют о том, количество параллельно работающих клиентов не сильно увеличивает скорость выполнения. Это свидетельствует о том, что расстояние между клиентом и сервером небольшое, и они быстро обмениваются информацией, а также о том, что распараллеливание задачи внутри сервера работает достаточно хорошо.

2. Выяснилось, что с увеличением количества серверов уменьшается время выполнения запросов и, следовательно, увеличивается пропускная способность − количество обработанных в секунду запросов. Это доказывает, что успешно выполняется первоначальная задача – автоматическое масштабирование распределенной сети. Так, при простом добавлении в систему дополнительных серверов, увеличивается ее пропускная способность без какой-либо дополнительной настройки и без добавления еще ка-

кой-то логики в клиент. Клиент считает, что работает с одним-единственным сервером, то есть для него эта распределенная система выглядит черным ящиком.

3. Начиная с некоторого момента, увеличение количества серверов не приводит к росту производительности и наблюдается повышение затрат на выполнение одного запроса. Это объясняется тем, что в эксперименте не использовались реальные серверы, а запускались на одном реальном сервере несколько копий СУБД. В итоге, в какой-то момент происходит ограничение скорости обработки по мощность процессора на данном сервере и возрастание накладных расходов по обработке большого количества параллельно работающих потоков.

6. АВТОМАТИЧЕСКОЕ РАСПРЕДЕЛЕНИЕ ЗАПИСЕЙ

Уникальной функцией создаваемой распределенной СУБД MFRDB является автоматическое перераспределение записей по серверам для хранения. При вставке записей они вставляются случайным, хаотическим образом. При этом некоторые записи являются связанными друг с другом смысловым образом. Например, если хранятся записи книг, то по смыслу можно выделить множество связанных групп:

- книги, написанные Львом Толстым:
- книги, вышедшие в июле 2011 года;
- книги, прочитанные Павловым Дмитрием;
- книги, прочитанные пользователями сайта более 1000 раз и т. д.

При использовании базы данных запросы, запрашивающие некоторые подмножества книг, зачастую запрашивают книги из определенной группы. При этом некоторые группы запрашиваются чаще, а некоторые реже.

Суть проблемы

Если при запросе некоторой группы книг записи этих книг находятся на различных серверах, то серверу, выполняющему запрос, придется запросить записи книг, обратившись к нескольким серверам. И чем на большем количестве серверов будут распределены записи из этой группы книг, тем к большему числу серверов придется обратиться серверу, выполняющему запрос, тем самым породив большое число операций общения в распределенной сети, а следовательно, и торможение работы сети.

Если же все книги из запрашиваемой группы будут храниться на одном сервере $S1$, то возможны две ситуации. Либо будет выполнено 0 операций общения между серверами – если запрос на выборку книг выполняет тот же сервер $S1$. Либо будет выполнена 1 операция общения между серверами – если запрос выполняет какой-либо другой сервер $S2$.

Данная ситуация является идеальной с точки зрения минимизации общения серверов друг с другом, и именно – в режиме приближения к этой ситуации будет работать разрабатываемая система.

Также необходимо учитывать, что тривиальным решением задачи было бы размещение вообще всех записей на одном сервере. Но такое решение неприемлемо по нескольким причинам:

- возможна ситуация, когда для хранения всех записей чисто физически не хватит памяти одного сервера;
- размещение всех записей на одном-единственном сервере противоречит самой концепции распределенной системы, так как все записи в одном месте хранить ненадежно из-за того, что при отключении сервера пропадет доступ ко всем данным, а не лишь к какой-то части данных;
- один сервер может не справиться с нагрузкой, то есть не сможет вовремя обработать все запросы чтения и изменения данных.

Таким образом, имеет смысл на всех серверах в распределенной системе размещать примерно равное количество записей.

Необходимо отметить, что не всегда возможно идеально распределить все записи таким образом, чтобы при запросах нужные записи всегда оказывались на одном определенном сервере.

В качестве примера рассмотрим следующую ситуацию.

Есть два сервера: $S1$, $S2$ и две записи: $R1$, $R2$. Для того, чтобы выдержать вышеуказанное ограничение, нужно, чтобы на всех серверах в системе было примерно равное число записей. Поэтому возможны два варианта распределения записей:

$$S1: R1, S2: R2;$$

$$S2: R2, S2: R1;$$

В обоих случаях при запросе множества записей $\{R1, R2\}$ необходимы записи с обоих серверов. Таким образом, не всегда можно распределить записи так, чтобы при запросе необхо-

димых множеств использовался только один сервер. Но нужно стремиться к тому, чтобы в среднем было наименьшее количество обращений между серверами. Если в системе N серверов, то идеально будет, если среднее число обращений в лучшем варианте AVG_{best} будет равно

$$AVG_{\text{best}} = \frac{0 + 1 \cdot (N-1)}{N} = 1 - \frac{1}{N}, \qquad (1)$$

и наихудшей ситуацией будет, если количество обращений между серверами AVG_{worst} в среднем равно

$$AVG_{\text{worst}} = \frac{N \cdot (N-1)}{N} = N - 1. \qquad (2)$$

Решение проблемы

В системе было принято решение, что примерно равное количество записей − это если количество записей на сервере с наименьшим количеством записей, поделенное на количество записей на сервере с наибольшим количеством записей, больше 0,75.

Таким образом, ставится задача исключить, например, ситуацию, когда на одном из серверов в два раза больше записей, чем на другом.

В создаваемой системе каждый сервер $S1$ в распределенной сети время от времени выполняет выборки записей, и при выборке подмножества записей {$R1$, $R2$, $R3$,…, R_K} сервер учитывает, сколько с каждого сервера распределенной системы было считано записей:

Серверы	S_1	S_2	S_3	…	S_N
Количество записей	X_1	X_2	X_3	…	X_N

Общее количество считанных записей

$$K = X_1 + X_2 + X_3 + ... + X_N.$$

Затем сервер $S1$ находит сервер $S_{\max}$, с которого было считано максимальное количество $X_{\max}$ записей:

$$X_{\max} = \max(\{X_1, X_2, X_3, ..., X_N\}).$$

Далее сервер $S1$ делает пометку в таблице пересылок, что, возможно, записи из выбранной группы, находящиеся на нем, желательно перенести на сервер $S_{\max}$, так как там находится максимальное число записей из выбранной группы. Пометка делается увеличением соответствующего числа C на 1 в таблице пересылок.

Таблица пересылок выглядит следующим образом:

Запись	Будущий сервер	Количество пометок
R_i	S_x	C_1
R_j	S_y	C_2
…	…	…
R_k	S_z	C_q

При этом таблица отсортирована по количеству пометок. То есть $C_1 \geq C_2 \geq \ldots \geq C_q$. Пересылке подлежат в первую очередь записи, находящиеся наверху таблицы.

Через каждый определенный временной интервал сервер принимает решение о пересылке нескольких записей на другой сервер. Сначала собирается информация со всех серверов о количестве записей на них и проверяется, нужно ли вообще с этого сервера отправлять записи. Если на этом сервере меньше записей чем 0,75 от количества записей на сервере с максимальным количеством записей, то записи не перенаправляются. Если же записей на данном сервере нормальное количество, то выбираются записи для пересылки. Это записи, набравшие к этому моменту наибольшее число пометок о том, что их желательно перевести на другой определенный сервер.

Тестирование автоматического распределения памяти в СУБД MFRDB

Для тестирования функции автоматического распределения памяти в СУБД MFDB проведен эксперимент 2.

Эксперимент 2. В эксперименте 2 рассматривается распределенная система из двух серверов: $S1$, $S2$; и двух таблиц: Таблица 3 − «Люди», и Таблица 4 − «Друзья».

В таблице 3 содержится информация о людях, записанных в базе данных. В таблице 4 − информация о дружеских связей между людьми.

Для тестирования много раз выполнялся следующий SQL запрос:

```
SELECT * FROM Друзья
        WHERE Друзья.Человек = 'X',
```

где X − случайное число от 1 до 8. Запрос посылался случайным образом то к серверу $S1$, то к серверу $S2$. При этом записывалось на каждом сервере количество запросов к нему (RC) и количество операций обращения к другому серверу (SC), которые он будет делать, выполняя SQL запрос.

Таблица 3

Люди	
ID	Имя
1	Дима
2	Костя
3	Антон
4	Леня
5	Вадим
6	Егор
7	Сережа
8	Паша

Таблица 4

Друзья		
ID	Человек	Друг
1	1	2
2	1	3
3	1	4
4	2	1
5	2	3
6	2	4
7	3	1
8	3	2
9	3	4
10	4	1
11	4	2
12	4	3
13	5	6
14	5	7
15	5	8
16	6	5
17	6	7

На рис. 2 стрелками обозначено, кто из людей с кем дружит. Каждая стрелка записана в табл. 4.

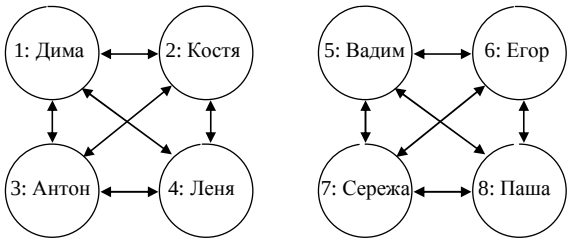

Рис. 2. Схема дружеских связей

На рис. 3 показана зависимость отношения *SC* / *RC* от *RC*.

Из рис. 3 видно, что в начале среднее количество операций обращения к другому серверу равно максимальному возможному MAX = 1, так как максимальное количество обращений, определяемое формулой (2) при количестве серверов, равном двум (N = 2), соответственно, равно:

$$MAX = N - 1 = 2 - 1 = 1.$$

Затем, по ходу выполнения случайных операций выборки записей из таблицы Друзья, записи перераспределились по серверам, и уже при числе обращений RC = 5000 (lg(RC) = = lg 5000 = 0,699) количество операций обращения к другим серверам приближается к минимальному (рис. 3), определяемому при N = 2 по формуле (1):

$$MIN = 1 - 1/N = 1 - 1/2 = 0{,}5.$$

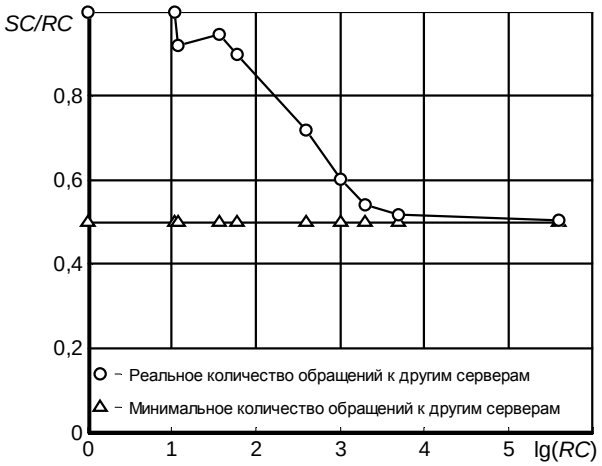

Рис. 3. Среднее реальное и среднее минимально возможное количество обращений к другим серверам

7. СИСТЕМА ТРЕХ МАССИВОВ (СТМ)

В данной работе предложена новая концепция индексирования элементов базы данных – СТМ-концепция, на основе которой построены новые алгоритмы функционирования базы данных [4].

Для хранения данных при СТМ-концепции предлагается использовать три массива.

Это может быть три вектора, три области памяти и т. д. Принципиальным моментом является то, что должно быть три элемента для хранения данных.

Будем для считать, что у нас есть три массива данных. Пронумеруем их как 0, 1, 2.

В 0 массиве хранятся данные, которые не изменяются, то есть из этого массива производится только чтение, и в этом массиве нельзя выполнять изменения. Основная же работа происходит с массивом 1.

Операции, выполняемые СУБД MFRDB с элементами массивов

Добавление. Новые данные добавляются в массив 1.

Удаление. В массиве 1 необходимо поставить пометку, что данные удалены, при этом физическое удаление данных не обязательно, но возможно, если они находятся в массиве 1.

Изменение. Если эти данные располагаются в массиве 1, то возможно их изменить прямо там же, если они располагаются в другом массиве, то мы делаем операцию удаления старых данных, затем вставку новых.

Поиск. Поиск данных производится следующим образом: сначала в массиве 1, потом в массиве 2, и потом в массиве 0.

Слияние. Когда возникает ситуация, что в массиве 1 заканчивается место, то запускается процесс слияния.

Процесс слияния всегда запускается в параллельном потоке, чтобы была возможность продолжать работать с данными. Меняются местами массивы 1 и 2. Создается новый массив, размер которого равен сумме массивов 0 и 2 массивов. Начинается процесс копирования информации из массивов 0 и 2 в новый массив. После того, как копирование завершено, массивы 0 и 2 удаляются. Новый массив встает на место массива 0. На месте массива 2 создается еще один новый массив. Пока идет слияние, попрежнему остается возможность работы с данными, так как массив 1 свободен и для чтения и для записи, а массивы 0 и 2 доступны для чтения.

Таким образом, всегда есть возможность циклически переходить к состоянию до момента переполнения массива 1. Сразу же, как массив 1 переполняется, вновь запускается процесс слияния.

Нужно понимать, что не должно произойти повторного заполнения массива 1 до тех пор, пока не закончилась операция слияния массивов 0 и 2. Если же такое произошло, то новые операции на вставку перестанут выполняться, пока не закончится первое слияние и не появится возможность начать второе слияние.

В изложенном выше тексте заключается основная суть СТМ-концепции.

При каждом конкретном ее применении алгоритм может немного отличаться от предложенного выше.

Сравнение создаваемой СУБД MFRDB с существующими аналогами

Разрабатываемая СУБД MFRDB сравнивалась с известной СУБД MySQL. В СУБД MFRDB используется СТМ-индексация с тремя таблицами, а в СУБД MySQL в качестве индексации применяется B-дерево.

Для сравнительного сопоставления данных СУБД была взята база данных, состоящая из 125 тысяч книг. При тестировании СУБД была использована следующая таблица *Bolero*.

Таблица *Bolero*:

ID – целое, уникальный идентификатор;

ISBN – строка длины до 255 символов;

Name – строка длины до 65 тысяч символов;

PH – строка длины до 255 символов

Price – 16-битное целое.

Данная база данных обрабатывалась вновь создаваемой СУБД MFRDB и известной СУБД MySQL.

Первый результат экспериментов – это объемы занимаемой памяти для хранения данных базы данных и индексов базы данных.

Выяснилось (табл. 5), что СУБД MFRDB тратит оперативной памяти на сами данные примерно также, как MySQL, но при этом на хранение индекса уходит почти в 2 раза меньше оперативной памяти. При этом с увеличением количества индексов в базе данных будет усиливаться преимущество по затратам оперативной памяти предлагаемого метода индексации в MyDB по сравнению с методом индексации в MySQL.

Таблица 5

СУБД	Данные (МБ)	Индекс (МБ)
MySQL	12,7	4,3
MyDB	12	2,4

В эксперименте выявлена зависимость между количеством добавленных книг N и временем их добавления t, из которой видно, что затраты времени t на добавление книг СУБД MFRDB существенно ниже по сравнению с СУБД MySQL (рис. 4).

Для сопоставления временных затрат на операцию добавления разными СУБД определялось относительное время записи в СУБД MFRDB по отношению ко времени записи в СУБД MySQL– t_{MyDB} / t_{MySQL}, где t_{MyDB} – время добавления книг в СУБД MFRDB, t_{MySQL} – время добавления в СУБД MySQL.

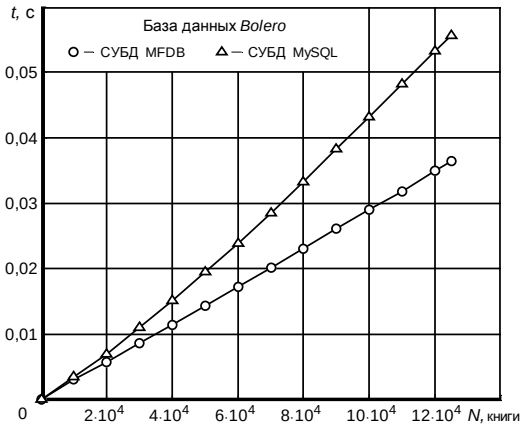

Рис. 4. Изменение времени записи t в зависимости от числа книг N для СУБД MFRDB и MySQL

Зависимость величины $t_{\text{MyDB}} / t_{\text{MySQL}}$ от числа добавляемых книг представлена на рис. 5, где видно, что с ростом количества добавляемых книг N, затраты времени t_{MyDB} в СУБД MFRDB существенно снижаются по сравнению с СУБД MySQL, и при $N = 1{,}25 \cdot 10^5$ книг составляют только $\approx$ 65 % от времени t_{MySQL} для СУБД MySQL.

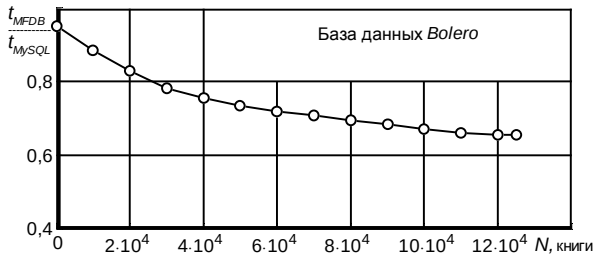

Рис. 5. Изменение относительного времени $t_{\text{MyDB}} / t_{\text{MySQL}}$ в зависимости от числа книг N для СУБД MFRDB и MySQL

ВЫВОДЫ

1. Предложена новая концепция распределенной СУБД с уникальными свойствами, базирующаяся на новых решениях автоматической масштабируемости СУБД, автоматического распределении записей и индексации таблиц в базе данных.

2. Новое решение проблемы автоматической масштабируемости СУБД базируется на предложенном в работе алгоритме рационального выбора серверов и их копий.

3. Разработан алгоритм перераспределения записей в распределенной системе для уменьшения накладных расходов, базирующийся на предлагаемой в работе схеме перераспределения записей между серверами в зависимости от частоты запросов определенных групп записей и степени заполнения серверов.

4. Предложен новый метод индексации таблиц в базе данных (CTM-индексация), ориентированный на работу в оперативной памяти и позволяющий существенно снизить объем требуемой памяти для хранения как самих данных, так и их индексов по сравнению с традиционными методами индексации.

5. Создано и отлажено программное обеспечение, на основе которого экспериментально проверены предлагаемые концепции и методы и показано, что автоматическое масштабирование, распределение записей и метод CTM-индексации позволяют построить СУБД MFRDB, характеризующуюся меньшими затратами памяти и большей скоростью функционирования по сравнению с широко распространенной СУБД MySQL.

СПИСОК ЛИТЕРАТУРЫ

1. **Дьюба П**. MySQL. Полное и исчерпывающее руководство по применению и администрированию баз данных MySQL 4, а также программированию приложений. Вильямс, 2004. 1056 с.

2. http://www.facebook.com/note.php?note_id=24413138919. Cassandra – A structured storage system on a P2P Network

3. Архитектура Вконтакте http://www.insight-it.ru/masshtabiruemost/arkhitektura-vkontakte/

4. **Павлов Д. В**. Новый метод индексации данных для системы управления базами данных, хранящихся в оперативной памяти// Сб. тр. VIII Всероссийской научно-практической конференции студентов, аспирантов и молодых ученых «Молодежь и современные информационные технологии», Ч. II. Томск, СПБ Графикс, 2010.

5. http://habrahabr.ru/blogs/nosql/119703/

6. **Дейт К. Дж.** Введение в системы баз данных. Вильямс, 2006. 1328 с.

7. http://www.parleys.com/#st=5&id=1866

ОБ АВТОРЕ

Павлов Дмитрий Викторович, асп. каф. компьютерн. математики. Дипл. математик-программист по математическ. обеспечению и администрированию инф. систем (УГАТУ, 2009). Иссл. в обл. построения СУБД.