Литература

1 Суровцев, И. С. Нейронные сети. Введение в современную информационную технологию: учеб. пособие / И. С. Суровцев, В. И. Клюкин, Р. П. Пивоварова; ред. И. С. Суровцев. – Воронеж : ВГУ, 1994. – 224 с.

2 Осовский, С. Нейронные сети для обработки информации / С. Осовский; пер. с польского И. Д. Рудинского. – Москва : Финансы и статистика, 2002. – 344 с.

3 Дмитриева, М. Самоучитель JavaScript: справочное пособие / М. Дмитриева. – Санкт-Петербург : БХВ-Петербург, 2001. – 512 с.

4 Флэнаган, Д. JavaScript. Подробное руководство: справочное пособие / Д. Флэнаган. – 5-е изд. – Санкт-Петербург : Символ-Плюс, 2008. – 992 с.

5 Нейронная сеть. Wikipedia [Электронный ресурс]. – Режим доступа : https://ru.wikipedia.org/. – Дата доступа : 8.05.2020.

УДК 004.04

А. В. Барановский

РАЗРАБОТКА МЕТАЯЗЫКА ФОРМИРОВАНИЯ ОТЧЕТА НА ОСНОВЕ ДАННЫХ ИЗ ОБЛАЧНЫХ ХРАНИЛИЩ

Описываются особенности разработанной библиотеки метаязыка построения отчетов по данным из облачных хранилищ с использованием языка программирования Scala и на основе ее интеграции с распределенной платформой обработки данных Apache Spark. Рассмотрены ее модули для работы с входными данными и выполнением запросов к базе данных. Проанализированы ее преимущества над существующими решениями. Показано, что универсального решения не существует и библиотека метаязыка не избавлена от определенных минусов.

В облачных хранилищах можно с относительной простотой развернуть вычисления без необходимости тонкой настройки аппаратной части решения. Облачные провайдеры предоставляют уже готовые кластеры, способные запускать различные типы задач, выполняющие трансформации над данными из облачных хранилищ. Одним из самых популярных фреймворков для работы с подобными задачами является Apache Spark, реализующий распределенную обработку неструктурированных и слабоструктурированных данных и входящий в экосистему проектов Hadoop. Spark использует специализированные примитивы для рекуррентной обработки в оперативной памяти, благодаря чему позволяет получать значительный выигрыш в скорости работы. Фреймворк позволяет выполнять SQL-запросы над данными, работать с потоковыми данными, используется для задач машинного обучения, поддерживает распределенные системы хранения данных, такие как HDFS, Amazon S3, ElasticSearch и т. д. Spark написан на Scala – мультипарадигмальном языке программирования, сочетающем возможности функционального и объектно-ориентированного программирования. Во многом благодаря Spark язык Scala получил широкое распространение в последнее десятилетие. Трансформации с использованием Spark являются распространенной практикой. Однако зачастую при переносе приложений на облачную инфраструктуру некоторые трансформации, ранее работавшие с использованием SQL и старых систем, необходимо переписать на Spark людям, не знакомым ни со Scala, ни с Python.

Поэтому актуальной стала работа по проектированию и разработке библиотеки метаязыка формирования отчета на основе данных из облачных хранилищ различного формата. Для написания трансформаций в ней должен использоваться метаязык, который упростит конструкции, уменьшит время, необходимое на изучение синтаксиса относительно Spark-трансформаций, а также позволит сократить время на поиск ошибок, связанных с некорректно описанными трансформациями.

Созданная библиотека метаязыка включает в себя несколько различных модулей. Все они выполняют одну и ту же задачу: принимают какие-то данные и возвращают их в измененном виде. Модули чтения трансформируют данные во внутренний формат, так как изначально чтение данных из файла означает преобразование данных в набор строк, а их нужно будет преобразовать в объекты внутренних классов библиотек, чем и занимаются модули чтения. При этом модуль, отвечающий за преобразование правил трансформации, которые тоже имеют вид набора строк, выполняет их компиляцию с использованием встроенных утилит языка Scala. На выходе получается функция, запуск которой будет означать выполнение трансформаций. Далее главный модуль применит правила трансформации над данными. Так как в трансформациях могут содержаться обращения к БД, то главный модуль опционально способен выполнять данные запросы, используя другой, специально созданный для этого модуль. В связи с тем, что различные БД могут использовать различные языки запросов (SQL, ELK и т. д.), на входе модуль получает не конкретный запрос, а его структуру в некотором внутреннем формате, определенном библиотекой. Затем в зависимости от используемой БД этот внутренний формат будет преобразован в конкретный язык запросов. После того как все трансформации выполнены, результат их работы все еще будет находиться в виде некоторого набора внутренних объектов библиотеки. Чтобы данный набор стал понятен внешней системе, использующей библиотеку, его необходимо преобразовать в некий конкретный формат данных. Этим и занимается последний модуль. При этом формат данных определяется входным форматом, т. е., если входные данные были в формате json, то и результат библиотека будет возвращать в формате json. Модульная структура библиотеки приведена на рисунке 1.

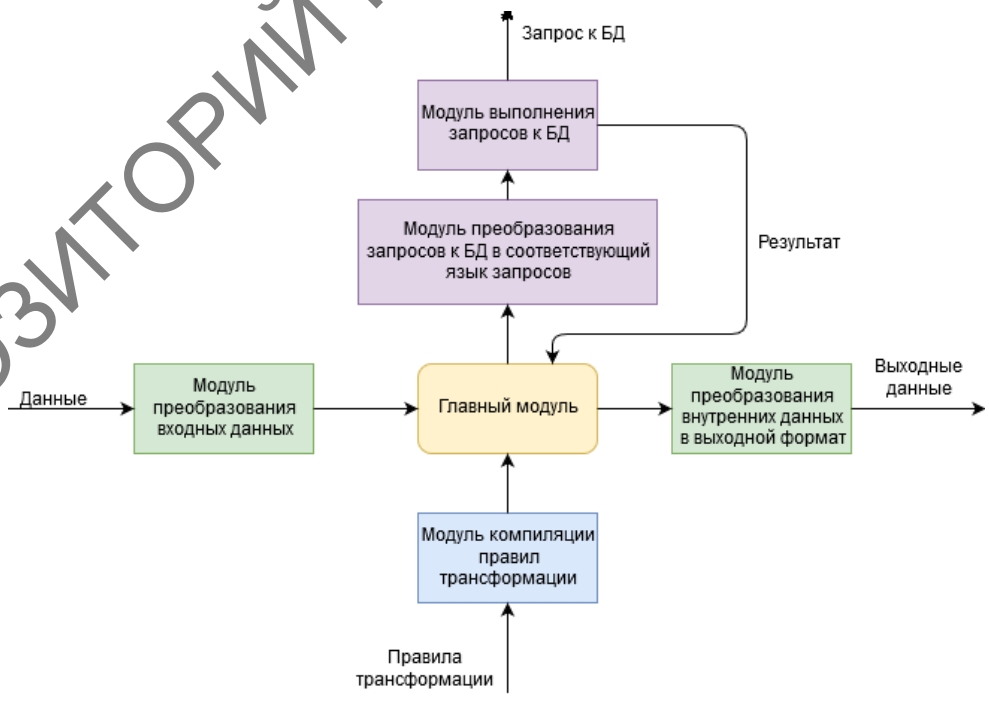

Рисунок 1 – Модульная структура библиотеки метаязыка

При сравнении решения задачи формирования отчета на основе данных из облачных хранилищ различного формата с использованием метаязыка с аналогичным решением на основе обычных Spark-трансформаций (на основе запроса, выполненного на Spark SQL в связке с DataFrames) можно увидеть, что количество вызовов различных методов и функций больше, чем в метаязыке. И комплексность решения с использованием выражения на Spark будет только расти, когда появятся дополнительные условия, например, запросы к разным таблицам в зависимости от входных данных. В случае со Spark придется перемножать все таблицы в независимости от того, используются они или нет, что даст дополнительную нагрузку на систему и сеть.

Также преимуществом метаязыка является то, что он в целом понятен даже тем, кто не привык работать с программным кодом. Люди, знакомые только с SQL, способны в короткие сроки освоить написание новых правил трансформации без необходимости изучения Scala или Spark. «Оболочка» же, необходимая библиотеке метаязыка, может быть написана один раз и затем использоваться для запуска библиотеки метаязыка с различными трансформациями. Либо, если идти дальше, трансформации на метаязыке вполне реально вынести на какой-либо веб-интерфейс, где их можно будет вносить и редактировать, а Spark, в свою очередь, будет их оттуда подгружать и исполнять. Таким образом, людям, работающим с метаязыком, будет предоставлен удобный веб-интерфейс, а не программный. Также эти правила прозрачны и в них легко искать ошибки, в отличие от кода Spark. Зачастую люди, которые отвечают за работу трансформаций, делегируют их написание программистам, знакомым со Spark. В таком случае поиск ошибки также включает себя время, за которое пользователь, заметивший неточность, обратится к программисту. Поиск ошибок в коде, в свою очередь, является достаточно трудоемким занятием и может занять неопределенное количество времени. Были даже выработаны различные стратегии для подхода к решению подобных проблем [1]. Данный же подход предлагает делегировать поиск ошибок пользователям метаязыка и владельцам бизнес-логики приложения. Предметная область и конкретные требования знакомы им куда лучше, чем программистам, задачей которых было лишь «подготовить площадку», поэтому поиск ошибок занимает не так много времени.

Современная тенденция перевода систем обработки данных на облачные платформы, связанная с высоким порогом вхождения пользователей в новые системы, и изучение языка работы с данными, привела к необходимости данной разработки. В ней предоставляется альтернативное решение для работы с данными из облачных хранилищ с целью упрощения изучения и поиска ошибок в наборах трансформаций данных. Практическая значимость работы состоит в возможности использования нового решения для написания трансформаций при обработке данных из облачных хранилищ, уменьшения времени на обучение пользователей новому, более удобному в использовании, метаязыку трансформаций данных. Сравнение среднего времени, затраченного на поиск ошибки на основе 10 ситуаций, предоставлено на рисунке 2.

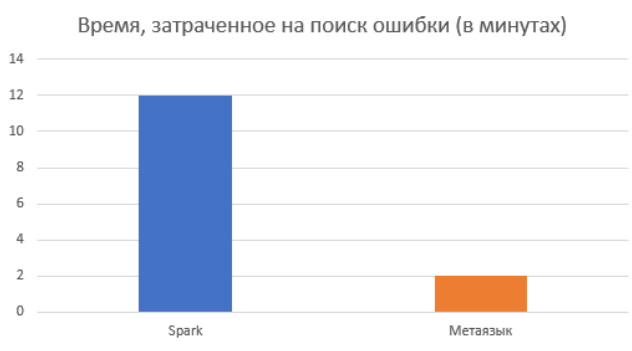

Рисунок 2 – Среднее время, затраченное на поиск ошибки (на основе 10 ситуаций)

С другой стороны, то, что является плюсом, в некоторых ситуациях может являться и минусом. В связи с тем, что библиотека метаязыка применяет трансформации построчно, на большом количестве данных время их выполнения может значительно увеличиться. Пример приведен на рисунке 3.

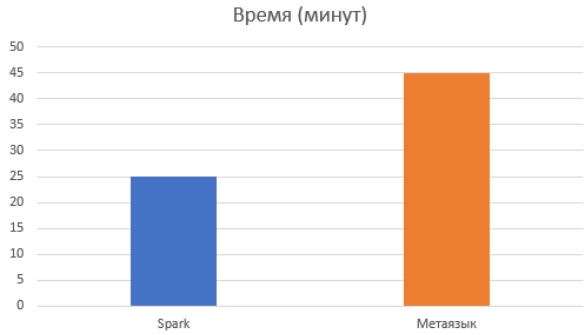

Рисунок 3 – Время, затраченное на применение трансформаций для выборки объёмом в 1000000 строк

Спроектированный метаязык способен решить несколько проблем в ставшем уже классическом подходе Apache Spark, а именно вынести операции над данными «наружу» из кода, что позволяет им быть более прозрачными для тех, кто отвечает за корректность данных трансформаций, а также за анализ их результата. Используя метаязык, можно повысить продуктивность работы за счет уменьшения времени ознакомления с синтаксисом, а также уменьшения времени на поиск ошибок. Однако, как и зачастую, универсального решения не существует, и библиотека метаязыка не избавлена от определенных минусов, но здесь уже все зависит от множества индивидуальных параметров и особенностей конкретных проектов.

Итак, модули разработанного метаязыка связаны друг с другом, формируя каркас библиотеки, определяя ее функционал. Применение разработанного метаязыка увеличивает читаемость запроса, уменьшает время на ознакомление с трансформациями и время на поиск ошибок.

Литература

1 Explicit Programming Strategies [Electronic resource] / T. D. Latoza [et al.]. – 2019. – Mode of access : https://arxiv.org/pdf/1911.00046.pdf. – Date of access : 24.12.2019.

УДК 004.7

В. В. Василевский

ОСОБЕННОСТИ РАЗРАБОТКИ ИГРОВЫХ ПРИЛОЖЕНИЙ НА ПЛАТФОРМЕ ANDROID

В статье рассматриваются особенности разработки игровых приложений на платформе Android, обоснован выбор платформы Android для разработки игровых приложений. Дается описание программных средств разработки приложений, приводится их сравнительный анализ, указываются основные этапы создания Android-приложений. Приводится описание примера игрового приложения «Кто хочет стать миллионером», реализованного на языке Kotlin.