Список использованных источников:

1. Manning D., Raghavan P., Schutze H. Introduction to Information Retrieval. Cambridge University Press. 2008.
2. Lan M., Tan C.L., Su J., Lu Y. Supervised and Traditional Term Weighting Methods for Automatic Text Categorization. IEEE Transactions on Pattern Analysis and Machine Intelligence. vol. 31. pp. 721–735. 2009.
3. Wang D., Zhang H. Inverse-category-frequency based supervised term weighting scheme for text categorization. 2010.
4. Mori, T. Information gain ratio as term weight: the case of summarization of ir results. In Proceedings of the 19th International Conference on Computational Linguistics, 688–694. Association for Computational Linguistics Publisher. 2002.
5. Lan, M.; Sung, S.-Y.; Low, H.-B.; and Tan, C.-L. A comparative study on term weighting schemes for text categorization. In Proceedings of the International Joint Conference on Neural Networks 2005. 2005
6. Рубцова Ю. В. Разработка и исследование предметно независимого классификатора текстов по тональности. Труды СПИИРАН. – 2014. – Т. 5. – №. 36. – С. 59-77.
7. Воронцов К. В. Лекции по методу опорных векторов. 2007.
8. Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A Practical Guide to Support Vector Classification. 2016.
9. Spark MLlib linear methods. https://spark.apache.org/docs/latest/mllib-linear-methods.html

КЕШИРОВАНИЕ КАК СПОСОБ ОПТИМИЗАЦИИ ROR ПРИЛОЖЕНИЙ

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Свито А.И., Пырлог Е.С., Дорошкевич П.Е.

Стержанов М.В. – канд. техн. наук, доцент

Ruby On Rails (RoR) представляет собой фреймворк для быстрой разработки динамических Web-приложений, написанный на языке Ruby. Платформа обладает огромным списком достоинств, таких как обширное комьюнити, самая «чистая» реализация паттерна MVC и огромное количество готовых инструментов для тестирования, локализации и обеспечения безопасности приложения. Благодаря чему при написании на RoR достигается высокая скорость разработки поддерживаемых и масштабируемых приложений. Однако одним из недостатков данной платформы является её низкая производительность. Нами рассматривается кэширование как один из способов оптимизации производительности RoR приложений.

Наиболее часто используемым способом оптимизации приложения является кэширование, суть которого заключается в следующем:

Зачастую приложению приходится решать по запросу пользователя одни и те же задачи. Если для решения этой задачи требуется большое количество времени, то многократное её выполнение может значительно снизить производительность приложения. Чтобы избежать решения одной и той же задачи много раз используют кэширование. Вместо того чтобы многократно выполнить процедуру, она выполняет всего один раз, но после сохраняет результат своего выполнения. Таким образом, при необходимости повторного решения одной и тоже задачи нам достаточно просто передать тот результат, который был сохранен при первом запросе решения. В Ruby on Rails наиболее активно используются четыре вида кэширования: кэширование объектов, страниц, действий и фрагментов. Рассмотрим подробнее каждый из них:

1. *Кэширование объектов*. Самым простым в понимании является кэширование объектов, которое заключается в выделении в приложении наиболее часто используемых объектов и помещении их во внешнее хранилище, доступ к которому осуществляется быстрее чем получение и создание объекта внутри приложения. Недостаток данного подхода в том, что не всегда удаётся выделить часто используемые сущности в приложении. RubyOnRails является объектно-ориентированным языком и зачастую вся работа приложения заключается в извлечении из базы данных коллекции объектов, к котором пользователь имеет неограниченный доступ. Ввиду этого сложно предугадать объекты, с которыми приложению придётся оперировать чаще всего.

2. *Кэширование страниц*. Кэширование страниц – наиболее эффективная форма кэширования данных, имеющаяся в Rails. Когда пользователь посылает некоторый запрос по конкретному URL, приложение после получения данных и выполнения необходимых действий создает HTML-страницу. Код разметки этой страницы сохраняется в кэше. При необходимости повторной обработки запроса по тому же URL, страница не создаётся заново, а просто извлекается прямо из хэша. При этом фактически приложение не вступает в работу, всю работу в последнем случае выполняет веб-сервер. Таким образом, обработка подобных запросов занимает минимально возможное время, которое зависит исключительно от скорости работы веб-сервера. Однако такой подход, несмотря на простоту, также не лишён недостатков. Поскольку обработка запросов в данном случае происходит без участия приложения, мы не можем обработать один и тот же запрос по-разному, в зависимости от, например, роли пользователя, производящего запрос. Также, из-за того что приложение не включено в обработку запросов, фактически, доступ к содержимому страницы ничем не ограничен.

3. *Кэширование действий*. Для того, чтобы иметь возможность кэширования страниц, доступ к которым имеют лишь ограниченные категории привилегированных пользователей, используется кэширование действий. Как и в предыдущем случае, его целью является кэширование содержимого всей страницы целиком, однако данный подход дает возможность фильтрации пользователей, которым доступен просмотр страницы.

Это позволяет перед передачей содержимого принять решение о том, разрешен ли данному пользователю просмотр содержимого запрашиваемой страницы. Такой способ позволяет избежать проблемы неограниченного доступа к содержимому, однако не позволит нам возвращать разные HTML-страницы для разных ролей пользователя при одном и том же URL запросе.

4. *Кэширование фрагментов*. Как было сказано нами ранее, при создании динамических веб-приложений кэширование страницы целиком оказывается возможным только в очень редких случаях, поскольку в современных приложениях каждая HTML-страница представляет собой не единый блок данных, а набор различных компонент, выполняющих свои роли. Кэширование отдельных фрагментов страницы способно разрешить эту проблему. Очевидно, что если наша страница представляет из себя набор компонент, мы можем кэшировать не саму страницу, а её составные части. При этом отсутствует ограничение на количество кэшируемых фрагментов на странице. Поскольку каждая страница может состоять из почти неограниченного числа компонент (как в случае бесконечно прокручиваемых страниц) при использовании этого метода резко увеличивается объем хранимой информации для каждой страницы, перед нами встает вопрос об эффективном средстве для хранения кэша. Ruby on Rails предоставляет несколько стратегий хранения: в памяти, в отдельном каталоге, на отдельном сервере и др. Наиболее рациональным средством для кэширование фрагментов на сегодняшний день является memcached. Memcached – высокопроизводительная распределенная система кэширования. По сути, данная технология представляет из себя структуру данных на подобии ассоциативного массива, где каждом ключу (идентификатору фрагмента) ставится в соответствие готовый код компоненты HTML-страницы.

Инструмент кэширования фрагментов достаточно гибок и лёгок в использовании, что позволяет значительно ускорять работу приложения, сохраняя часто отображаемые фрагменты веб-страниц. При этом область применения кэширования фрагментов практически не ограничена.

Список использованных источников:

1. Edd Dumbill Learning Rails Rails from the Outside In // ch.18 November 2008 O'Reilly Media
2. Jose Valim Crafting Rails 4 Applications // p.137-161 November 24, 2013 Pragmatic Bookshelf
3. Brad Ediger Advanced Rails // ch.6 December 2007 O'Reilly Media

РЕАЛИЗАЦИЯ ВИРТУАЛЬНОЙ МАШИНЫ

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Сернацкий В. И.

Жвакина А. В. – канд. техн. наук, доцент

Со стремительным развитием информационных технологий мощности компьютеров достигли такого уровня, когда одна физическая машина может поддерживать несколько изолированных сред в виртуальных машинах. На данный момент разрабатывается и существует множество вариантов их применения. Виртуальная машина по сравнению с физической обладает большей гибкостью в контексте переносимости на другие платформы. В работе рассматривается спецификация мнимой архитектуры набора команд и реализация виртуальной машины, исполняющей код этой архитектуры.

Виртуальная машина – программная или аппаратная система, эмулирующая некоторую платформу и исполняющая код для этой платформы в изолированной среде на другой или такой же платформе.

Виртуальные машины применяются для решения различных задач: эмуляция различных архитектур, моделирование информационных компьютерных систем на одной машине, исследование поведения и производительности ПО, оптимизация использования виртуальными средами физических ресурсов, защита от вредоносного кода, требующего доступа к физическим ресурсам, спецификация некоторой вычислительной среды.

Архитектура набора команд – часть архитектуры компьютера, определяющая программируемую часть ядра микропроцессора: модели памяти, взаимодействие с внешними устройствами ввода-вывода, режимы адресации, машинные инструкции, типы внутренних данных, обработчики прерываний и исключений.

Разработка велась на языке программирования C++ с соблюдением современных общепринятых стандартов, таких как ISO/IEC 14882:2014, и без внедрения платформозависимого кода. Это гарантирует способность разработанной виртуальной машины работать на более чем одной аппаратной платформе – кроссплатформенность. Получившаяся архитектура достаточно компактная, чтобы уместиться и работать на микроконтроллерах и однокристальных системах, таких, как Arduino и RaspberryPi. Это распространяет спектр применения на сферу интернета вещей.

От таких известных аналогов, как OracleVMVirtualBox, MicrosoftHyper-V, VMwareWorkstation, созданная виртуальная машина отличается тем, что её архитектура набора команд разработана специально для данной виртуальной машины и не имеет реализации в существующих аппаратных платформах. В этом данный программный продукт схож с виртуальными машинами языков программирования: Java Virtual Machine, Common Language Runtime, Forth.

Для упрощения рассматриваемая архитектура оперирует единственной моделью памяти –