

2.3.1.

А.Ю. Унгер

МИРЭА – Российский технологический университет,
институт информационных технологий,
кафедра вычислительной техники,
Москва, unger@mirea.ru

ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ НА БАЗЕ ФОРМАЛЬНЫХ ГРАММАТИК

В работе предлагается новый подход к проектированию распределенных информационных систем в облачном кластере. За основу взята традиционная архитектура клиент-сервер. Предложена стратегия разделения полномочий между клиентом и сервером, при котором логика работы системы реализуется на стороне клиента, а сторона сервера обеспечивает целостность хранения данных и контролируемый доступ к ним. Предложены два подхода к обеспечению защите данных: на уровне отдельных сущностей и на уровне транзакций. Показано, как реализовать программный интерфейс доступа к данным на уровне транзакций.

Ключевые слова: *формальная грамматика, реляционная база данных, дескриптор транзакции*.

Введение. Архитектура клиент-сервер продолжает играть ведущую роль в построении масштабируемых информационных систем. Серверная часть приложения располагается на некоторой виртуальной или физической машине. Совокупность таких машин объединяется в информационных кластер, который является ядром всей системы. Ключевые характеристики кластера, такие как масштабируемость и отказоустойчивость, позволяют обсуживать непрерывный поток клиентских запросов.

Тенденцией последних лет является миграция информационных систем в облако целиком или частично [1]. Таким образом, практически все задачи системы решаются на стороне сервера. Проектирование системы подобным образом позволяет добиться целостности и надежности хранения данных, устойчивости к программным и аппаратным сбоям оборудования и равномерности распределения нагрузки. От клиента требуется лишь отправить запрос на сервер и отобразить результат выполнения данного запроса.

В настоящей статье предлагается новый подход к проектированию информационных систем, который унифицирует некоторые аспекты архитектуры клиент-сервер.

Основная концепция. Современные возможности аппаратной и программной виртуализации позволяет рассматривать облако, как вычислительную среду с некоторым количеством процессорных ядер, некоторым объемом оперативной памяти и некоторой емкостью для постоянного хранения данных. Одной из задач программной инженерии в аспекте проектирования информационной системы в облаке является минимизация вычислительных ресурсов для обслуживания заданного количества клиентов.

Рассмотрим основные задачи, возникающие в практически любой информационной системе. Три основные компонента, составляющие ядро информационной системы, включают: 1) хранилище данных; 2) контроль доступа к данным; 3) кэширование данных.

Хранилище данных может быть реализовано, как реляционная или нереляционная база данных. Реляционные базы данных обеспечивают большие возможности манипулирования данными и в целом показывают лучшие характеристики по производительности по сравнению с нереляционными [2].

Введем в рассмотрение единицу хранения данных. Самый популярный объектно-ориентированный подход [3] к разработке приложений, предполагает модель информационной системы в виде совокупности объектов, которые связаны между собой некоторым отношениями. В мире реляционных баз данных естественным аналогом класса является таблица, строки которой представляют собой отдельные экземпляры данного класса.

Для того чтобы связать два введенных нами понятия – таблица и класс – введем в качестве единицы хранения данных *сущность*. Это понятие встречается практически во всех ORM – объектно-реляционных преобразователях [4]. В терминах ORM сущность описывается некоторым классом, а экземпляр данной сущности сохраняется и извлекается из реляционной базы данных посредством некоторого адаптера (*mapper*).

Наша задача – минимизировать ресурсы системы, затрачиваемые на обслуживание одного запроса. Для этого откажемся от сложных (JOIN) запросов. Очевидно, в этом случае увеличится общее количество запросов к СУБД. Например, вместо запроса вида

$$\begin{aligned} & \text{SELECT } a.* , b.* \text{ FROM } <\text{parent_table}> \text{ AS } a \text{ LEFT JOIN } <\text{child_table}> \text{ AS } b \text{ ON } b.parent_id \\ & = a.id \text{ WHERE } <\text{some_condition}> \text{ ORDER BY } a.id; \end{aligned}$$

придется использовать два последовательных запроса вида

- 1) $\text{SELECT } * \text{ FROM } <\text{parent_table}> \text{ WHERE } <\text{some_condition}>;$

- 2) $\text{SELECT } * \text{ FROM } <\text{child_table}> \text{ WHERE } parent_id \text{ IN } (<\text{parent_id_set}>) \text{ [WHERE } <\text{some_condition}>;]$

Можно видеть, что оба варианта выбирают родительскую сущность совместно с массивом дочерних сущностей. Однако первый вариант предполагает, что сервер базы данных явно знает о взаимосвязи сущностей. Эта взаимосвязь важна при описании схемы базы данных, но не при работе с ней. Итак, следующий шаг в оптимизации работы серверной части информационной системы заключается в том, что логику работы с сущностями мы перекладываем на сторону клиента. Сервер баз данных используется, как хранилище сущностей.

Сказанное позволяет нам сформулировать следующее положение относительно выборки объектов из реляционной базы данных: один запрос может вернуть или единичную сущность, или массив сущностей, выбранных по определенному критерию [5]. Необходимо отметить, что здесь мы говорим не о разгрузке системы в целом, а о перераспределении полномочий. Отслеживание сущностей и связей между ними не является задачей сервера баз данных. Эта задача становится задачей клиентской части информационной системы.

Вопросы безопасности. Перекладывание логики работы с объектами информационной системы выдвигает на передний план вопрос безопасного и разграниченного доступа к данным. В настоящей работе предлагается два подхода к решению данного вопроса.

1) Ограничение доступа на уровне сущности. В этом случае доступ к объектам осуществляется по трехуровневой хеш-таблице (ACL), в которой каждая сущность отображается следующим образом:

$$\text{ACL}[EntityName][EntityAction][EntityFingerprint] = GroupList$$

Здесь *EntityName* – имя сущности, *EntityAction* – производимое действие, *EntityFingerprint* – опознавательный признак сущности (уникальная строка, однозначно идентифицирующая принадлежность объекта к заданному подмножеству сущностей), *GroupList* – список групп пользователей, которым разрешен доступ.

EntityAction ограничивается множеством операторов языка манипулирования данными. К базовым операторам относятся: *создать*, *получить*, *обновить*, *удалить*. Этими действиями описывается полный жизненный цикл существования сущности в базе данных.

2) Доступ на уровне транзакции. Если единицей хранения данных является сущность, то единицей обмена информацией между клиентом и сервером естественной выбрать *транзакцию*. Транзакция представляет собой последовательность обращений к базе данных, гарантирующих целостность и непротиворечивость данных. В таком определении, транзакция практически полностью проецируется на стандартное определение транзакции в

терминах СУБД. Поэтому естественно взять за основу нашей транзакции транзакцию на уровне базы данных.

Ограничение доступа на уровне транзакции предполагает, что по отношению к клиенту, система определяет некоторый программный интерфейс (API). Если описать последовательность обращений, составляющих тело транзакции, и присвоить этому описанию уникальный идентификатор, то можно в запросе к серверу ссылаться на данный идентификатор и передавать необходимые для выполнения запроса данные (контекст). Совокупность этих идентифицированных описаний (дескрипторов) и составляют программный интерфейс доступа к информационной системе. Дескриптор может быть описан на любом языке программирования, поддерживающим базовые инструкции контроля управления (условия, циклы и т.д.). Таким образом, дескриптор можно описать с помощью контекстно-свободной формальной грамматики [6].

Концептуально два предложенных подхода различаются тем, что в первом случае мы передаем на сервер всю схему транзакции, т.е. данные и действия над ними, а во втором случае – только сами данные. Схема транзакции хранится на стороне сервера.

Заключение. Предложенный в статье подход к проектированию информационной системы, состоящий в разделении полномочий между клиентом и сервером, был реализован в облачном кластере, состоящем из четырех принципиальных узлов: узла балансировки нагрузки, узла хранения данных, сервера приложений и сервера кеширования [7]. В качестве сервера кеширования выбрана резидентная система управления базами данных *Redis*. В качестве балансировщика использовался веб-сервер *Nginx*. В качестве основного хранилища данных выбрана СУБД *PostgreSQL*.

Список литературы

1. Hatton L. Empirical test observations in client-server systems // Computer. 2007. V. 40, no 5. pp. 24-29.
2. Lenhardt J., Chen K., Schiffmann W. Energy-efficient web server load balancing // IEEE Systems Journal. 2015. V. 11, no 2. pp. 878-888.
3. Linthicum D.S. A guide to cloud-enabling your software // IEEE Cloud Computing. 2016. V. 3, no 2. pp 20-23.
4. Bolognesi T. Toward constraint-object-oriented development // IEEE Transactions on Software Engineering. 2000. V. 26, no 7. pp. 594-616.
5. Ermakov N.V., Molodyakov S.A. A caching model for a quick file access system // Journal of Physics: Conference Series. 2020. V. 1864. 012095.
6. Лесько С.А. Модели и сценарии реализации угроз для интернет-ресурсов // Russian Technological Journal. 2020. Т. 8, № 6. с. 9-33.
7. Unger A.Y. A formal pattern of information system design // Journal of Physics: Conference Series. 2021. V. 2094. 032045.