Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

Кафедра математической
кибернетики и компьютерных наук

РАЗРАБОТКА МЕТОДА ОПТИМИЗАЦИИ СИСТЕМНОЙ АРХИТЕКТУРЫ ОБЛАЧНОЙ ВЫЧИСЛИТЕЛЬНОЙ СРЕДЫ ПРИ ВЗАИМОДЕЙСТВИИ С БАЗАМИ ДАННЫХ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

Студентки 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Шахрай Дарьи Александровны

Научный руководитель
доцент, к. ф.-м. н. _____ И. А. Батраева

Заведующий кафедрой
к. ф.-м. н. _____ С. В. Миронов

Саратов 2016

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3
1 Основная часть 4
1.1 Доменная можель проекта Glance 4
1.2 Обновление метаданных образа виртуальной машины 5
1.3 Альтернативые решения возникновения «ситуации гонки» 8
2 Решение возикающих проблем с помощью транзакционного слоя 10
ЗАКЛЮЧЕНИЕ 11
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 12

ВВЕДЕНИЕ

OpenStack — это платформа с открытым исходным кодом для создания частных и публичных инфраструктурных облачных сервисов и хранилищ. OpenStack состоит из 18 компонентов. Каждый компонент — это отдельный проект, который отвечает за выполнения определенных задач.

Glance является одним из основных компонентов плаформы является OpenStack. Он предоставляет сервис для работы с образами виртуальных машин и их метаданными.

Самой частой операцией с метаданными является обновление. При обновлении метаданных возникает две проблемы:

1. возникновение «ситуации гонки»;
2. нерациональная нагрузка на базу данных.

Целью данной бакалаврсой работы является оптимизация системной архитектуры компонента Glance для устранения вышеописанных ошибок.

1 Основная часть

Облачные вычисления — это информационно-технологическая концепция, подразумевающая обеспечение повсеместного и удобного сетевого доступа по требованию к общему пулу конфигурируемых вычислительных ресурсов, таких как сети передачи данных, сервера, устройства хранения данных, приложениям и сервисам, которые могут быть оперативно предоставлены и освобождены с минимальными эксплуатационными затратами или обращениями к провайдеру [1].

OpenStack — это платформа с открытым исходным кодом для создания частных и публичных инфраструктурных облачных сервисов и хранилищ [2].

OpenStack состоит из 18 компонентов. Каждый компонент — это отдельный проект, который отвечает за выполнения определенных задач. Компоненты делятся на основные, т. е. ядро платформы, и дополнительные.

Glance является одним из основных проектов OpenStack. Glance был разработан и представлен 2 марта 2011 года на презентации релиза Bexar. Glance разработан на языке Python версии 2.7.

Проект Glance предоставляет сервис, который позволяет загружать и скачивать данные, предназначенные для использования другими проектами платформы OpenStack. В настоящее время Glance позволяет работать с данными, которые являются образами виртуальных машин и метаданными образов.

1.1 Доменная можель проекта Glance

Архитектура проекта Glance построена на основе доменной модели.

Доменные объекты — это объекты в ООП, выражающие сущности из модели предметной области и реализующие бизнес-логику программы.

Основная цель использования доменной модели — это разделение логики на независимые слои для работы с доменными объектами. Каждый слой доменной модели оборачивает объект в логику слоя, создавая тем самым «лук» структуру [3].

Основой архитектуры являются базовый класс объекта и прокси-классы. Прокси-классы реализуются на каждом слое доменной модели и реализуют саму логику слоя [3].

Доменная модель в проекте Glance состоит из 6 слоев [4].

Каждый слой доменной модели имеет следущие классы:

1. ImageFactoryProxy
2. ImageRepoProxy
3. ImageProxy

Класс ImageFactoryProxy наследуется от одноименного базового класса и отвечает за создание нового образа. Данный класс накаждом слое содержит метод инициализации и метод `new_image`, который создает основу нового образа.

Помимо метода инициализации, в классе могут присутствовать методы, реализующие логику слоя.

Класс ImageProxy наследуется от базового класса Image отвечает за работу с образом как с объектом доменной модели на каждом слое. Класс может содержать как основной метод инициализации, так и дополнительные методы, реализующие логику слоя. При инициализации экземпляра класса, к объекту можно добавить дополнительный методы и параметры, которые будут доступны только при работе на данном слое.

Класс ImageRepoProxy наследуется от базового класса Repo и отвечает за работу с данными, за их изменение и сохранение. Класс содержит метод инициализации, а так же может содержать следующие методы:

1. `get` — возвращет метаданные указанного образа;
2. `list` — возвращает список всех образов, доступных пользователю;
3. `add` — добавляет метаданные в БД при создании образа;
4. `save` — сохраняет в БД измененные метаданные образа;
5. `remove` — удаляет метаданные образа из БД.

1.2 Обновление метаданных образа виртуальной машины

Основным объектом с которым работает Glance является образ виртуальной машины — `image`. Образ состоит из метаданных и файла. Файл храниться в хранилище, метаданные — в базе данных.

Метаданные представляются в виде словаря, ключами которого являются названия параметром, а значениями — значения данного параметра.

Каждый образ имеет следующие параметры:

— `image_id` — это уникальный идентификатор образа, задается автоматически, имеет формат UUID, неизменяем, является primory key при работе с базой данных;

— `status` — статус образа, статус может изменяться автоматически и с помощью специальных запросов;
    1. `queued` — «в очереди», устанавливается автоматически при создании образа
    2. `saving` — «сохранение», устанавливается автоматически в момент начала загрузки файла образа.
    3. `active` — «активный», устанавливается автоматически при завершения загрузки образа, также устанавливается с помощью специального запроса.
    4. `deactivated` — «деактивный», устанавливается с помощью запроса, что бы ограничить доступ к образу для всех пользователей, кроме администратора.
    5. `killed` — «убитый», устанавливается при возникновении ошибки чтения файла.
    6. `deleted` — «удаленный», устанавливается автоматически после удаления образа
    7. `pending_deleted` — «помеченный на удаление», устанавливается автоматически в момент начала удаления образа.
— `created_at` — дата и время создания образа, устанавливается автоматически в формате ISO 8601 DateTime;
— `updated_at` — дата и время последнего обновления образа, устанавливается автоматически в формате ISO 8601 DateTime;
— `name` — имя образа;
— `visibility` — тип видимости образа (публичный или приватный);
— `min_disk` — минимальный размер памяти на диске для работы с образом;
— `min_ram` — минимальный размер оперативной памяти для работы с образом;
— `protected` — уровень защиты образа;
— `locations` — список внешних ссылок, откуда может быть скачен образ;
— `checksum` — чексумма по загруженному файлу, устанавливается автоматически;
— `owner` — владелец образа;
— `disk_format` — формат образа;
— `container_format` — формат используемого контейнера;

— `size` — размер загруженного образа, указывается автоматически в байтах;
— `virtual_size` — виртуальный размер образа;
— `extra_properties` — дополнительные свойства;
— `tags` — теги.

Самой частой операцией над образом является обновление метаданных.

Алгоритм обновления метаданных образа следующий:

1. Инициализируем `image_repo` — экземпляр класса ImageProxy.
2. Вызывается метод `get` объекта `image_repo`, получаем объект `image`. Из базы данных считывается соответсвующий образ. Поля объекта `image` заполняются значениями из БД.
3. К объекту `image` применяются изменения. Для принятия изменений вызываются соответствующие методы для каждого изменяемого параметра.
4. Вызывается метод `save` объекта image. Объект image передается на нижний слой доменной модели, преобразуется и записывается в БД.
5. Пользователю возвращается измененный образ.

При обновлении метаданных возникает две проблемы:

1. нерациональная нагрузка на базу данных — при обновлении даже одного параметра метаданных образа происходит перезапись всех полей метаданны;
2. возникновение «ситуации гонки».

Ситуация гонки — это ошибка в многопоточной системе, при которой работа системы зависит от порядка выполнения части кода.

В Glance эта ошибка заключается в потере данных, в случае параллельной обработке нескольких запросах [5] [6].

Рассмотрим подробнее на примере.

Для удобства, временные отрезки, о которых говориться в примере, представлены на рисунке 1.

В базе данных имеется образ с параметрами:
`image_in_db = {name: name1, key1: value1, key2: value2}`.

Glance обрабатывает 2 запроса:

— запрос 1 — задать значение параметра `key1=new_value1`.
— запрос 2 — задать значение параметра `key2=new_value2`.

1. В момент времени 1 сервис делает запрос к базе данных, получает образ, назовем его `image1`.

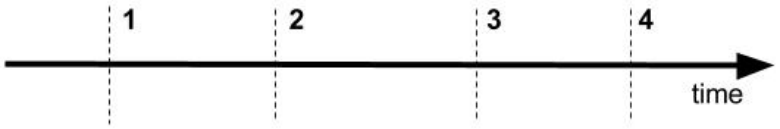

Рисунок 1 – Пример возникновения «ситуации гонки»

`image1 = {name: name1, key1: value1, key2: value2}`

2. В момент времени 2 сервис делает запрос к базе данных, получает образ, назовем его `image2`.

   `image2 = {name: name1, key1: value1, key2: value2}`

3. В отрезок времени 1 — 3 обрабатывается первый запрос, изменяется `image1`.

   `image1 = {name: name1, key1: new_value1, key2: value2}`

4. В момент времени 3 `image1` записывается в БД. Меняется значение `image_in_db`.

   `image_in_db = {name: name1, key1: new_value1, key2: value2}`

5. В отрезок времени 2 — 4 обрабатывается второй запрос, изменяется `image2`.

   `image2 = {name: name1, key1: new_value2, key2: value2}`

6. В момент времени 4 image2 записывается в БД. Меняется значение `image_in_db`.

   `image_in_db = {name: name1, key1: value1, key2: new_value2}`

В итоге, видно, что в подобной ситуации, выполнение запроса 2 полностью затирает изменения, внесенные по запросу 1.

1.3 Альтернативые решения возникновения «ситуации гонки»

Ранее было сделано несколько попыток решения проблемы возникновения «ситуации гонки».

Вариант 1. Каждый образ имеет параметр `updated_at`, который отображает время последнего изменения метаданных. Изменение данного параметра происходит в момент записи метаданных в БД. Было предложено перед записью метеденных в БД сравнить значения данного параметра измененного

образа и оригинального образа, который находится в БД. Если значения параемтра `updated_at` различны, то это означает, что во время данной транзакции, данные в БД были изменены. В таком случае запись изменений отменяется и пользователю возвращается ошибка [7].

Предложенный вариант решения проблемы не подходит по нескольким причинам. Предложенное решение вызвает неудобства для пользователей при загрузке образов в хранилище. Во время загрузки образа другой пользователь может изменить метаданные образа, например, его имя. В таком случае, изменится значение параметра `updated_at` и необходимое изменение метаданных после загрузки образа будет невозможно. Данные из хранилища будут удалены. А так как загрузка образа модет длится от нескольких часов до нескольких секунд, то возникновение подобной ситуации принесет пользователям большие неудобства.

Вариант 2. Было предложено создать дополнительные параметры у образа, в которые будут записываться сделанные изменения в момент их применения. Это вызывает несколько проблем. Во-первых, это очень сложный, громоздкий и разрозненный код, так как для каждого параметра изменения применяются на разных слоях и с различной логикой. Во-вторых, созданные дополнительные параметры будут передаваться с помощью наследования. А это противоречит основной концепции применяемого шаблона проектирования [8].

2 Решение возикающих проблем с помощью транзакционного слоя

Для решения проблемы быль новый слой доменной модели, который отслеживает изменения и передает в базу данных только те параметры, которые были изменены.

Слой получил название «транзакционный». В доменной модели он находится между слоями «Location» и «DB».

Транзакционный слой реализует следующую логику: при считывании образа из базы данных, на транзакционном слое сохраняются значения параметров образа. После применений изменений к образу, на транзакционном слое вызывается метод, который находит разницу между оригинальным образом и измененным. Найденная разница передается на следующий слой.

Для проверки работы кода было проведено следующее тестирование:

1. модульное тестирование;
2. тестирование производительности;
3. ручное тестирование.

В процессе проведения модульного тестирования было разработано 18 тестов, которые проверяют работу отдельных частей кода.

В ходе тестирования производительности было показано, что тримение транзакционного слоя ускоряет время обновления метаданных образа на 34.35%.

В ходе проведенния ручного тестирования было показано, что применение транзакционного слоя помогает избежать возникновения «ситуации гонки».

ЗАКЛЮЧЕНИЕ

В ходе бакалаврской работы была оптимизирована системная архитектура компонента Glance проекта OpenStack путем создания транзакционного слоя в доменной модели компонента. Разработанная оптимизация позволила уменьшить нагрузку на базу данных при обновлении метаданных образа, тем самым ускоряя обновление на 34.35%, так же логика разработанного слоя позволяет избежать возникновения «ситуации гонки».

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Kavis, M. J.* Architecting the Cloud: Design Decisions for Cloud Computing Service Models (SaaS, PaaS, and IaaS) / M. J. Kavis. — first edition. — Willey, 2014.

2 *Jackson, K.* OpenStack Cloud Computing Cookbook / K. Jackson; edited byC. Bunch. — second edition. — Packt Publishing, 2013.

3 *Wilder, B.* Cloud Architecture Patterns: Using Microsoft Azure / B. Wilder. — O'Reilly Media, 2015.

4 Glance domain model implementation. — URL: `http://docs.openstack.org/developer/glance/domain_implementation.html` (Дата обращения 10.04.2016). Загл. с экр. Яз. англ.

5 OpenStack Glance: Concurrency Update issue in v2. — URL: `https://bugs.launchpad.net/glance/+bug/1371728` (Дата обращения 20.05.2016). Загл. с экр. Яз. англ.

6 OpenStack Glance: Incorrect status change after image uploading in v2. — URL: `https://bugs.launchpad.net/glance/+bug/1372564` (Дата обращения 20.05.2016). Загл. с экр. Яз. англ.

7 Fix incorrect status update during upload in v2. — URL: `https://review.openstack.org/#/c/123799/` (Дата обращения 22.05.2016). Загл. с экр. Яз. англ.

8 v2 update image persists only modified attributes. — URL: `https://review.openstack.org/#/c/123722/` (Дата обращения 22.05.2016). Загл. с экр. Яз. англ.