УДК 004.75 + 004.8 + 004.9

Л.Н. Лядова, К.М. Малькова, М.В. Тимофеев

АРХИТЕКТУРА РЕКОМЕНДАТЕЛЬНОЙ СИСТЕМЫ, НАСТРАИВАЕМОЙ НА ПРЕДМЕТНЫЕ ОБЛАСТИ

Аннотация: Описаны требования к универсальным рекомендательным системам, интегрируемым с сервисами сторонних разработчиков. Приведено описание архитектуры системы, настраиваемой на различные предметные области, её компонентов и особенностей их реализации.

Ключевые слова: рекомендательная система, настройка на предметную область, микросервисная архитектура, анализ данных.

Введение

Последнее время становится все более популярным создание систем, которые способны угадывать предпочтения и нужды пользователей и на основе этого предлагать подходящие решения. Такие системы, называемые *рекомендательными*, имеют множество применений. Крупные компании такие, как Amazon, Apple, eBay, Pandora и т.д., используют рекомендательные системы в составе своих сервисов. Что важно, рекомендации формируются для конкретного пользователя, исходя из его личных данных и интересов. Такой подход значительно упрощает пользователю работу с большим объемом данных, помогает сократить время поиска нужного решения, обеспечивая релевантность информации. Однако зачастую рекомендательные системы разрабатываются для решения задач конкретной области. Например, Amazon специализируется на продаваемых им продуктах, Netflix на фильмах и т.д. [1, 2].

Большие компании имеют необходимые средства и квалифицированных специалистов, для того чтобы разработать собственную рекомендательную систему. Компании же, которые только начинают развиваться и имеют необходимость создания сервиса рекомендаций, зачастую не обладают необходимыми ресурсами и временем для этого. Разработка универсальной рекомендательной системы, адаптируемой к различным областям, может решить данную проблему. Можно отметить, что при разработке рекомендательных систем применяются общепринятые походы, основанные на реализации таких методов машинного обучения как коллаборативная фильтрация и фильтрация на основе содержания [3] и пр., что позволяет поставить задачу реализации универсального решения, базирующегося на использовании этих методов. Настройка на конкретные предметные области может быть выполнена на основе моделей, создаваемых с использованием данных, предоставляемых компаниями-клиентами (пользователями).

Сервис рекомендаций, настраиваемый на несколько областей применения, должен иметь в основе не только модель рекомендательной системы, её предметной области и средства сбора данных о пользователях системы, но и включать дополнительные модули ее поддержки, в частности, средства анализа данных. Данные, используемые для выработки рекомендаций, должны соответствовать определенным требованиям: актуальность, правильность, уникальность, полнота, структурированность и т.д. Только при наличии качественных данных возможно принятие решений, наиболее полно отвечающих потребностям пользователей.

Как показал анализ, на данный момент не существует решений, которые позволили бы учесть все требования к системе.

Анализ существующих решений в области рекомендательных систем

В последнее время популярность рекомендательных систем стала очень велика. Это объясняется стремлением компаний упростить выбор пользователя в той или иной ситуации и получить от этого прибыль. В силу большой востребованности систем, которые

автоматически формируют предложения пользователям на основе их интересов, предпринимаются попытки разработки универсальных рекомендательных систем.

На данный момент существуют следующие решения, связанные с созданием рекомендательных систем:

- Во-первых, это сервисы с модулем рекомендаций для определенной предметной области. Такой подход используется, например, в Apple, Netflix, Amazon и других компаниях.
- Во-вторых, это исследования, направленные на создание универсальной рекомендательной системы, обеспечивающей работу с различными предметными областями (Unresyst); различные библиотеки, в которых реализованы методы, которые могут быть использованы для создания собственной рекомендательной системы.
- Третьим решением являются универсальные рекомендательные системы (Apache Prediction IO), которые представляют собой законченный программный продукт.

Для сравнения данных решений были выделены следующие *критерии*:

1) настройка системы на любую предметную область (универсальность);
2) возможность интеграции системы в продукт сторонних разработчиков (использования системы как встраиваемого модуля в собственный сервис);
3) наличие компонент сбора данных для дальнейшего формирования оценок;
4) наличие модуля анализа для обеспечения качества данных и рекомендаций;
5) наличие готового программного продукта на рынке программного обеспечения.

Сравнение существующих решений приведено в табл. 1. Используется следующая шкала для оценивания решений:

0 – решение не выполняет указанной функции;
1 – решение частично выполняет указанную функцию;
2 – решение полностью выполняет указанную функцию.

Таблица 1. Сравнение существующих решений

Критерии	**Рекомендательные системы в составе готовых сервисов**	***Unresyst* (исследовательский прототип)**	**Библиотеки (*Crab, LensKit, RankSys*)**	***Prediction IO***
Универсальность системы	0	2	1	2
Возможность иетеграции	0	0	0	1
Наличие компонент сбора данных	2	0	0	0
Наличие модуля анализа	1	0	0	0
Наличие продукта на рынке	0	0	2	2

Универсальность обеспечивается системой Unresyst, библиотеками (платформами) для разработки рекомендательныхй систем, Prediction IO. Однако Unresyst не является готовым программным продуктом, имеет лишь разработанный универсальный метод формирования рекомендаций без сбора и анализа данных. Библиотеки не являются встраиваемыми модулями, требуется писать код для того, чтобы они могли принимать данные; кроме того, библиотеки не являются масштабируемыми. Сравнение показывает, что Prediction IO более всего удовлетворяет поставленным требованиям: его можно использовать как встраиваемый в системы сторонних разработчиков модуль, однако он не интегрируется полностью с основным сервисом. Кроме того, в нем отсутствуют модули сбора данных и

анализа, которые необходимы при создании рекомендательной системы (готовые сервисы, имеющие модуль рекомендаций, обязательно имеют и эти средства).

Таким образом, выявленные у существующих решений ограничения указывают на *актуальность разработки универсальной рекомендательной системы*, которая способна встраиваться в сервис клиента, имеет компоненты сбора и анализа данных для обеспечения качества используемых данных и результатов.

Общий подход к разработке рекомендательной системы

Общая схема работы с рекомендательной системой в рамках предлагаемого подхода показана на рис. 1. Предлагается интегрировать систему с основным сервисом компании, предлагающей товары или услуги, настроив её на соответствующую область через создание модели предметной области на основе данных, получаемых от основного сервиса в определённом формате.

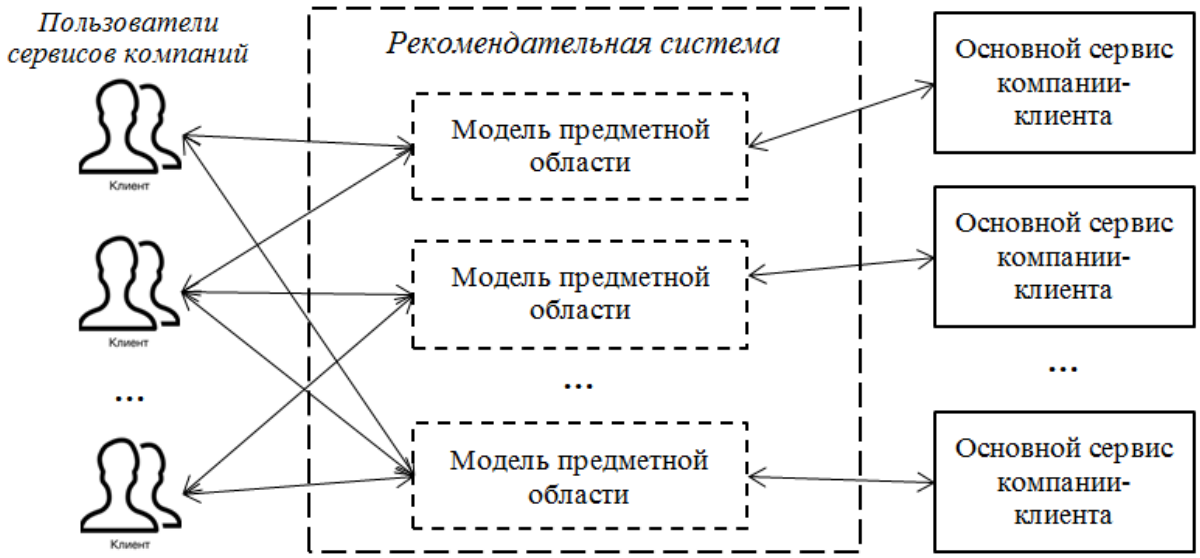

Рис. 1. Схема интеграции рекомендательной системы с основными сервисами компаний-клиентов

Универсальная рекомендательная система включает две подсистемы, взаимодействующие с основным сервисом: собственно рекомендательная система, которая обеспечивает сбор данных и выработку рекомендаций; модуль обработки и интеллектуального анализа данных (рис. 2).

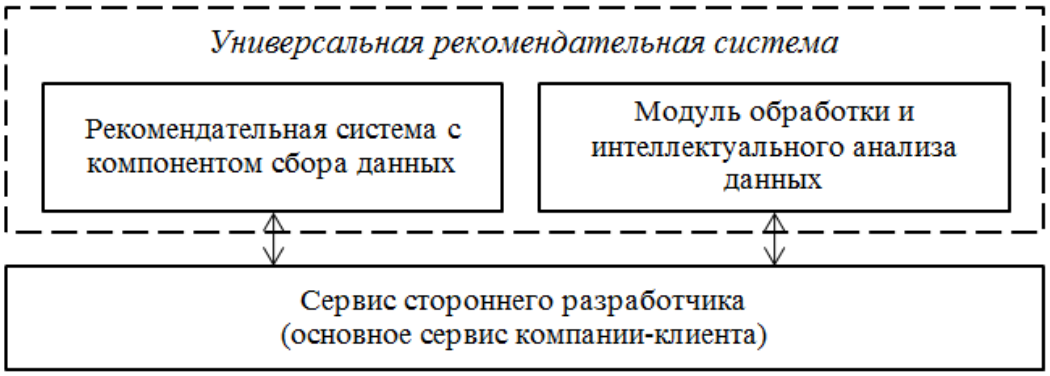

Рис. 2. Подсистемы универсальной рекомендательной системы

Эти две подсистемы могут работать независимо друг от друга. Однако дополнение рекомендательной системы средствами анализа данных существенно повышает качество её работы.

Арххитектура рекомендательной системы

При проектировании архитектуры системы, которая взаимодействует с сервисом и пользователями, был выбран микросервисный подход. *Микросервисный подход* – это особый метод разработки программных систем, который находит широкое применение в последние годы. *Микросервисная архитектура* предполагает разработку программных приложений как набора независимых развертываемых небольших сервисов, каждый из которых запускает уникальный процесс и обеспечивает коммуникацию с помощью четко определенного, легкого механизма. Благодаря гибкости и масштабируемости эти архитектурные решения являются наиболее подходящими, когда нужно включить поддержку для целого ряда платформ и устройств различных типов.

Основные *преимущества* применения микросервисов:

- архитектура предоставляет разработчикам свободу самостоятельно разрабатывать и развертывать службы;
- микросервис может быть разработан небольшой командой;
- код для различных служб может быть написан на разных языках;
- простая интеграция и автоматическое развертывание (с использованием инструментов непрерывной интеграции с открытым исходным кодом, таких как Jenkins, Hudson и т. д.);
- легкость понимания и модификации кода;
- возможность использования разработчиками новейших технологий;
- простота масштабирования и интеграции со сторонними сервисами.

Но этот подход имеет свои *недостатки* – разработчикам приходится иметь дело с увеличением сложности работ при проектировании, реализации и сопровождении распределенной системы, в частности:

- распределённая архитектура привносит дополнительную сложность, поскольку разработчикам приходится балансировать нагрузку, поддерживать отказоустойчивость и т.д.;
- разработчики должны приложить дополнительные усилия для внедрения и поддержания механизмов взаимодействия между сервисами, обработки различных форматов сообщений, реализации более дорогостоящих удаленных вызовов, должны использовать более грубые удаленные API-интерфейсы;
- благодаря распределенному развертыванию тестирование может стать сложным и утомительным;
- при «перераспределении обязанностей» между компонентами могут возникнуть проблемы, несогласованная работа может привести к дублированию усилий; обработка функций, которые охватывают более одной службы, требует взаимодействия и сотрудничества между различными командами;
- увеличение числа услуг может привести к информационным барьерам;
- когда число услуг увеличивается, интеграция и управление системой в целом может стать сложным;
- архитектура обычно приводит к увеличению потребления памяти и пр.

Таким образом, этап проектирования архитектуры, выбора технологий для реализации компонентов системы становится самым сложным и ответственным этапом разработки.

В разрабатываемой системе выделены следующие задачи, реализуемые как отдельные *микросервисы*: сбор данных; формирование оценки объектов; выдача рекомендаций.

Для служебных вычислений выбрана архитектура, основанная на *кластерах*, обеспечивающих прозрачность системы. В результате этого повышается надежность, сбалансированность и производительность платформы.

Общая архитектура рекомендательной системы представлена на рис. 3.

В качестве *основного кластера* для хранения и обработки данных используется программная платформа Hadoop – система, которая состоит из множества различных

компонентов, в совокупности создающих единую платформу. Данная платформа имеет открытый исходный код и предназначена для хранения и обработки данных, которые слишком велики для одного конкретного устройства или сервера. Сила Hadoop заключается в ее способности масштабироваться по тысячам товарных серверов, которые не разделяют память или дисковое пространство. Hadoop делегирует задачи через рабочие серверы, которые называются «рабочими узлами» или «подчиненными узлами», существенно используя возможности каждого устройства и запуская их одновременно. Для целей работы используется *распределенная файловая система* Hadoop (HDFS) – масштабируемая файловая система, которая распределяет и хранит данные на всех машинах в кластере Hadoop (группе серверов), которая позволяет хранить огромные файлы.

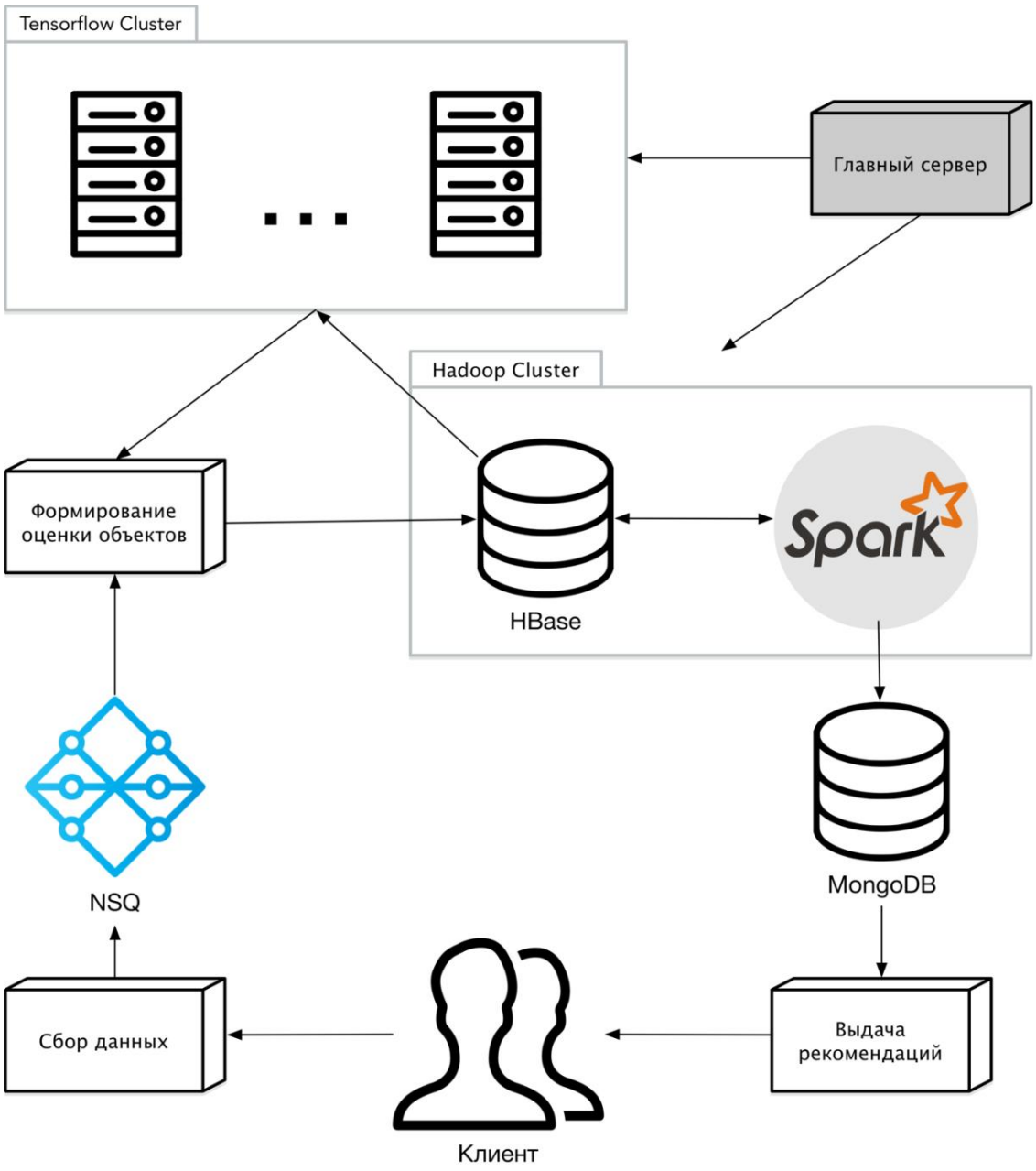

Рис. 3. Упрощённая архитектура рекомендательной системы

На кластере Hadoop в качестве *базы данных* используется Apache Hbase. Данная база данных является колоночной, работает на HDFS. Она масштабируется для хранения больших объемов разреженных данных. В схеме Big Data она соответствует категории хранилища и является альтернативным или дополнительным вариантом хранилища данных.

Для *обработки данных на кластере* используется Apache Spark. Spark – это среда исполнения, которая работает с файловой системой, чтобы распределять данные по кластеру и обрабатывать эти данные параллельно. Spark также принимает набор инструкций из приложения, написанного разработчиком. Эта среда поддерживает разные языки программирования (Java, Python и Scala). Данный кластер использует библиотеку MLlib.

Для *обучения нейронной сети*, которая формирует оценки объектов по данным пользователей, используется кластер TensorFlow. TensorFlow – это одна из самых популярных библиотек для машинного обучения. Данная библиотека использует тензоры и графы в качестве основы. Кластер TensorFlow – это несколько компьютеров, на которых запущен сервер TensorFlow, объединенных мастер-сервером.

Первичная задача *сервиса сбора данных* – получить данные о просмотренных объектах от пользователей. Для ожидания данных от клиента необходим отдельный сервис. После отправки данных пользователя на сервер сбора данных эти данные попадают на сервер NSQ – распределенной платформы обмена сообщениями в реальном времени. Эта платформа поддерживает распределенные и децентрализованные топологии, обеспечивая отказоустойчивость и высокую доступность в сочетании с надежной гарантией доставки сообщений, а также горизонтально масштабируется. Встроенное обнаружение упрощает добавление узлов в кластер, поддерживает доставку сообщений. От NSQ данные распределяются на *микросервисы формирования оценки объектов*, где есть обученная модель нейронной сети, которая формирует оценку, если она отсутствует, и накапливает пакет сообщений, чтобы отправить информацию на сервер базы данных.

Архитектура *сервиса формирования предложений* включает три компонента: главный сервер, кластер Hadoop и компонент выдачи рекомендаций. Сервис формирования предложений начинает свою работу в Hadoop-кластере, где по сигналу главного сервера происходит формирование предложений для пользователей на платформе Apache Spark. Сформированные предложения передаются на сервер баз данных MongoDB, где происходит репликация данных на другие серверы MongoDB для работы с микросервисом выдачи рекомендаций. Репликация – процесс копирования и синхронизации данных на нескольких серверах – обеспечивает избыточность для повышения доступности данных, отказоустойчивости.

При необходимости *клиент* (*приложение основного сервиса*) отправляет запрос на получение предложений для конкретного пользователя. Логика обработки предоставленных рекомендаций и выдача их конечному пользователю реализуется разработчиками приложения самостоятельно.

Таким образом, система работает с двумя СУБД: для сбора данных – HBase; для выдачи рекомендаций – MongoDB. HBase обеспечивает быстрый доступ к всей таблицам для их обработки. MongoDB обладает достаточно большим значением RPS (*request per second*), что позволяет обработать большое число запросов, а также легко масштабируется для нужд микросервисов.

Обмен данными между сервером и клиентом необходим в двух случаях: при сборе данных о просмотренных объектах и при выдаче рекомендаций. Весь обмен данными предполагает собой использование REST API.

Арххитектура подсистемы анализа данных

Архитектура *модуля анализа данных* включает микросервисы работы с исходными данными, для поиска дубликатов, для группировки объектов и поиска ассоциативных правил (рис. 4). Каждый микросервис имеет доступ только к своей базе данных. Так как микросервисам, которые реализуют основные задачи, необходим доступ к данным, которые загружены в систему клиентом, *вводится сервис для загрузки и доступа к общим данным*. Микросервисы *поиска дубликатов*, *группировки* и *поиска ассоциативных правил* по запросу получают необходимые данные от *микросервиса для работы с данными* для их дальнейшей обработки. Существует также коммуникация между *микросервисом де-дупликации* и *микросервисом группировки*, так как процесс поиска дубликатов происходит на

предварительно кластеризованных данных. Микросервис группировки по запросу возвращает сгруппированные данные сервису поиска дубликатов. *Коммуникация между всеми микросервисами* осуществляется с помощью вызова удаленных процедур (RPC). В данном случае этот метод реализуется с помощью технологии gRPC от Google, одним из применений которой как раз является обеспечение связи между микросервисами. В качестве инструмента описания типов данных и сериализации gRPC использует Protobuf, отличительной особенностью которого является производительность даже при больших объемах данных. Данная технология поддерживает работу с большим количеством языков программирования, а также работает быстрее чем REST, так как в качестве протокола передачи структурированных данных использует Protobuf.

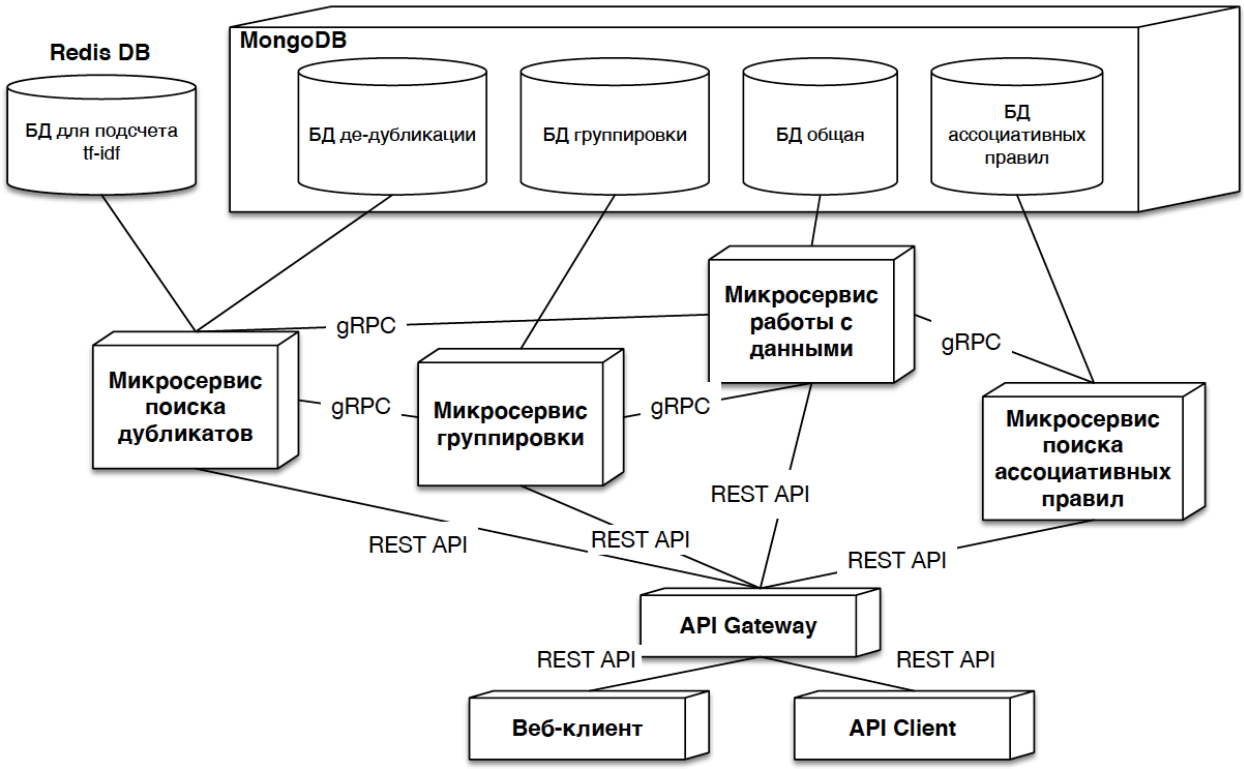

Рис. 4. Упрощённая архитектура подсистемы анализа данных

Каждый микросервис управляет своей базой данных: сервис де-дупликации имеет доступ к базе данных Redis для хранения данных по подсчету *tf-idf* (необходим для сравнения строковых полей) и к базе данных де-дупликации, где хранятся результаты работы алгоритмов по поиску дубликатов; сервисы группировки, поиска ассоциативных правил также имеет свои базы данных, где хранятся результаты их работы, и по запросу от клиента они выдают необходимую информацию, основываясь на этих данных. Микросервис для работы с данными имеет доступ лишь к общей базе данных, куда он заносит данные по запросу от клиента, либо забирает по запросу от микросервисов, выступающих в роли клиентов. В общей базе данных хранятся исходные данные клиента, которые тот поместил в систему для анализа. Таким образом, реализованы все основные условия микросервисной архитектуры: каждый сервис выполняет свою задачу, является отдельным узлом, имеет собственное пространство данных.

Для обеспечения взаимодействия данных модулей с клиентами необходим сервер API Gateway, который перенаправляет запросы клиентов на необходимые микросервисы. Это позволяет создать единый API для доступа к микросервисам. Клиенты не используют адреса всех узлов микросервисов непосредственно, так как это снижает масштабируемость. Для реализации этого прокси-сервера выбрана платформа nginx, которая позволяет также

балансировать нагрузку между узлами. Прокси-сервер взаимодействует с микросервисами посредством REST API.

Система имеет веб-клиент, а также предоставляет возможность создания собственного клиента (имеет API для обращения к функциям системы из программного кода). Клиенты связываются с прокси-сервером с помощью REST API.

Автоматическая *группировка объектов* – задача, решаемая методом кластеризации в машинном обучении. Так как в рамках данной работы разрабатывается исследовательский прототип модуля анализа, для группировки объектов выбран наиболее простой и доступный алгоритм *k-means*. Минус алгоритма *k-means* в том, что необходимо подбирать количество кластеров вручную, однако существуют подходы, которые могут помочь оценить количество кластеров. В данной работе применен метод *Affinity propagation* [4], также реализованный в библиотеке *scikit learn*. Для получения более точных результатов в дальнейшем планируется включить другие, более эффективные методы кластеризации для обеспечения еще большей адаптируемости системы.

Одним из главных условий «правильности» рекомендаций является обеспечение *уникальности данных*. Каждая ликвидированная копия объекта может способствовать увеличению процента точности при формировании предложений. В данной работе предлагается такой алгоритм поиска дубликатов, который на основе полученной функции сходства между парой объектов находит кластеры дубликатов. Функция сходства должна определять вероятность того, что два объекта являются дубликатами, получая на вход разницу между ними. Для создания функции сходства могут быть использованы методы машинного обучения с учителем. Наиболее универсальным методом, который зарекомендовал себя на практике, является *gradient boosting decision trees* [5]. Использование этого метода в данной работе не отвергает другие, возможно, более эффективные решения. Для того чтобы модуль анализа был как можно более универсальным и мог работать с различного вида данными, в дальнейшем необходимо добавлять другие методы решения задачи поиска функции сходства.

Поиск закономерностей между событиями в базе данных (например анализ покупательских корзин и поиск продуктов, покупаемых совместно) в анализе данных ведется с помощью *ассоциативных правил* [6]. Поиск дополняющих объектов (часто встречающихся комбинаций в данных) также можно осуществить с помощью ассоциативных правил. В основе ассоциативных правил лежит понятие транзакции – произошедшее одновременно множество событий. Масштабируемым алгоритмом ассоциативных правил является алгоритм *Apriori* [7, 8].

На основе реализации этих средств можно формировать предложения при выборе пользователем очередного объекта при составлении им неких комбинаций объектов (товаров, услуг и т.п.), а также проводить анализ и решать другие задачи, которые специфичны для каждой области, например, по оценке покупательских корзин может решаться задача по выгодному размещению товаров на полках в супермаркете.

Заключение

Теоретическая значимость работы заключается в разработке универсального подхода к формированию рекомендаций на основе данных из разных предметных областей. Для повышения качества рекомендаций предложено использовать композицию методов анализа данных, которая позволяет эффективно решать задачи кластеризации, поиска дубликатов и ассоциаций в рекомендательных системах нескольких спектров областей.

Данный подход может использоваться для разработки рекомендательных систем для групп предметных областей со специфическими требованиями, которые сложно учесть при разработке.

На основе предложенного подхода создан исследовательский прототип универсальной рекомендательной системы, которая охватывает все этапы процесса построения сервиса рекомендаций: от сбора данных о пользователях до формирования личных рекомендаций на основе полученных данных. Универсальность разработанной

системы обеспечивает её преимущества перед другими продуктами, редставленными на рынке: сервис может быть настроен на потребности множества компаний, работающих в различных областях, при этом не потребуются затраты на его разработку, установка (подключение) и настройка выполняются в минимальные сроки и не требуют высоккой квалификации персонала компаний.

При реализации системы использованы методы интеллектуального анализа данных и машинного обучения, методы объектно-ориентированного проектирования и программирования, а также следующие технологии: Apache Hadoop, Apache Spark, TensorFlow.

Разработанный прототип прошёл апробацию для нескольких областей и показал практическую значимость подхода.

Библиографический список

1. *Linden G.* Amazon.com recommendations: itemto-item collaborative filtering / *G. Linden, B. Smith, J. York* // IEEE Internet Computing. – 2003. – No. 7(1). – P. 76-80.
2. *Jones A.* Marketing Personalization: Maximizing Relevance and Revenue / *A. Jones* // VB Insight. 2015.
3. *Hahsler M.* recommenderlab: A Framework for Developing and Testing Recommendation Algorithms: Technical Report / *M. Hahsler* // Texas: Southern Methodist University. 2011.
4. *Frey B.J.* Clustering by passing messages between data points / *B.J. Frey, D. Dueck* // Science. – 2007. – No. 315 (5814) – P. 972-976.
5. *Friedman J.* Greedy Function Approximation: A Gradient Boosting Machine / *J. Friedman* // The Annals of Statistics. – 2001. – Vol. 29, No. 5 – P. 1189-1232.
6. *Kumar J.* A Survey: On Association Rule Mining / *J. Kumar, N. Tiwari, M. Ramaiya* // International Journal of Engineering Research and Applications (IJERA). – February 2013 – Vol. 3, Issue 1 – P. 2065-2069.
7. *Abaya S.A.* Association Rule Mining based on Apriori Algorithm in Minimizing Candidate Generation / *S.A. Abaya* // International Journal of Scientific & Engineering Research. – 2012. – Vol. 3, Issue 7.
8. *Kaur C.* Association Rule Mining using Apriori Algorithm: A Survey / *C. Kaur* // International Journal of Advanced Research in Computer Engineering & Technology (IJARCET). – June 2013 – Vol. 2 – P. 2081-2084.

L.N. Lyadova, K.M. Mal'kova, M.V. Timofeev

ARCHITECTURE OF THE RECOMMENDER SYSTEM,
ADAPTED TO CUSTOMER DOMAINS

Abstract: Requirements imposed to the universal recommendatory systems integrated with services of the third-party developers are described. The descriptions of the system architecture, adapted to different domains, of its components and features of their implementation are presented.

Keywords: recommendation system, customization, microservice architecture, data analysis.