Применение NoSQL для построения рекомендательных сервисов реального времени

© П.А. Клеменков
Московский Государственный Университет имени М.В. Ломоносова,
Москва
parser@cs.msu.su

Аннотация

В статье обсуждаются вопросы анализа взаимодействия пользователя с веб-приложением, методы проведения подобного анализа и их недостатки. Приводится реализация новостного рекомендательного сервиса с использованием существующих подходов. Описывается новый подход к построению рекомендательных систем, работающих в режиме, близком к режиму реального времени, с использованием технологии NoSQL.

1. Введение

Основным отличием приложений Веб 2.0 от их более старых аналогов является анализ взаимодействия пользователя с приложением и использование результатов этого анализа для модификации контента и его представления. Темпы развития сети Интернет диктуют создателям современных веб-приложений необходимость очень быстро адаптировать контент под предпочтения пользователей. Наиболее востребованным решением этой задачи являются рекомендательные системы, способные анализировать поведение пользователя, его склонности, и предлагать наиболее интересное наполнение. Проблема с подобными системами заключается в том, что они недостаточно быстро реагируют на постоянно изменяющийся поток входных данных. Особенно подвержены этому новостные ресурсы. Такое поведение связано не столько с алгоритмами, применяемыми для выявления пользовательских предпочтений, сколько с архитектурными особенностями той инфраструктуры и библиотек, которые широко используются для подобного анализа. В данной статье представляется подход к организации новостного рекомендательного сервиса, призванного максимально устранить задержки в пересчете рекомендаций и обеспечить работу в режиме, близком к режиму реального времени.

2. Методы веб анализа

Сегодня для анализа взаимодействия пользователя с веб-приложением применяются два основных подхода:

- аналитика в реальном времени
- отложенная пакетная обработка логов доступа к веб-приложению

У каждого из этих подходов есть преимущества и недостатки, на которых стоит остановиться подробней.

2.1 Аналитика в реальном времени

Суть подхода заключается в том, что в ответ на взаимодействие пользователя с веб-приложением специально установленный фрагмент кода (счетчик) генерирует определенные события, обрабатываемые приложением-анализатором в реальном времени. Очевидно, что основным преимуществом подобной парадигмы является немедленное получение результатов и их обновление. Однако методы, применяемые при анализе данных в реальном времени наиболее подходят для различных статистических расчетов (CTR, Churn Rate). При этом целые классы приложений не могут быть реализованы предложенными средствами.

2.2 Отложенная пакетная обработка логов доступа к веб-приложению

Этот подход строится на сборе логов доступа к веб-приложеню и их последующей обработке большими частями. Имея срез данных о взаимодействии пользователей с приложением за определенный период, возможно строить сложные модели поведения и применять их, например, для выдачи рекомендаций. Современные фреймворки (напр. Apache Hadoop) обеспечивают высокую производительность, реализуя потоковую обработку больших объемов данных с использованием метода параллельных вычислений MapReduce [6,9].

3. Рекомендательный сервис проекта Рамблер-новости

Рекомендательный сервис проекта Рамблер-новости основывается на объединении пользователей в группы по схожести интересов и вычислении наиболее популярных среди групп новостей в заданном временном окне.

3.1 Реализация сервиса

Суть алгоритма заключается в том, что все пользователи идентифицируется уникальными идентификаторами. Эти идентификаторы связываются с каждым HTTP-запросом к новостным ресурсам (если, конечно, запрос содержал заголовок Cookie с корректным значением). Таким образом поведение пользователя на сайте характеризуется подмножеством логов доступа к веб-серверам. Подсчитав схожесть каждого подмножества со всеми другими, можно объединить пользователей в группы с похожими предпочтениями.

В качестве меры схожести множеств естественно использовать коэффициент Жаккарда. Однако проблема заключается в том, что время работы алгоритма подсчета этого коэффициента на нескольких миллионах множеств с сотнями и тысячами элементов является неприемлемо большим. В качестве оптимизации широко применяется вероятностный алгоритм MinHash [5]. Основная идея этого алгоритма заключается в вычислении вероятности равенства минимальных значений хеш-функций элементов множеств. Очевидно, что чем больше одинаковых элементов в двух сравниваемых множествах, тем выше указанная вероятность. А так как вычисление сигнатуры множества (минимумов используемых хеш-функций) происходит только один раз, а размер сигнатуры фиксирован, то вычислительная сложность решаемой задачи резко сокращается.

Для вычисления новостных рекомендаций было принято решение производить обработку логов доступа веб-серверов Рамблер-новостей во временном окне 5 дней. Средний объем логов за указанный период составляет примерно 7 ГБ. Для реализации алгоритма был выбран фреймворк Hadoop, являющийся стандартом де-факто для потоковой обработки больших объемов данных.

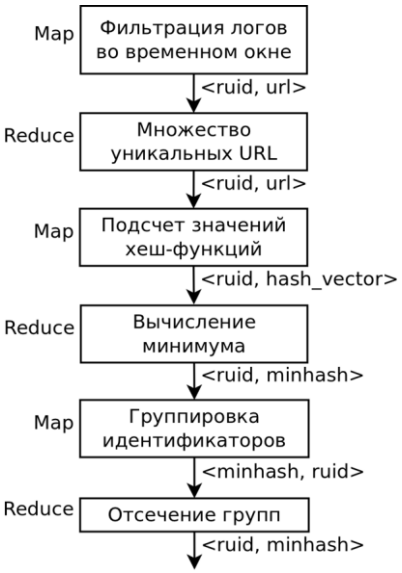
Алгоритм вычисления рекомендаций был реализован в виде последовательности MapReduce задач, разделенных на два этапа: подсчет групп пользователей во временном окне 5 суток и подсчет рекомендаций для групп во временном окне 5 часов. Первый этап составляют следующие ступени (см. рис. 1):

1. Фильтрация логов во временном окне 5 суток и генерация множества уникальных URL для каждого идентификатора пользователя.

2. Подсчет значений хеш-функций для всех уникальных URL каждого пользователя и вычисление минимума, который становится идентификатором группы.

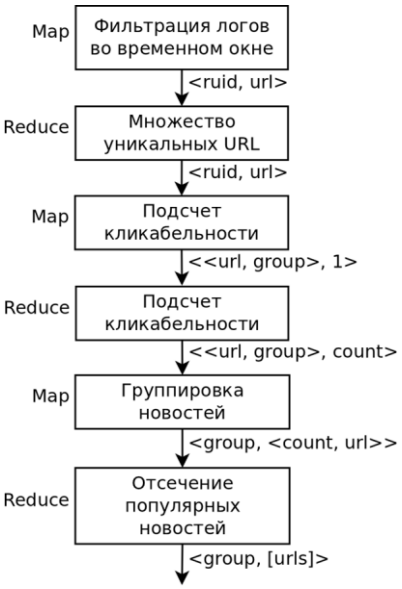
3. Подсчет численности групп и отсечение ~100 групп с наибольшей численностью. Необходимо заметить, что порог отсечения вычисляется статистически, поэтому имеет место небольшая дисперсия количества групп. Однако на производительность и поведение алгоритма это влияет незначительно.

Рис. 1. **Схема работы первого этапа Hadoop реализации**



Также необходимо отметить, что первоначальная реализация алгоритма имела еще один шаг, который позволял строго отсечь необходимое количество групп, но ради сокращения вычислений, им было решено пренебречь.

Рис. 2. **Схема работы второго этапа Hadoop реализации**



Второй этап разделен на следующие ступени (см. рис. 2):

1. Фильтрация логов во временном окне 5 часов и генерация множества уникальных URL для каждого идентификатора пользователя.

2. Подсчет кликабельности новостей во всех группах.

3. Отсечение заданного количества наиболее популярных новостей в каждой группе.

Получающиеся в результате отображения иден-

тификаторов в группы и групп в популярные новости загружаются в хранилище Redis, позволяющее запрашивать список рекомендаций для данного пользователя в реальном времени.

3.2 Производительность

Приведенная реализация алгоритма использовалась в продуктивном окружении проекта Рамблер-новости более полугода, показывая приемлемое время работы. На Hadoop кластере из 8 узлов первая ступень обсчитывалась примерно 7 минут, а вторая 3.5-4 минуты, при условии, что другие задачи не выполнялись параллельно. Необходимо отметить, что важным фактором производительности MapReduce задач является верный выбор количества мапперов и редьюсеров. Выбор количества мапперов производился фрейдистском автоматически. Экспериментальным путем было выяснено, что оптимальное количество редьюсеров в данной конфигурации — 16.

3.3 Проблемы

Внимательно изучив получившуюся архитектуру и приняв во внимание проблемы, возникшие при реализации рекомендательного сервиса, можно отметить следующие аспекты:

1. Загрузка логов в HDFS (Hadoop Distributed File System) и их обработка — две несвязанные задачи. В нашем случае синхронизация логов выполнялась с помощью утилиты rsync, а вычисление разности между файлами в директории синхронизации и файлами в HDFS, а также загрузка новых файлов — с помощью специально написанного Makefile и shell-скриптов.

2. В Hadoop отсутствует возможность получать данные из разных источников. В частности, результаты работы первого этапа алгоритма приходилось передавать в окружение второй ступени второго этапа в виде файла в кеше Hadoop. При том, что этот файл может иметь весьма внушительный размер, MapReduce задачи на всех узлах могут столкнуться с проблемой нехватки памяти.

3. Задачи подсчета рекомендаций и их использования также не являются связанными и выполняются разными инструментами. В нашем случае Hadoop и Redis.

4. Ну и самое главное — пакетный потоковый режим работы Hadoop не позволяет хоть сколько-нибудь приблизиться к реальному времени пересчета результатов.

Отсюда возникает вопрос: можно ли решить все вышеперечисленные проблемы, воспользовавшись другим подходом? В следующей части статьи я опишу архитектуру подобного решения с применением NoSQL хранилищ данных.

4. Введение в NoSQL

Термин NoSQL впервые был использован в 1998 году для описания реляционной базы данных, не использовавшей SQL. Он был вновь подхвачен в 2009 году и использован на конференциях приверженцами нереляционных баз данных. Основной движущей силой развития NoSQL хранилищ стали веб-стартапы, для которых важнейшей задачей является поддержание постоянной высокой пропускной способности хранилища при неограниченном увеличении объема данных. Рассмотрим основные особенности NoSQL подхода, делающие его таким привлекательным для высоконагруженных веб-проектов [7,8]:

1. **Исключение излишнего усложнения.** Реляционные базы данных выполняют огромное количество различных функций и обеспечивают строгую консистентность данных. Однако для многих приложений подобный набор функций, а также удовлетворение требованиям ACID являются излишними.

2. **Высокая пропускная способность.** Многие NoSQL решения обеспечивают гораздо более высокую пропускную способность данных нежели традиционные СУБД. Например, колоночное хранилище Hypertable, реализующее подход Google Bigtable, позволяет поисковому движку Zvent сохранять около миллиарда записей в день. В качестве другого примера можно привести саму Bigtable [3], способную обработать 20 петабайт информации в день.

3. **Неограниченное горизонтальное масштабирование.** В противовес реляционным СУБД, NoSQL решения проектируются для неограниченного горизонтального масштабирования. При этом добавление и удаление узлов в кластере никак не сказывается на работоспособности и производительности всей системы. Дополнительным преимуществом подобной архитектуры является то, что NoSQL кластер может быть развернут на обычном аппаратном обеспечении, существенно снижая стоимость всей системы.

4. **Консистентность в угоду производительности.** При описании подхода NoSQL нельзя не упомянуть теорему CAP. Следуя этой теореме, многие NoSQL хранилища реализуют доступность данных (availability) и устойчивость к разделению (partition tolerance), жертвуя консистентностью в угоду высокой производительности. И действительно, для многих классов приложений строгая консистентность данных — это то, от чего вполне можно отказаться.

5. Классификация NoSQL хранилищ

На сегодняшний день создано большое количество NoSQL хранилищ. Все они основываются на четырех принципах из предыдущего раздела, но могут довольно сильно отличаться друг от друга. Многие теоретики и практики создавали свои собственные классификации, но наиболее простой и общеупотребительной можно считать систему, основанную на используемой модели данных, предло-

женной Риком Кейтелем (Rick Cattel) [2]:

1. **Хранилища ключ-значение.** Отличительной особенностью является простая модель данных — ассоциативный массив или словарь, позволяющий работать с данными по ключу. Основная задача подобных хранилищ — максимальная производительность, поэтому никакая информация о структуре значений не сохраняется.

2. **Документо-ориентированные хранилища.** Модель данных подобных хранилищ позволяет объединять множество пар ключ-значение в абстракцию, называемую «документ». Документы могут иметь вложенную структуру и объединяться в коллекции. Однако это скорее удобный способ логического объединения, т.к. никакой жесткой схемы у документов нет и множества пар ключ-значение, даже в рамках одной коллекции, могут быть абсолютно произвольными. Работа с документами производится по ключу, однако существуют решения, позволяющие осуществлять запросы по значениям атрибутов.

3. **Колоночные хранилища.** Этот тип кажется наиболее схожим с традиционными реляционными СУБД. Модель данных хранилищ подобного типа подразумевает хранение значений как неинтерпретируемых байтовых массивов, адресуемых кортежами <ключ строки, ключ столбца, метка времени> [3]. Основой модели данных является колонка, число колонок для одной таблицы может быть неограниченным. Колонки по ключам объединяются в семейства, обладающие определенным набором свойств.

4. **Хранилища на графах**. Подобные хранилища применяются для работы с данными, которые естественным образом представляются графами (напр. социальная сеть). Модель данных состоит из вершин, ребер и свойств. Работа с данными осуществляется путем обхода графа по ребрам с заданными свойствами.

Таблица *1*

Классификация NoSQL хранилищ по модели данных

Тип	**Проекты**
Хранилища ключ-значение	Redis Scalaris Tokio Tyrant Voldemort Riak
Документо-ориентированные хранилища	SimpleDB CouchDB MongoDB
Колоночные хранилища	Bigtable HBase HyperTable Cassandra
Хранилища на графах	Neo4j

6. Построение рекомендательного сервиса Рамблер-новостей с помощью NoSQL

Вспоминая недостатки реализации рекомендательного сервиса на фреймворке Hadoop, можно отметить, что NoSQL хранилища кажутся приемлемым вариантом их устранения. NoSQL хранилища обеспечивают высокую пропускную способность данных как при чтении, так и при записи. Из этого следует, что логи доступа к веб-приложению можно записывать непосредственно в базу данных. Важно также отметить, что при использовании документо-ориентированных решений, логам можно придавать произвольный вид, не создавая жесткую схему. Это позволяет решать довольно интересную задачу хранения и обработки структурированных логов. К тому же механизм выборки документов по значениям атрибутов позволяет решать множество аналитических задач.

Большинство современных NoSQL решений реализуют парадигму вычислений MapReduce. Наряду с фундаментальным свойством горизонтального масштабирования, это дает возможность переносить алгоритмы, предназначенные для фреймворков типа Hadoop на хранилища NoSQL, получая все дополнительные преимущества.

Учитывая высокую пропускную способность операций чтения, задачу подсчета рекомендаций и их использования, можно не разделять. Следовательно обновленные рекомендации будут тут же доступны потребителям, приближая сервис к требованиям реального времени.

Далее следовало определиться с конкретным продуктом, который можно было бы использовать для реализации сервиса. Среди документо-ориентированных баз данных первоначальный выбор пал на проект Apache CouchDB [1]. CouchDB работает с документами, представленными в формате JSON. Для работы с документами предоставляется REST API. Для построения запросов к документам CouchDB и их анализа применяются так называемые «представления». По сути представление является обычной MapReduce задачей, которая может сохранять результаты выполнения в базе. Интересной особенностью модели данных CouchDB является то, что для индексации документов и представлений используются модифицированные B-деревья. Сохраняя все особенности и преимущества стандартного B-дерева, B-деревья CouchDB реализуют режим «только добавление». Это означает, что любые операции вставки, модификации и изменения записываются в конец файла, представляющего B-дерево на диске. Такая архитектура дает два основных преимущества: высокая скорость записи и возможность исполнять MapReduce задачи только на изменившихся данных. Однако, при всех своих преимуществах, CouchDB не подходила для нашей задачи. Во-первых, проект не поддерживает никакого языка запросов, что сильно затрудняет выборку документов по определенным

критериям. Во-вторых, важным критерием выбора была поддержка ссылок на другие документы. Подобная возможность есть в CouchDB, но работает она только на этапе эмиссии документа из map-задачи. К тому же, нет возможности создания ссылок на документы из других баз. В-третьих, неоптимизированное JSON представление документов приводит к увеличению трафика между клиентом и хранилищем, чего хотелось избежать.

Окончательный выбор пал на проект MongoDB[4]. Обладая всеми преимуществами CouchDB, это хранилище устраняет перечисленные недостатки и предоставляет дополнительные удобные возможности. Они будут упомянуты в следующем разделе, описывающем реализацию рекомендательного сервиса.

7. Реализация рекомендательного сервиса Рамблер-новости

Первая задача, которую предстояло решить — это запись логов в базу данных MongoDB. Первым делом требовалось определить, какое количество операций записи в секунду обеспечивала наша конфигурация. Стоит отметить, что тестовая конфигурация представляла кластер из двух узлов, на каждом их которых был запущен демон mongod без репликации. На одном их хостов запускался демон mongos, обеспечивавший шардинг документов. Для определения скорости записи был разработан простой скрипт, производивший загрузку суточных логов новостей в базу MongoDB. Лог состоял из 2770695 записей. Среднее время записи составило 18 минут 30 секунд. Таким образом средняя скорость записи в представленной конфигурации - 2496 операций/сек. Шардинг документов осуществлялся по атрибуту ruid (уникальный идентификатор пользователя). Подобный результат более чем достаточен для нашего сервиса, т.к. среднее количество запросов в секунду к веб-сайту Рамблер-новости существенно меньше. Однако, загрузка логов из ротированных лог-файлов нам совсем не подходила. Для удовлетворения требования реального времени необходимо было обеспечить загрузку логов в базу сразу после обработки запроса веб-сервером. Для этого, с помощью библиотеки ZeroMQ, был разработан специализированный демон, агрегировавший логи с нескольких фронт-эндов новостей в хранилище MongoDB. Необходимо отметить, что загрузчик логов не только производил их фильтрацию, представление в формате BSON и запись в базу, но и подсчет значений хеш-функций для каждого URL. Это было обусловлено двумя факторами: снижением времени вычислений и отсутствием приемлемых реализаций быстрого хеширования в языке JavaScript (на нем реализуются MapReduce задачи в MongoDB).

После того, как задача загрузки логов была решена, необходимо было перенести реализацию алгоритма подсчета рекомендаций с Hadoop на MongoDB. Возвращаясь к реализации первого этапа в разделе 3.1, можно отметить, что задачи фильтрации логов и подсчета значений хеш-функций для них реализуются загрузчиком. Поэтому оставалось перенести только подсчет минимальных значений хешей и отсечение групп с заданной численностью (см. рис. 3).

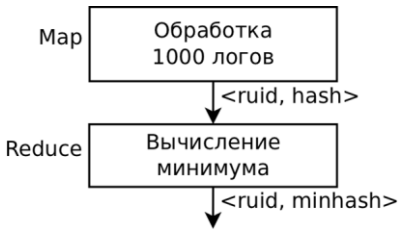


Рис. 3. Схема работы первого этапа NoSQL реализации

Стоит обратить внимание, что из новой реализации пропал этап отсечения групп по численности. В первоначальной реализации отсечение делалось, главным образом, для сокращения времени загрузки отображения <идентификатор пользователя → группа> в Redis. При использовании NoSQL хранилища подобной проблемы не возникает.

Возвращаясь к цифрам, задача подсчета минимального хеша для суточных логов (2770695 записей) заняла примерно 3 минуты 10 секунд. Это не сильно отличается от времени выполнения той же задачи на Hadoop кластере, и почему это происходит вполне очевидно. Однако здесь нам на помощь приходит вся мощь MongoDB. Во-первых, результаты MapReduce задач сохраняются в отдельной коллекции. Последующие вычисления можно производить только на добавленных с прошлого запуска логах, выполняя rereduce на получившихся результатах. Во-вторых, мощный язык запросов MongoDB позволяет осуществить выборку логов, добавленных с момента последнего запуска задачи. В нашей архитектуре задача сохранения времени последнего выполнения и перезапуск вычислений возложена на загрузчик логов. Важно отметить, что высокая производительность библиотеки ZeroMQ позволила не масштабировать загрузчик логов, поэтому проблем с синхронизацией времени не возникало. В-третьих, MongoDB поддерживает создание и поддержание индексов на атрибутах документов, что существенно ускоряет выборки. Сделав выводы из всего вышесказанного было принято решение перезапускать задачу подсчета минимального хеша после записи одной тысячи новых логов с выборкой по атрибуту timestamp документа. Данная задача без индекса завершалась в среднем через 3 сек, а с индексом — через 400-500мс, что уже существенно приблизило нас к требованиям реального времени.

Теперь перейдем ко второму этапу алгоритма — выработке рекомендаций. Здесь возникают три основные проблемы: выборка логов в заданном

временном окне, дополнительная фильтрация и ввод данных из нескольких источников. Выборку логов в заданном временном окне можно, как и на первом этапе, осуществлять запросом по атрибуту timestamp. Стоит отметить, что MongoDB реализует capped collections. Это коллекции с заранее определенным объемом. Если объем коллекции достиг заданного порога, то новые значения затирают старые. Это интересный подход к ротации, но для нашей задачи он не подходит, т.к. количество логов может меняться день ото дня. Дополнительная фильтрация осуществляется регулярными выражениями JavaScript, здесь нет никаких сложностей. Проблема ввода данных из нескольких источников решается с помощью механизма DBRef MongoDB. Он позволяет создавать ссылки на связанные документы в виде вложенных документов и получать к ним доступ при выполнении map-задач. Удобная особенность DBRef следует из отсутствия схемы документов и других ограничений — ссылаться можно на несуществующие документы и коллекции. Этим фактом пользуется загрузчик логов, создавая ссылки на группы, которых еще нет. Таким образом, первые две ступени второго этапа получилось объединить в одну: map-задача фильтрует выборку логов во временном окне 5 часов и возвращает пару <group_id:url, 1>, a reduce-задача подсчитывает количество кликов по каждой новости всех групп. Среднее время выполнения этой ступени — 350мс на той же тысяче логов. Третья ступень была просто адаптирована для исполнения MongoDB. Надо, правда, отметить, что отсечение заданного количества популярных новостей не производится. Эту задачу, с целью сокращения вычислений, было решено возложить на потребителя. Также следует сказать, что в последней ступени используется функция finalize, позволяющая видоизменить результаты reduce-задачи. В нашем случае функция finalize производит сортировку новостей в группах по числу кликов.

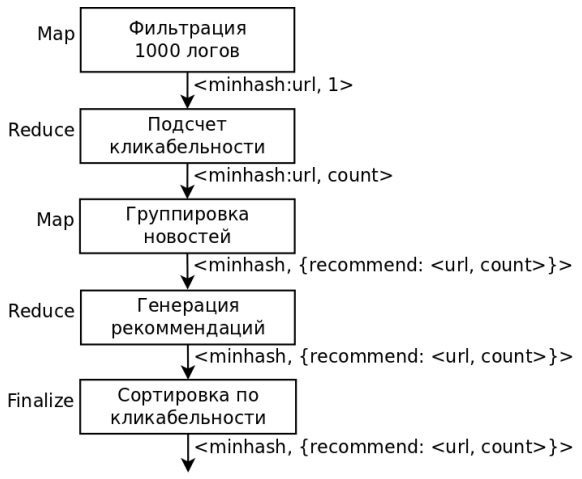


Рис. 4. Схема работы второго этапа NoSQL реализации

8. Проблемы, возникшие при реализации сервиса рекомендаций

Естественно, при реализации сервиса возник определенный набор трудностей, о которых важно упомянуть. Первая трудность — ротирование логов. Так как в MongoDB отсутствует механизм времени жизни ключей, задачу ротирования логов приходится решать периодическим запуском отдельной MapReduce задачи. К тому же во всех документах, требующих удаления, приходится явно хранить метку времени жизни. Вторая трудность заключается в том, что формат возвращаемых map-задачей значений должен совпадать с форматом значений, возвращаемых reduce-задачей. Из-за этого приходится создавать довольно сложные структуры, чего хотелось бы избежать. Третья трудность — это специфическое устройство шардинга в MongoDB. Ключи распределяются по узлам не равномерно, а группами. Из-за этого некоторые MapReduce задачи на небольшом количестве документов выполняются на одном узле, содержащем все ключи.

9. Заключение

В результате проведенного эксперимента удалось создать рекомендательный сервис, время пересчета рекомендаций в котором на каждую тысячу новых логов составляет 1.5-2 секунды. Для проекта Рамблер-новости подобный результат является удовлетворительным, т.к. 1000 новых запросов к сайту делается за чуть большее время. Стоит отметить, что алгоритм MinHash, как таковой, не предназначен для подсчета рекомендаций в режиме реального времени. Более того, эффективность новой реализации рекомендательного сервиса может оказаться ниже, чем предыдущая реализация с помощью фреймворка Hadoop. Однако целью данной работы было показать целесообразность применения NoSQL подхода к построению систем анализа данных в режиме близком к режиму реального времени. Сделанные выводы позволят реализовать на описанной платформе более подходящие рекомендательные алгоритмы, например Covisitation [5]. Важным свойством приведенной реализации является то, что задачи хранения и анализа данных удалось объединить с задачей предоставления доступа к результатам в единой системе, избежав накладных расходов на перемещение данных из одного источника в другой и улучшив общую производительность сервиса. Кроме того, предложенный подход упрощает решение повседневных задач сбора статистики о взаимодействии пользователя с веб-приложением путем анализа структурированных логов мощным языком запросов СУБД MongoDB. Можно утверждать, что применение NoSQL подхода к решению подобного класса задач является перспективным и может быть использован в продуктивном окружении высоконагруженных веб-приложений.

Литература

[1] J. Chris Anderson, Jan Lehnardt, Noah Slater. CouchDB: The Definitive Guide. O'Reilly Media, 2010.

[2] Rick Cattel. Scalable SQL and NoSQL data stores. ACM SIGMOD Record, 39(4), p. 12-27, ACM New York, NY, USA, December 2010.

[3] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber. Bigtable: a distributed storage system for structured data. Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation, vol. 7, p. 15-15, USENIX Association Berkeley, CA, USA, 2006.

[4] Kristina Chodorow, Michael Dirolf. MongoDB: The Definitive Guide. O'Reilly Media, 2010.

[5] Abhinandan S. Das, Mayur Datar, Ashutosh Garg, Shyam Rajaram. Google news personalization: scalable online collaborative filtering. Proceedings of the 16th international conference on World Wide Web, p. 271-280, ACM New York, NY, USA, 2007.

[6] Jeffrey Dean, Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation, vol. 6, p. 10-10, USENIX Association Berkeley, CA, USA, 2004.

[7] Jaroslav Pokorny. NoSQL databases: a step to database scalability in web environment. Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services, p. 278-283, ACM New York, NY, USA, 2011.

[8] Christof Strauch. NoSQL Databases. http://www.christof-strauch.de/nosqldbs.pdf

[9] Jason Venner. Pro Hadoop. Apress, 1st edition, 2009.

Building real-time recommendation services with NoSQL

P.A. Klemenkov

The paper discusses the analysis of user interacttion with a Web application, methods of conducting such an analysis, and their shortcomings. An implementation of the news recommendation service using existing approaches is described. A new NoSQL approach to building recommendation systems that operate in near real time is proposed.