

Санкт-Петербургский государственный университет

Кафедра Системного Программирования

Группа 21.Б10-мм

Координация клиентских библиотек Java AWS Lambda с OpenJDK Project CRaC

Ксенчик Никита Владимирович

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
старший преподаватель кафедры Системного программирования, А. П. Козлов

Санкт-Петербург
2022

Оглавление

Введение	3
1. Постановка задачи	5
2. Обзор	6
2.1. Project CRaC	6
2.2. CRaC API	7
2.3. Каноничный пример использования CRaC API	7
2.4. AWS lambda	8
2.5. AWS lambda + CRaC	8
3. Ход работы	10
3.1. Изучение CRaC	10
3.2. Изучение AWS lambda	10
3.3. Подготовка окружения	10
3.4. Изучение текущей реализации поддержки CRaC в AWS lambda	10
3.5. Выявление текущих проблем и/или возможностей для улуч- шения координации проекта CRaC и библиотеки Java AWS Lambda	11
3.6. Подготовка к внесению изменений в текущую поддержку CRaC'a Java AWS lambda	11
3.7. Текущий прогресс	12
4. Эксперимент	13
Заключение	14
Список литературы	15

Введение

С момента релиза и по сегодняшний день Java является одним из самых популярных языков программирования. [7] Java используется во многих отраслях: веб-разработка, мобильная разработка, финансы, настольные приложения, микросервисы... Одним из ключевых требований предъявляемых к языку программирования является его производительность. В этом у Java программ есть несколько проблем: долгий старт и необходимость в "разогреве" приложения.

Java программы состоят из множества классов, и перед началом работы Java приложения происходит загрузка классов, как написанных в процессе создания приложения, так и нужных для работы виртуальной машиной Java (JVM). Перед использованием класса он должен быть проинициализирован. Для ускорения этих процессов Java использует Just-In-Time компиляцию, которая даёт прирост производительности, но и требует использования дополнительных ресурсов системы. [6] Также существует Application Class Data Sharing - технология, которая помогает ускорять запуск и экономить память за счет оптимизации загрузки классов. Однако эта технология даёт лишь небольшой прирост производительности.

Вторая проблема связана с используемой в Java отложенной загрузкой классов. Загрузка классов ссылочных полей загружаемого класса не будет выполняться до тех пор, пока в приложении не встретится явное к ним обращение. [8] Этот процесс может быть оптимизирован с помощью Ahead-of-Time компиляции (то есть компиляции перед исполнением), которая позволяет предгенерировать нативный код, который можно использовать вместо траты времени на JIT-компиляцию. Однако этот процесс тоже требует ресурсов памяти и времени. Другое решение — технология GraalVM Native Image, позволяющая получить нативный образ программы при помощи Ahead-of-Time компиляции из Java в машинный код. Но Graal имеет ряд свой ряд ограничений. [5]

В качестве альтернативного решения данных проблем был разработан проект CRaC - Coordinated Restore at Checkpoint. [10] [11]

CRaC - технология, которая позволяет создать образ запущенного Java приложения и после развернуть оригинальное приложение в том состоянии, в котором оно было при создании образа, пропуская стадии загрузки классов и разогрева приложения.

Для того чтобы библиотека поддерживала CRaC необходимо сделать небольшие изменения в этой библиотеке.

На данный момент в библиотеке AWS lambda существует начальная поддержка проекта CRaC, которая нуждается в улучшении.

1. Постановка задачи

Целью работы является улучшение координации клиентских библиотек Java AWS Lambda с OpenJDK Project CRaC. Для её выполнения были поставлены следующие задачи:

1. сделать обзор взаимодействия библиотеки Java AWS Lambda и проекта CRaC;
2. обнаружить возможности для исправления текущих проблем и/или улучшения координации проекта CRaC и библиотеки Java AWS Lambda;
3. произвести обнаруженные исправления и/или улучшения и оценить их эффект;

2. Обзор

2.1. Project CRaC

Coordinated Restore at Checkpoint - проект в OpenJDK исследующий возможности ускорения старта Java программ. CRaC позволяет создавать образ JVM:

- куча
- треды и их состояния
- загруженные классы
- JIT компиляции

для дальнейшей возможности быстрого восстановления уже ”разогретого” Java приложения и работы с ним.

CRaC предоставляет две операции:

- Checkpoint - создание образа
- Restore - восстановление образа

В момент Checkpoint’а приложение может иметь внешние зависимости: открытые файлы, сетевые соединения и тп. Для безопасного создания образа CRaC требует, чтобы все такие соединения были закрыты, иначе CRaC отказывается делать Checkpoint.

В момент Restore’а все соединения должны быть восстановлены, для продолжения работы приложение. Если это невозможно, то CRaC запускает пользовательский код ответственный за Restore, собирает все ошибки и позволяет их обработать.

Если Checkpoint был создан, то CRaC обязуется восстановить процесс в первоначальном виде, без соединений с внешним миром, после чего Java код должен восстановить все необходимые соединения и обработать все исключительные ситуации, что говорит о надёжности CRaC’а.

2.2. CRaC API

Управление ресурсами предоставляется посредством CRaC API.

2.3. Каноничный пример использования CRaC API

Листинг 1: CRaC API

```
import jdk.crac.*;

class ResourceManager implements Resource {
    Server server;

    @Override
    public void beforeCheckpoint(Context<? extends Resource>
        context) throws Exception {

    }

    @Override
    public void afterRestore(Context<? extends Resource>
        context) throws Exception {

    }
}

public Manager() {
    ...
    Core.getGlobalContext().register(this);
}
```

Класс `ResourceManager` реализует методы `beforeCheckpoint` и `afterRestore`. Эти методы вызываются в соответствующие им фазы.

Метод `beforeCheckpoint` отвечает за безопасное закрытие всех соединений.

Метод `afterRestore` отвечает за восстановление соединений после восстановления.

Так как в общем случае соединения не могут быть прерваны сразу или иметь нетривиальный процесс закрытия, то закрытие соединений прописывается вручную, CRaC берёт на себя обнаружение этих соединений.

При выбрасывании одним из этих методов исключений, собирает их, записывает в логи и заканчивает работу с ошибкой. [9]

2.4. AWS lambda

AWS Lambda — это событийно-ориентированный сервис бессерверных вычислений, который позволяет выполнять код без выделения и администрирования серверов и дополнять другие сервисы AWS на основе пользовательской логики. Lambda автоматически реагирует на различные события (так называемые триггеры), например на HTTP-запросы через Amazon API Gateway, изменение данных в корзинах Amazon S3 или таблицах Amazon DynamoDB.

Lambda выполняет код на высокодоступной вычислительной инфраструктуре и полностью отвечает за администрирование нижележащей платформы, включая обслуживание серверов и операционной системы, выделение ресурсов, автоматическое масштабирование, мониторинг кода и ведение журналов. То есть достаточно загрузить код и настроить, как и когда он должен выполняться. В свою очередь, сервис позаботится о его запуске и обеспечит высокую доступность вашего приложения. [2] [12]

2.5. AWS lambda + CRaC

Для AWS lambda пользу CRaC'a невозможно переоценить, ведь в самом принципе работы AWS lambda стоит создание новой среды выполнения и запуска "холодной" функции обработчика при каждом обособленном запросе (среда может обрабатывать несколько запросов поочередно или параллельно, но рано или поздно она всё равно будет уни-

чтожена), а плата взимается именно за время работы функции обработчика.

[3]

3. Ход работы

3.1. Изучение CRaC

Для начала работы над координацией CRaC'a и AWS lambda требовалось изучить проект CRaC: ознакомиться с принципами, посмотреть примеры.

3.2. Изучение AWS lambda

Далее следовало изучить сервис AWS lambda:

- узнать идеи стоящие за AWS lambda service;
- изучить структуру AWS lambda проекта;
- посмотреть примеры AWS lambda функций на языке Java;

3.3. Подготовка окружения

Для запуска и дальнейшего изучения следовало проверить наличие нужных инструментов и установить недостающие: maven, docker, openjdk14. Далее следовало установить сам проект и прописать пути до зависимостей (openjdk14).

3.4. Изучение текущей реализации поддержки CRaC в AWS lambda

Следующим этапом следовало пошаговое исполнение сценария работы примера и его изучение.

3.5. Выявление текущих проблем и/или возможностей для улучшения координации проекта CRaC и библиотеки Java AWS Lambda

В процессе изучения работы сценария примера `example-lambda` было обнаружено следующее: при отправке запроса `post checkpoint` должно происходить создание Checkpoint'a, но в реальности JVM 'замораживается', а создание самого Checkpoint'a требует отправки фиктивного, пустого запроса. Данная "особенность" противоречит ожиданиям пользователя, её наличие ничем не оправдано, и вредно для выполнения программы, поэтому её следует исправить.

Также в проекте обнаружены закомментированные куски кода указывающие на то, что в данном проекте планировалась поддержка сервиса `aws-xray`, который в данный момент недоступен.

Далее было решено приступить к исправлению первого бага.

3.6. Подготовка к внесению изменений в текущую поддержку CRaC'a Java AWS lambda

Данный баг располагается в одной из зависимостей подключаемых к проекту-примеру, поэтому следовало научиться вносить в них изменения.

На гитхабе лежат исходники используемых в `example-lambda` библиотек [1]. При сборке проекта `maven` смотрит, есть ли данная библиотека в локальном хранилище, и если её нет загружает её из удалённого репозитория (если такая нашлась там), и помещает в локальный, чтобы избежать скачивания при каждой сборке [4]. Имея исходники можно вносить в них нужные изменения и командой `mvn clean install` устанавливать их в локальный репозиторий, после чего при сборке `example-lambda` будут подключаться уже изменённые зависимости.

3.7. Текущий прогресс

На этом месте семестр подошёл к концу, поэтому работа будет продолжена в следующем.

4. Эксперимент

По завершении исправления проблемы с зависанием создания Checkpoint'a планируется собрать уже имеющийся простой пример проекта Java AWS lambda с поддержкой CRaC [3] с внесёнными исправлениями и проверить следующее:

1. при совершении post checkpoint не происходит зависания программы;
2. Checkpoint создаётся;
3. Restore проходит успешно;

Если вышеизложенные пункты выполнены, тогда исправление зависания создания Checkpoint'a будем считать успешным.

Заключение

В ходе выполнения данной работы были достигнуты следующие результаты:

- был сделан обзор взаимодействия проекта CRaC и библиотеки Java AWS Lambda;
- была обнаружена возможность для улучшения координации проекта CRaC и библиотеки Java AWS Lambda;
- была произведена подготовка к улучшению координации проекта CRaC и библиотеки Java AWS Lambda;

Список литературы

- [1] AWS Lambda Libraries. — URL: <https://github.com/CRaC/aws-lambda-java-libs> (дата обращения: 2022-12-12).
- [2] AWS lambda - developer guide. — URL: <https://docs.aws.amazon.com/lambda/latest/dg/welcome.html> (дата обращения: 2022-12-12).
- [3] Example lambda. — URL: <https://github.com/CRaC/example-lambda> (дата обращения: 2022-12-12).
- [4] Maven — хранилища. — URL: <https://coderlessons.com/tutorials/java-tekhnologii/uchitsia-maven/maven-khranilishcha> (дата обращения: 2022-12-14).
- [5] Native Image Compatibility and Optimization Guide. — URL: <https://docs.oracle.com/en/graalvm/enterprise/20/docs/reference-manual/native-image/Limitations/> (дата обращения: 2022-12-11).
- [6] Sergey Vitali. Инструменты для запуска и разработки Java приложений, компиляция, выполнение на JVM. — URL: <https://habr.com/ru/post/471772/> (дата обращения: 2022-12-10).
- [7] TIOBE Index. — URL: <https://www.tiobe.com/tiobe-index/> (дата обращения: 2022-12-10).
- [8] Zimin Aleksandr. Как происходит загрузка классов в JVM. — URL: <https://javarush.com/groups/posts/646-kak-proiskhodit-zagruzka-klassov-v-jvm> (дата обращения: 2022-12-10).
- [9] Антон Козлов — Coordinated Restore at Checkpoint: Быстрый старт OpenJDK. — URL: https://www.youtube.com/watch?v=RLFQj2mPqUM&t=221s&ab_channel=JPoint%2CJoker%D0%B8JUGru (дата обращения: 2022-12-12).

- [10] Проект CRaC. — URL: <https://openjdk.org/projects/crac/> (дата обращения: 2022-12-10).
- [11] Проект CRaC на github. — URL: <https://github.com/CRaC/docs> (дата обращения: 2022-12-14).
- [12] Строчков Владислав. Детальный разбор AWS Lambda. — URL: <https://habr.com/ru/company/otus/blog/466519/> (дата обращения: 2022-12-12).