П. А. Клеменков, Построение новостного рекомендательного сервиса реального времени с использованием NoSQL СУБД, *Информ. и её примен.*, 2013, том 7, выпуск 3, 14–21

DOI: http://dx.doi.org/10.14357/19922264130302

ИНФОРМАТИКА И ЕЁ ПРИМЕНЕНИЯ, 2013. Т. 7. Вып. 3. С. 14–21

ПОСТРОЕНИЕ НОВОСТНОГО РЕКОМЕНДАТЕЛЬНОГО СЕРВИСА РЕАЛЬНОГО ВРЕМЕНИ С ИСПОЛЬЗОВАНИЕМ NoSQL СУБД*

П. А. Клеменков[1]

Аннотация: Обсуждаются вопросы анализа взаимодействия пользователя с веб-приложением, методы проведения подобного анализа и их недостатки. Приведена реализация новостного рекомендательного сервиса с использованием существующих подходов. Описан новый подход к построению рекомендательных систем, работающих в режиме, близком к режиму реального времени, с использованием NoSQL (not only structured query language) системы управления базами данных (СУБД).

Ключевые слова: рекомендательные системы; minhash; mapreduce; 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 стали 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 представлен 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:

(1) аналитика в реальном времени;

(2) отложенная пакетная обработка логов доступа к веб-приложению.

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

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-приложению 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 (например, Apache Hadoop) обеспечивают высокую

*Статья рекомендована к публикации в журнале Программным комитетом конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» (RCDL-2012).

[1]Московский государственный университет им. М. В. Ломоносова, parser@cs.msu.su

производительность, реализуя потоковую обработку больших объемов данных с использованием метода параллельных вычислений MapReduce [1, 2].

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 идентифицируются 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 [3]. 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 вычисли-

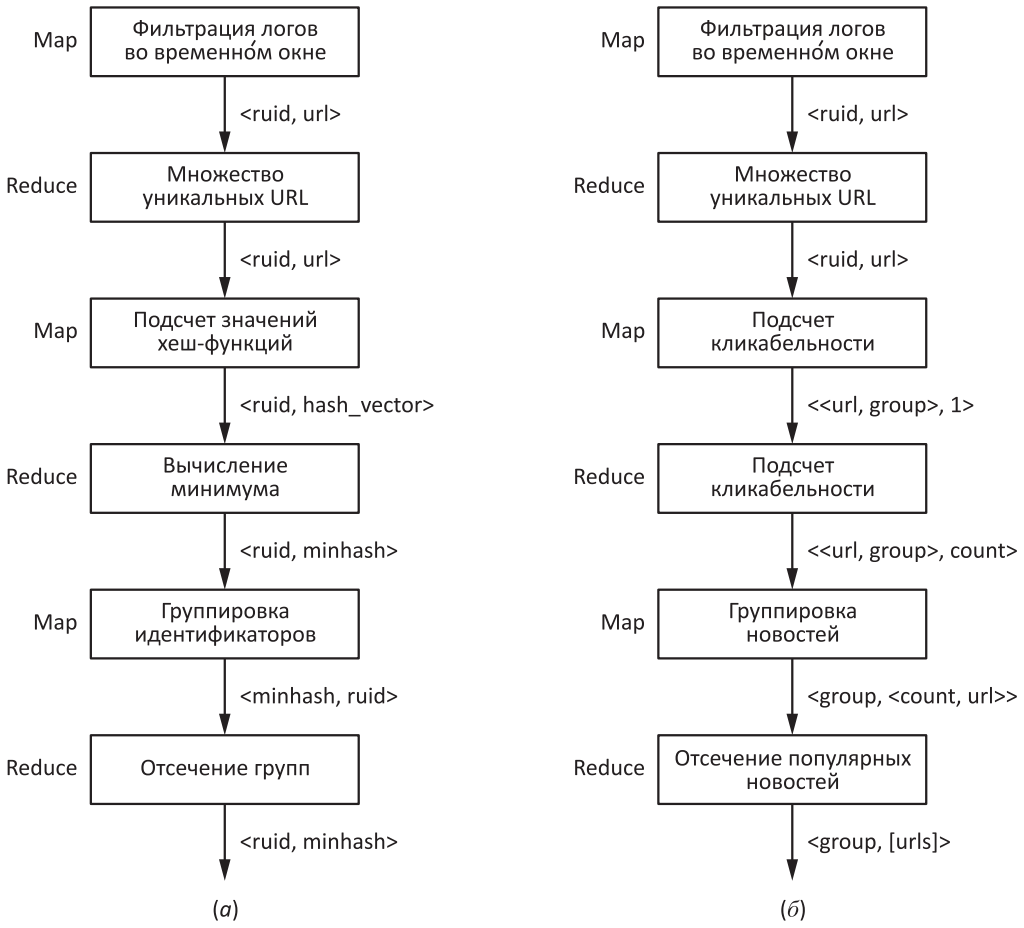

Рис. 1 Схема работы Hadoop-реализации на первом (*а*) и на втором (*б*) этапах

тельная сложность решаемой задачи резко снижается.

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

Алгоритм вычисления рекомендаций был реализован в виде последовательности MapReduce-задач, разделенных на два этапа: подсчет групп пользователей во временно́м окне 5 сут. и подсчет рекомендаций для групп во временно́м окне 5 ч. Первый этап составляют следующие ступени (рис. 1, *а*).

1. Фильтрация логов во временно́м окне 5 сут. и генерация множества уникальных URL (uniform resource locator) 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 числа групп. Однако на производительность и поведение алгоритма это влияет незначительно.

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

Второй этап разделен на следующие ступени (рис. 1, *б*).

1. Фильтрация логов во временно́м окне 5 ч и генерация множества уникальных URL для каждого идентификатора пользователя.
2. Подсчет кликабельности новостей во всех группах.
3. Отсечение заданного числа наиболее популярных новостей в каждой группе.

Получающиеся в результате отображения идентификаторов 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 мин, а вторая — 3,5–4 мин 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 автоматически. Экспериментальным путем было выяснено, что оптимальное число 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 не связанные задачи. В данном 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. Вследствие того 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 данном 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 будет описана архитектура подобного решения с применением NoSQL-хранилищ данных.

4 Введение в NoSQL

Термин NoSQL впервые был использован в 1998 г. для описания реляционной базы данных, не использовавшей SQL. Он был вновь подхвачен в 2009 г. 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 [4, 5].

1. **Исключение излишнего усложнения.** Реляционные базы данных выполняют огромное множество различных функций и обеспечивают строгую консистентность данных. Однако для многих приложений подобный набор функций, а также удовлетворение требованиям ACID (atomicity, consistency, isolation, durability) 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 [6], способную обработать 20 ПБ 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 (consistency, availability, partition tolerance). Согласно 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 число 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, предложенную Риком Кейтелем (см. табл.) [7].

Классификация NoSQL-хранилищ по модели данных

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

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 информации о структуре значений не сохраняется.
2. **Документно-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, так как 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 в хранилищах подобного типа подразумевает хранение значений как неинтерпретируемых байтовых массивов, адресуемых кортежами ⟨ключ строки, ключ столбца, метка времени⟩. Основой модели данных служит 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 (например, социальная сеть). Модель данных состоит из вершин, ребер и свойств. Работа с данными осуществляется путем обхода графа по ребрам с заданными свойствами.

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 документно-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, задачи подсчета рекомендаций и их использования можно не разделять. Следовательно, обновленные рекомендации будут тут же доступны потребителям, что приближает сервис к требованиям реального времени.

Далее следовало определиться с конкретным продуктом, который можно было бы использовать для реализации сервиса. Среди документно-ориентированных баз данных первоначальный выбор пал на проект Apache CouchDB [8]. CouchDB работает с документами, представленными в формате JSON (JavaScript Object Notation). Для работы с документами предоставляется REST API (REpresentation State Transfer Application Programming Interface). 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: высокую скорость записи и возможность исполнять MapReduce-задачи только на изменившихся данных. Однако при всех своих преимуществах CouchDB не подходила для решения поставленной 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 [9]. 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 число операций записи в секунду обеспечивала выбранная конфигурация. Стоит отметить, что тестовая конфигурация представляла собой 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 2 770 695 записей. Среднее время записи составило 18 мин 30 с. Таким образом, средняя скорость записи в представленной конфигурации — 2496 операций/с. Шардинг документов осуществлялся по атрибуту ruid (уникальный идентификатор пользователя). Подобный результат более чем достаточен для рассматриваемого сервиса, так как среднее количество запросов в секунду к веб-сайту Рамблер-новости существенно меньше. Однако загрузка логов из ротированных лог-файлов разработчика не устраивала. 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 (Binary JavaScript Object Notation) 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 подразд. 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 (рис. 2).

Стоит обратить внимание на то, 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 возникало.

Возвращаясь к цифрам, отметим, что задача подсчета минимального хеша для суточных логов (2 770 695 записей) заняла примерно 3 мин 10 с. Это не сильно отличается от времени выполнения той же задачи на Hadoop-кластере, и почему это

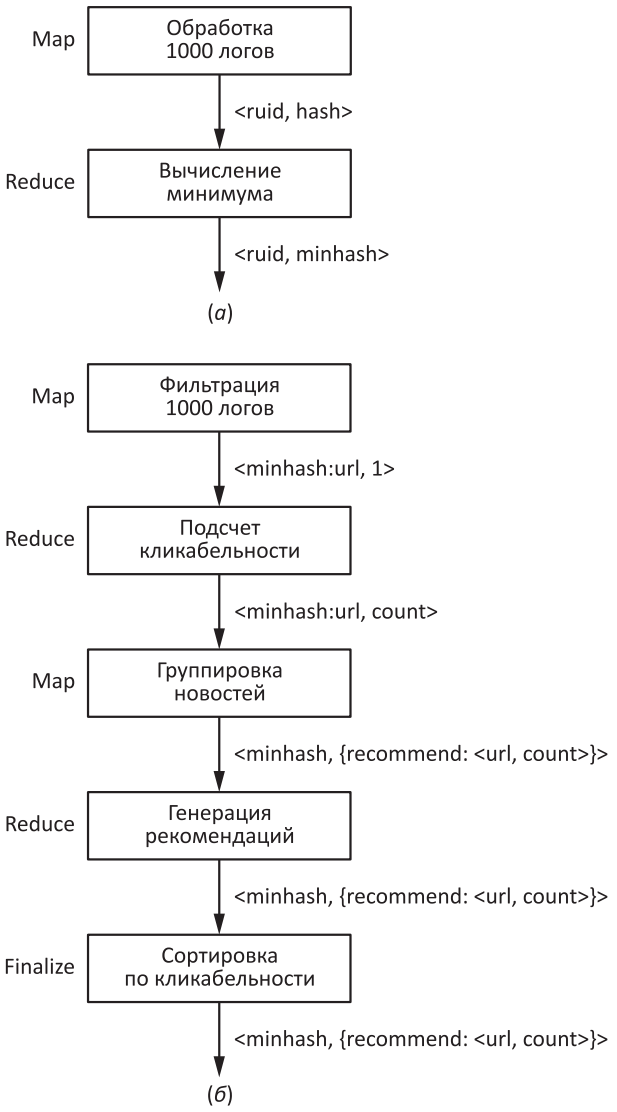

Рис. 2 Схема работы NoSQL-реализации на первом (*а*) и на втором (*б*) этапах

происходит, вполне очевидно. Однако здесь 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 предлагаемой 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. На основании 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 с, а с индексом — через 400–500 мс, что уже существенно приблизило разработку к требованиям реального времени.

Теперь перейдем ко второму этапу алгоритма — выработке рекомендаций (рис. 2, *б*). Здесь возникают три основные проблемы: выборка логов в заданном временно́м 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 рассматриваемой задачи он не подходит, так как 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 удалось объединить в одну: map-задача фильтрует выборку логов во временно́м окне 5 ч и возвращает пару $\langle \text{group\_id} : \text{url}, 1 \rangle$, а reduce-задача подсчитывает количество кликов по каждой новости всех групп. Среднее время выполнения этой ступени — 350 мс 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 объема вычислений было решено возложить на потребителя. Также следует сказать, что на последней ступени используется функция finalize, позволяющая видоизменить результаты reduce-задачи. В данном случае функция finalize производит сортировку новостей в группах по числу кликов.

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 числе 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 с. Для проекта Рамблер-новости подобный результат является удовлетворительным, так как 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 производитель-

ность 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 весьма перспективно и может быть использовано в продуктивном окружении высоконагруженных веб-приложений.

Литература

1. *Dean J., Ghemawat S.* MapReduce: Simplified data processing on large clusters // OSDI'04: 6th Symposium on Operating System Design and Implementation Proceedings. — Berkeley, CA, USA: USENIX Association, 2004. P. 137–149.
2. *Venner J.* Pro Hadoop. — N.Y.: Apress, 2009.
3. *Das A. S., Datar M., Garg A., Rajaram Sh.* Google news personalization: Scalable online collaborative filtering // 16th Conference (International) on World Wide Web Proceedings, 2007. P. 271–280.
4. *Pokorny J.* NoSQL databases: A step to database scalability in web environment // 13th Conference (International) on Information Integration and Web-Based Applications and Services Proceedings, 2011. P. 278–283.
5. *Strauch C.* NoSQL databases. http://www.christof-strauch.de/nosqldbs.pdf.
6. *Chang F., Dean J., Ghemawat S., Hsieh W. C., Wallach D. A., Burrows M., Chandra T., Fikes A., Gruber R. E.* Bigtable: A distributed storage system for structured data // 7th USENIX Symposium on Operating Systems Design and Implementation Proceedings. — Berkeley, CA, USA: USENIX Association, 2006. Vol. 7. P. 205–218.
7. *Cattel R.* Scalable SQL and NoSQL data stores // ACM SIGMOD Record, 2010. Vol. 39. No. 4. P. 12–27.
8. *Anderson J. C., Lehnardt J., Slater N.* CouchDB: The definitive guide. — Sebastopol: O'Reilly Media, 2010.
9. *Chodorow K., Dirolf M.* MongoDB: The definitive guide. — Sebastopol: O'Reilly Media, 2010.