

Распределенная отказоустойчивая СУБД

77-30569/339624

03, март 2012

Ковега Д. Н., Крищенко В. А.

УДК 004.65

МГТУ им. Н.Э. Баумана
mstu@sevik.ru

Введение.

В настоящее время реляционные СУБД де-факто являются стандартным средством хранения и управления данными. РСУБД подходят для широкого спектра задач, которые не требуют операций над большим количеством информации.

Инструментальные средства для реляционных баз данных не справляются с высокими запросами к масштабированию. Создание масштабируемых систем на реляционных базах данных требует шардинга, балансировки нагрузки, управления кластером, заботы о целостности данных, самостоятельной реализации распределённых запросов и других программных прослоек.

Принятые практики масштабирования в высоконагруженных системах превращают реляционную базу данных в нереляционную.

Также стоит отметить, что многие проекты имеют высокую динамичность данных. Сущности системы часто меняют структуру, похожие сущности могут немного отличаться набором данных, а некоторые из уже существующих могут со временем “обрастать” новыми свойствами. Из-за строгой структуры данных в базе, любые изменения схемы сущностей нужно отражать в структуре таблиц, постоянно усложняя модель (например, в случае наследования) и проводить миграцию существующих данных. При этих изменениях необходимо менять SQL-запросы или отображения объектов, к тому же скорость доступа к данным может существенно измениться. В итоге — необходимо поддерживать две разнородные модели — реляционную в базе и объектную в программном коде. Также, многие проекты не требуют строгой поддержки ACID-принципов. Намного важнее представляется быстрая обработка данных с минимальным временем отклика, а также быстрая и простая горизонтальная масштабируемость. Кроме того, иногда объем

обрабатываемых данных настолько велик, что единственным возможным путем работы является параллельное её решение на кластере.

Типы современных нереляционных СУБД.

Современные нереляционные СУБД в литературе принято называть *NoSQL*. Под термином *NoSQL* скрывается большое количество продуктов с различными архитектурными решениями (см. [6]). Поэтому выделим основные концептуальные отличия *NoSQL* систем от РСУБД:

- *NoSql* системы не являются реляционными;
- *NoSql* системы являются распределенными;
- Доступ к данным в *NoSql* системах как правило осуществляется по средствам простого API.

Таким образом, данный подход, как правило, используются в задачах, требующих обработку большого количества информации и в системах с жесткими требованиями ко времени выполнения запросов.

NoSQL системы не являются строго структурированными (*schema-less*). Вместо этого данные представляются коллекциями документов, а в некоторых системах и как наборы ключ-значение. Структура каждого документа может быть индивидуальной. Поддержка документов разной структуры ложится на плечи логики приложения (см. [5]).

Для осуществления приемлемой скорости операций над данными системы реализуют простую горизонтальную масштабируемость. Добавление сервера в кластер осуществляется с минимальным количеством действий со стороны администратора системы. Большинство *NoSQL* систем мультиплатформенны, что позволяет собрать распределенное хранилище из практически любого набора серверов.

В качестве API доступа к данным используется технология *MapReduce* (см. [2]). Таким образом концепция *NoSQL* соответствуют поставленным заданиям и ее следует использовать при реализации СУБД.

NoSQL системы принято различать по моделям хранения данных: ключ-значение, документно-ориентированная модель, колоночно-ориентированная модель, графовая модель.

Методы обеспечения отказоустойчивости.

При разработке и последующей эксплуатации систем с повышенным значением надежности необходимо наличие решений, повышающих вероятность

того, что система в момент времени t будет находиться в работоспособном состоянии. Такая вероятность называется доступностью системы.

Существует два основных направления при построении отказоустойчивых систем. Первый способ — использование только отказоустойчивых компонентов. При реализации этого направления каждый компонент системы может продолжать свое функционирование, даже если один/несколько подкомпонентов системы, выходят из строя. Второй способ разработка методов, гарантирующих построение отказоустойчивой системы из компонентов, не являющихся отказоустойчивыми. В таких системах отказоустойчивость реализована за счет введения избыточности и разработки специального программного обеспечения, элементных взаимосвязей и алгоритмов функционирования.

Внесение отказоустойчивости в систему или отдельно взятый компонент всегда нуждается в появлении некоторой избыточности. Избыточность — это наличие в структуре устройства возможностей сверх тех, которые могли бы обеспечить его нормальное функционирование. Избыточность вводится для повышения надёжности работы и для исключения влияния на достоверность передаваемой информации помех и сбоев. В основном используется четыре основных вида избыточности.

а) Аппаратная избыточность. Существуют методы постоянного резервирования (синтез избыточных устройств, нечувствительных к определенному количеству ошибок) и методы резервирования замещением (использование системы контроля, которая может действовать непрерывно или периодически).

б) Программная избыточность используется для контроля и обеспечения достоверности наиболее важных решений по управлению и обработке информации. Она заключается в сопоставлении результатов обработки одинаковых исходных данных разными программами и исключении искажения результатов, обусловленных различными аномалиями.

в) Информационная избыточность наиболее присуща телекоммуникационным системам, в которых информация передается многократно. Информационная избыточность заключается в дублировании накопленных исходных и промежуточных данных.

г) Временная избыточность заключается в использовании некоторой части производительности компьютера для контроля над исполнением программ и восстановления вычислительного процесса.

С точки зрения разработки СУБД актуальными являются методы информационной избыточности и временной. Первое обеспечивается за счет репликаций, второй — за счет использования супервизоров управления процессами.

Выбор и обоснование среды разработки.

Так как СУБД должна поддерживать большое число параллельных подключений, то в качестве языка разработки было решено использовать *Erlang*. *Erlang* — это язык программирования общего назначения и среда исполнения. Среди наиболее важных характеристик *Erlang* следует отметить следующие (см. [1]):

- 1 Функциональный язык программирования с неизменяемыми переменными (см. [7]).
- 2 Параллельная обработка и передача сообщений. Вместо использования распространенной в настоящее время модели параллельного программирования, в которой используются потоки с разделяемой памятью, *Erlang* поддерживает параллельную модель, основанную на легковесных процессах с асинхронной передачей сообщений (см. [4]).
- 3 Встроенные средства реализации распределенных приложений.
- 4 Средства обеспечения надежности системы. На низком уровне процессы могут быть связаны между собой и оповещаться посредством сообщений при завершении связанного процесса. На более высоком уровне, при использовании *OTP*, появляется возможность устанавливать различные политики мониторинга отдельных процессов и групп процессов. В распределенной системе резервные узлы могут автоматически заменять узлы, вышедшие из строя.

5

Общая архитектура разрабатываемого комплекса.

На рис. 1 показана структура разрабатываемого программного комплекса, а также взаимодействие различных подсистем. Для хранения больших объемов данных в системе предусмотрена возможность разделения данных по кластерам репликаций. Администратору системы предоставляется возможность управления разделением данных по кластерам репликаций, задавая хэш-функцию принадлежности. Также для избежания появления “узких” мест при работе с данными и обеспечения отказоустойчивости, в систему вводятся агенты доступа, предоставляющие клиентам унифицированный интерфейс для работы с СУБД и осуществляющие контроль выполнения запросов.

Представление данных

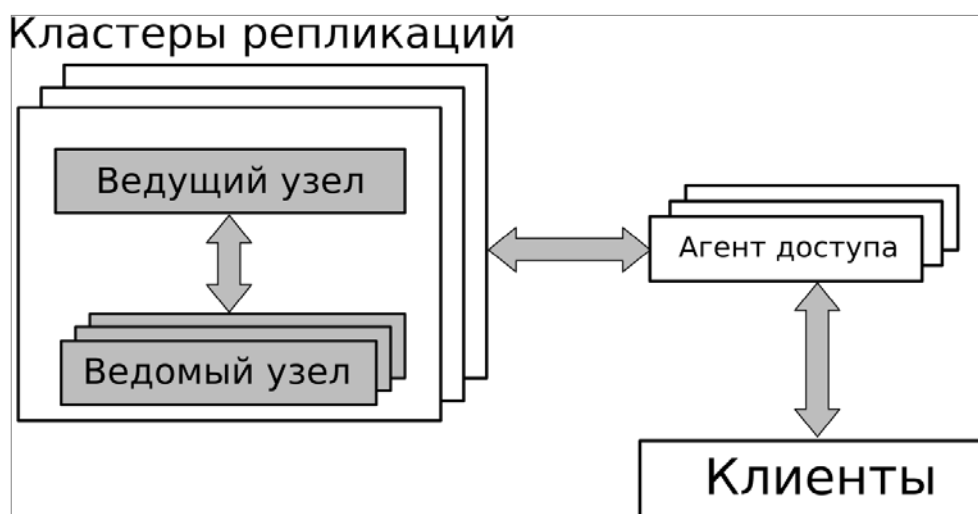


Рис. 1. Общая архитектура системы

Записи в СУБД представляют собой документы в терминах *NoSQL*. Документ представляет собой пару ключ-значение. Ключом могут быть данные следующих типов:

- Строки;
- Числа;
- Плоские списки;

Значение может представлять собой иерархическую структуру, состоящую из списков, строк, чисел. Таким образом, отсутствие ссылок в системе компенсируется хранением сложных структур данных. Для логического разделения данных в СУБД используются таблицы. Документ может принадлежать только одной таблице. При операции над данными пользователь должен явно указывать, с какой коллекцией происходит работа.

Для каждой таблицы в системе необходимо задать функцию принадлежности. Таким образом, возможны схемы развертывания системы, в которых таблицы распределены между кластерами репликаций по различным правилам. Данная возможность может быть использована для балансировки нагрузки.

Нагрузочное тестирование.

Для проверки масштабируемости разработанной системы разработана программа, осуществляющая нагрузку системы необходимым типом запросов. Средство тестирования позволяет регулировать количество потоков, выполняющих

запросы, число запросов, которое необходимо выполнить, и задавать адрес агента доступа, для каждого узла, участвующего в тестировании. Так как средство тестирования позволяет производить генерацию нагрузки с нескольких источников в сети, то возможно исключение влияния недостаточной мощности компьютера, генерирующего нагрузку, на результат тестирования.

При проведении тестирования применяются следующие варианты развертывания СУБД:

- а) Количество агентов доступа в системе (от 1 до 2);
- б) Изменение количества кластеров репликаций (от 1 до 5);
- в) Изменение состава кластеров репликаций — из одного ведущего узла или из ведущего узла и ведомого.

База данных состоит из одной коллекции с ключом типа целое число. Все запросы выполняются для данной коллекции. Исходя из типа ключа, функция принадлежности для коллекции задается следующим образом:

```
fun ( Key ) ->  
    Xrem <Количество репл. множеств>  
end .
```

Таким образом, использование равномерного закона распределения при генерации случайного ключа для формирования запроса, позволяет создать эквивалентную нагрузку на каждом кластере репликаций. Запрос на запись состоит из ключа и списка из четырех целых случайных целых.

График зависимости времени выполнения 200000 запросов на запись от конфигурации СУБД, при использовании двух агентов доступа, приведен на рис. 2.

Из графика видно, что использование нескольких узлов в кластере репликаций отрицательно сказывается на скорости выполнения операции записи, но регрессия по времени в зависимости от количества кластеров репликаций является практически константой и составляет порядка 70 мс для одного запроса при данной конфигурации сети. Система при правильной конфигурации позволяет достичь линейного роста скорости выполнения запросов в зависимости от числа кластеров репликаций (или узлов в кластере для операции записи). При росте числа кластеров репликаций агент доступа становится узким местом в системе и дальнейшее добавление кластеров не влияет на пропускную способность системы. Решением данной проблемы является добавление дополнительных агентов доступа.

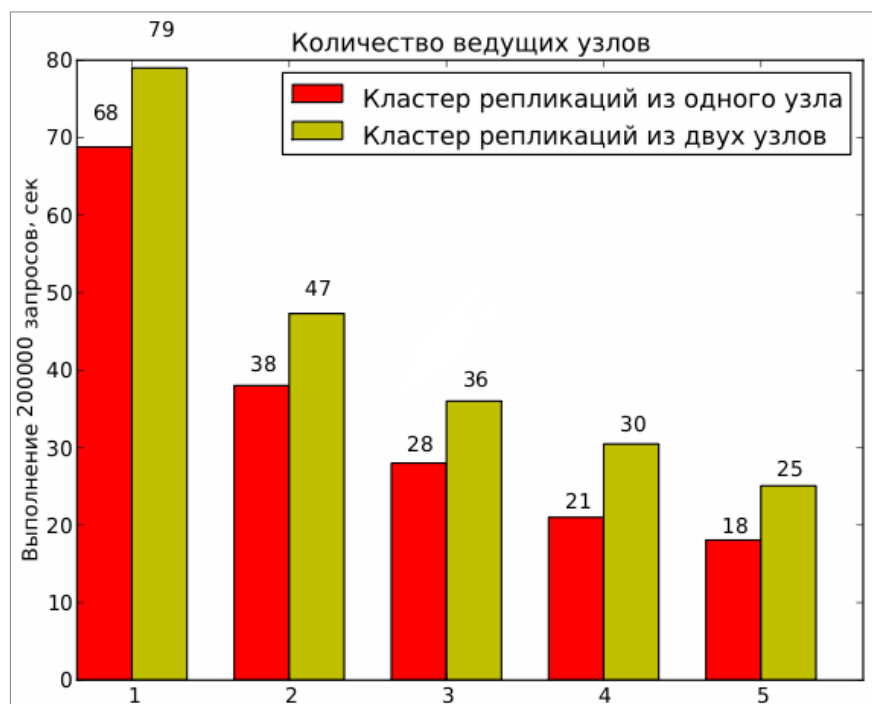


Рис. 2. Система с двумя агентами доступа

Заключение.

В результате проделанной работы была достигнута основная цель — была разработана распределенная отказоустойчивая СУБД. Тестирование позволило выявить, что программный комплекс позволяет достичь линейного роста скорости выполнения запросов в зависимости от числа кластеров репликаций (или узлов в кластере для операции записи). На основе проведенного тестирования были выработаны рекомендации по развертыванию системы.

Литература

1. Armstrong Joe. Programming Erlang: Software for a Concurrent World. — Pragmatic Bookshelf, 2007.
2. Dean Jeffrey, Ghemawat Sanjay. MapReduce: A Flexible Data Processing Tool // Communications of the ACM. — 2010. — Vol. 53, no. 1. — Pp. 72–77.
3. Gilbert Seth, Lynch Nancy. Brewer's Conjecture and the Feasibility of Consistent Available Partition-Tolerant Web Services // In ACM SIGACT News. — 2002. — pp. 51–59.
4. Gittleman Art. Multicore Computing Using Erlang. // FECS / Ed. By Hamid R. Arabnia, Azita Bahrami, Victor A. Clincy. — CSREA Press, 2009. — Pp. 261–266.

5. J. Chris Anderson, Jan Lehnardt, Noah Slater. CouchDB: Time to relax. — O'Reilly Media, 2009.
6. Leavitt Neal. Will NoSQL Databases Live Up to Their Promise? // Computer. — 2010. — Vol. 43. — Pp. 12–14.
7. Okasaki Chris. Purely Functional Data Structures. — New Ed edition. — Cambridge University Press, 1999. — July. — P. 220. Тема: Определение марки автотранспортного средства по фотографии на основе эмблемы.

Distributed fault-tolerant DBMS

77-30569/339624

03, March 2012

Kovega D.N., Krischenko V.A.

Bauman Moscow State Technical University

mstu@sevik.ru

The authors consider distributed fault-tolerant DBMS based on the Erlang runtime environment. As the result of testing, it was determined that the bundled software allowed achieving almost a linear growth of speed of query execution depending on the number of replication clusters. Recommendations on system deployment were produced.

Publications with keywords: [replication](#), [cluster](#), [non relational database](#), [NoSQL](#), [erlang](#), [load balancing](#)

Publications with words: [replication](#), [cluster](#), [non relational database](#), [NoSQL](#), [erlang](#), [load balancing](#)

References

1. Armstrong J. *Programming Erlang: Software for a Concurrent World*. Pragmatic Bookshelf, 2007. 536 p.
2. Dean J., Ghemawat S. Map reduce: A flexible data processing tool. *Communications of the ACM*, 2010, vol. 53, no. 1, pp. 72–77. DOI: 10.1145/1629175.1629198.
3. Gilbert S., Lynch N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 2002, vol. 33, no. 2, pp. 51-59. DOI: 10.1145/564585.564601.
4. Gittleman A. Multicore computing using Erlang. *Proc. 2009 Int. Conf. on Frontiers in Education: Computer Science & Computer Engineering (FECS-2009)*, July 13-16, 2009, Las Vegas, NE, CSREA Press, 2009, pp. 261-266.
5. Anderson J.Ch., Lehnardt J., Slater N. *CouchDB: The Definitive Guide. Time to relax*. O'Reilly Media, 2009. 272 p.
6. Leavitt N. Will NoSQL Databases Live Up to Their Promise? *Computer*, 2010, vol. 43, no. 2, pp. 12–14. DOI: 10.1109/MC.2010.58.
7. Okasaki Ch. *Purely functional data structures*. Cambridge, CUP Publ., 1999. 232 p.