Интеграция Hazelcast в сервер федеративных запросов

Выполнил:
Шапошников Алексей, 471 группа
Научный руководитель:
ст. преп. С. Ю. Сартасов
Консультант:
директор департамента
высокотехнологичного производства
ООО “БФТ” Б. В. Щукин

МатМех, СПбГУ
2021

Введение в тему

- Все больше информации и необходимости ее хранить и обрабатывать
- Возрастает актуальность вопроса об улучшении производительности
- Базы данных зачастую являются узким местом системы. Поэтому возникли новые технологии и подходы, в том числе:
  - Кэширование
  - Паттерны доступа к источникам данных
  - IMDG
- Рост числа источников данных привел к появлению Federated Search Engine

Цели и задачи

- Провести обзор технологий IMDG и выбрать подходящий для поставленной задачи data grid
- Внедрить технологию во внутренний продукт компании ORM DataMaps
- Реализовать шаблоны коммуникации с базой данных (read-through, write-through), выгрузку из базы данных
- Реализовать модуль источника данных для распределенных запросов с использованием data grid для ORM
- Протестировать работу data grid resolver в сервере федеративных запросов в автоматизированных тестах и на реальном продукте, а также провести замеры сравнения с запросами к удаленной базе данных

Обзор внутреннего ORM-фреймворка

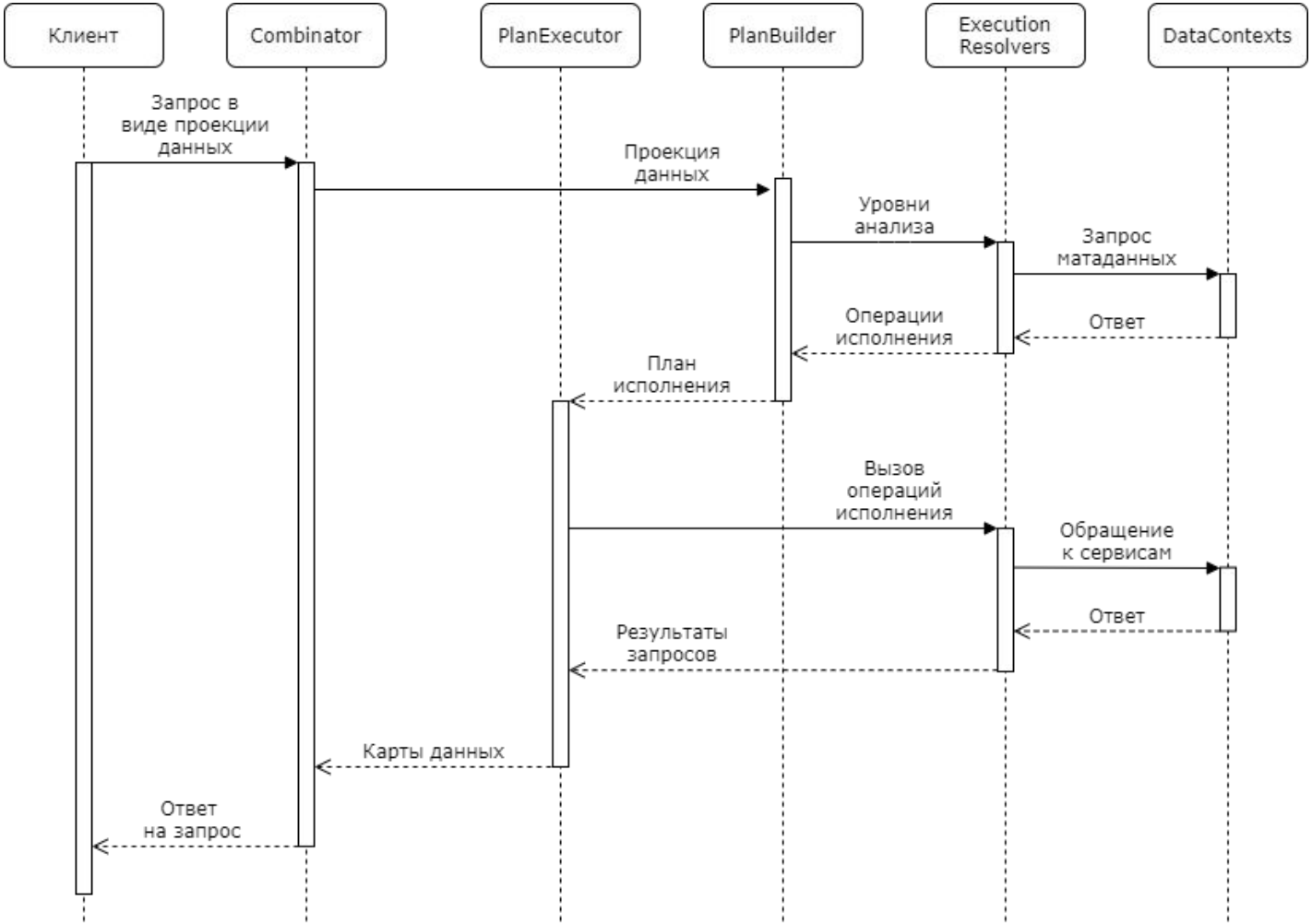
- Фреймворк для **Spring**, надстройка над **Spring JDBC**
- Написан на **Kotlin**, предоставляет динамический **DSL**
- **DataMaps** (карты данных) - представление табличных сущностей
- **FieldSets** (наборы полей) - шаблоны для построения карт данных с нужной структурой полей
- **DataProjections** (проекции) - инкапсулируют запрос к БД
- **DataService** (сервис данных) - основной сервис для взаимодействия с фреймворком
- **Combinator** - реализация Federated Search Engine

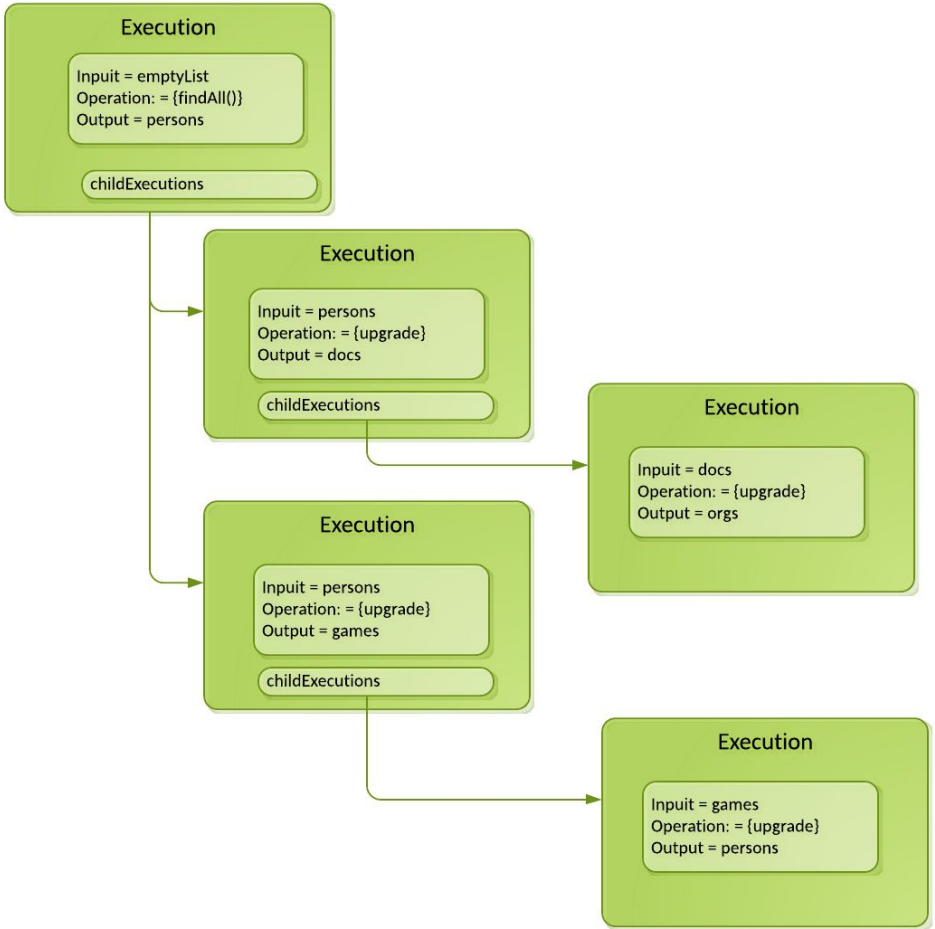
Combinator (FSE)

Многие фреймворки стали поддерживать функциональность федеративных запросов либо выделять в качестве отдельного продукта:

- Apollo Federation
- Amazon Athena Federated Query
- Arenadata Prostore
- Apache Drill
- Presto и др.

У фреймворка DataMaps хорошая совместимость для такой технологии


Клиент
Combinator
PlanExecutor
PlanBuilder
Execution Resolvers
DataContexts
Запрос в виде проекции данных
Проекция данных
Уровни анализа
Запрос матаданных
Ответ
Операции исполнения
План исполнения
Вызов операций исполнения
Обращение к сервисам
Ответ
Результаты запросов
Карты данных
Ответ на запрос


Execution
Inpuit = emptyList
Operation: = {findAll()}
Output = persons
childExecutions
Execution
Inpuit = persons
Operation: = {upgrade}
Output = docs
childExecutions
Execution
Inpuit = docs
Operation: = {upgrade}
Output = orgs
Execution
Inpuit = persons
Operation: = {upgrade}
Output = games
childExecutions
Execution
Inpuit = games
Operation: = {upgrade}
Output = persons

Выбор поставщика IMDG

Таблица 1: Таблица сравнения вендоров

Название	Memcached [22]	etcd [23]	Hazelcast [24]	Ignite [25]
Шаблоны доступа	Cache aside	–	Map Store [26] (read-though, write-through и предзагрузка данных)	Read-through, write-through, write-behind [27]
Schema-free	+	+	+	–
Распределенная обработка данных	–	–	+	+
Интерфейс предикатов	–	–	Predicate API и поддержка SQL-like запросов	Поддержка SQL запросов
Предикаты по неопределенным полям	–	–	Возможность строить запросы по json объектам и коллекциям в них, не описывая заранее структуру	Необходимо заранее прописывать структуру объектов для построения предикатов по ним

Hazelcast

Hazelcast был выбран, потому что позволяет:

- реализовывать паттерны доступа к данным
- работать с распределенными объектами без указания конкретной структуры
- выбрать мгновенную или конечную целостность данных
- строить предикаты и использовать SQL-подобный синтаксис
- оборачивать json-объекты в HazelcastJsonValue и строить запросы с фильтрами по json полям

Реализация

- Универсальная конфигурация
- MapStore<Any, HazelcastJsonValue>
- Интеграция с Combinator
- Восстановление данных из узла Hazelcast
- Поддержка запросов
  - по id и набору ключей
  - с фильтрами
  - с пагинацией
  - с сортировкой

Замеры

Таблица 2: Замеры времени запросов

Справочник	Театры (500 повторов на запрос)			Объекты культурного наследия (100 повторов)		
Способ запроса	Hazelcast: бинарный формат	Hazelcast: объектный формат	База данных	Hazelcast: бинарный формат	Hazelcast: объектный формат	База данных
findAll	20.084	21.976	196.148	132.97	175.25	1112.34
find с лимитом 20	1321.382	1270.958	44.256	912.73	910.96	20.76
find с лимитом 20 и фильтром	4.33	4.958	28.346	167.56	167.21	19.6
find с фильтром на табличное поле	11.654	23.89	116.96	32.82	31.13	81.32
find с фильтром на json-поле	2.616	3.256	28.924	35.85	35.65	105.7

Выводы о производительности

- Улучшение для запросов на все данные и с фильтрами
- Все запросы с пагинацией показали худшее время по сравнению с БД
  - чем больше записей для сортировки, тем хуже время
  - более сильный фильтр улучшает ситуацию
- Бинарный и объектный формат сравнимы по скорости
- Data Grid стоит использовать для запросов, не требующих сортировки или ограничений на количество возвращаемых записей
- Data Grid не убирает полностью необходимость общения с базой данных, стоит рассматривать как дополнение для конкретных задач
- В проекте со справочниками модуль не представил интереса и был отложен для тестирования в других проектах

Заключение

В течение производственной практики выполнены следующие задачи:

- Произведен обзор технологии IMDG и выбрана подходящая под задачу реализация
- Hazelcast подключен в качестве data grid в библиотеку DataMaps; реализованы шаблоны коммуникации write-through, read-through, выгрузка из базы, сериализация данных
- Реализован модуль разрешателя data grid для Combinator; реализована поддержка различных запросов
- Разрешатель внедрен и протестирован на реальном проекте, замерены данные о его производительности по сравнению с реляционной базой данных, сделаны выводы о перспективах использования