УДК 004.7:004.75:004.9:004.738.5

А.А. Урсатьев

Некоторые программные среды аналитики больших данных и машинного обучения

Рассмотрено машинное обучение и распределенная обработка данных на *Apache Mahout*. Сопоставлены две его реализации – на основе использования парадигмы *MapReduce* и программной конструкции (*framework*) *Spark* с математической средой *Mahout Samsara*, которая создает семантически дружественные условия для линейной алгебры, построена по образу базового пакета в *R*, содержит алгебраический *DSL Scala* и оптимизатор выражений. Библиотека *ML Mllib* поддерживает универсальную масштабируемую линейную алгебру и включает в себя многие современные алгоритмы.

Розглянуто машинне навчання і розподілена обробка даних з *Apache Mahout*. Зіставлені дві його реалізації – з використанням парадигми *MapReduce* та програмної конструкції *Spark* з математичним середовищем *Mahout Samsara*, яке створює семантично дружні умови для лінійної алгебри, побудоване за образом базового пакета в *R*, містить алгебраїчний *DSL Scala* та оптимізатор виразів. Бібліотека *ML Mllib* підтримує універсальну масштабовану лінійну алгебру і включає в себе чимало сучасних алгоритмів.

Введение. Работа является завершающим этапом исследований[1], связанных с использованием теории и инструментальных средств обработки Больших Данных в задачах интеллектуального анализа ситуаций.

Программные вычислительные среды

Apache Mahout™: масштабируемость машинного обучения и интеллектуального анализа данных. Библиотека [34] – так сейчас представляет разработку *Apache*. Менее десяти лет назад проект *Apache Mahout* начал создаваться и уже в 2010 г. *Hortonworks* позиционировал *Mahout* как библиотеку масштабируемых алгоритмов машинного обучения на основе парадигмы *MapReduce* в *Apache Hadoop*®. Эта разработка объединяет два очень сложных вопроса: машинного обучения (*ML – Machine Learning*) и распределенной обработки больших данных на *Apache Mahout*.

Машинное обучение – дисциплина искусственного интеллекта – основано на методах, предоставляющих компьютеру возможности, не будучи явно запрограммированным, улучшать результаты работы, опираясь на предыдущий опыт. Оно тесно связано с интеллектуальным анализом данных и часто использует методы, заимствованные из статистики, теории вероятностей, распознавания образов и ряда других областей знаний. После сохранения данных на распределенной файловой системе *HDFS Hadoop*, *Mahout* предоставляет инструменты с возможностью доступа к большим наборам данных и автоматического поиска значимых закономерностей в них. Проект *Apache Mahout* стремится выполнить это быстрее и проще, поддерживая основные категории науки о данных (*Data Science*):

- коллаборативная (*collaborative*)[2] фильтрация (*CF*) – добыча, извлечение знаний о поведении пользователя и выработка рекомендаций – рекомендательная система по посещениям (сайты, документы, новости и пр.), товарам, рекомендательная система на основе рейтингов; система анализа текстов (задачи тематического моделирования (*topic modeling*) – поиск научной информации, выявление трендов и фронта исследований и др.) [35];

[1] Урсатьев А.А. Некоторые программные среды аналитики больших данных // УСиМ. – 2016. – № 3. – С. 29-41.

[2] Один из методов построения прогнозов, использующий известные предпочтения (оценки) группы пользователей для прогнозирования неизвестных предпочтений другого пользователя или метод, который даёт автоматические прогнозы исходя из накопленной информации об интересах и вкусах пользователей [36].

• кластеризация (таксономия) – анализ предметов в определенном классе (например, веб-страницы, газетные статьи) и организация их в виде групп по сходству признаков;

• классификация или дискриминантный анализ – обучение с учителем по существующим классификациям (обучающей выборке), а затем отнесение неклассифицированных элементов к классу, к которому в категории *лучший* относится данный объект. Например, задачи медицинской диагностики – (классификация вида заболевания, определение наиболее целесообразного способа лечения, прогнозирование длительности и исхода заболевания и др.), оценивание кредитоспособности заёмщиков и пр.;

• определение совместного появления признаков, часто извлекаемых из набора элементов (*frequent itemset mining*) – анализирует предметы в группах или термины в сессии запросов, а затем определяет, какие элементы, как правило, появляются вместе [37].

Несмотря на то, что концепция *MapReduce* не слишком приспособлена для итерационных вычислений и низколатентных приложений, *Mahout* хорошо справлялся с решением наиболее насущных задач обработки больших объемов данных, акцентируя на масштабируемости и облегчая использование сложных алгоритмов машинного обучения. Программная модель *MapReduce* распределенных параллельных вычислений эффективна для реализации пакетных задач в среде обработки *in-database analytics*, и ей не присуща интерактивность при обработке.

Повторное использование данных распространено во многих итерационных алгоритмах машинного обучения. Однако задания в *Map Reduce* используют однопроходную (*one-pass*) модель вычислений, а итерационные вычисления и интерактивный анализ нуждаются в повторной обработке данных и, следовательно, запуске нового задания, т.е. многократном обращении к основному циклу с обязательным сохранением на платформе *Hadoop* промежуточных результатов вычислений на диск после каждого прохода через данные. Вследствие этого неизбежны значительные и не всегда предсказуемые задержки. Использование ациклической модели потоков данных (*directed acyclic graph*, *DAG*), не предполагающей какой-либо возможности вложения циклов на всем протяжении процесса от *map* до *reduce*, приводит к тому, что абстракция *MapReduce* не соответствует подобным приложениям. Такая природа рассматриваемых приложений противоречит организации потока работ в пакете, поскольку реализация итераций средствами традиционного *Hadoop MapReduce* оказывается чрезвычайно длительной и приводит к заметному снижению производительности [38].

Производительности, обеспечиваемой моделью распределенных вычислений *Map Reduce*, для аналитики больших данных становится уже недостаточно. Текущие решения в ряде случаев не обеспечивают скорость реакции системы, необходимую для обработки заданий анализа, что затягивает процесс принятия решений. К сожалению, в большинстве существующих программных конструкций (*framework*$_s$), обеспечивающих многочисленные абстракции для доступа к вычислительным ресурсам кластера, единственным способом повторного использования промежуточных результатов вычислений является их перезапись на стабильную систему хранения (например, распределенную файловую систему). Для обеспечения отказоустойчивости *Hadoop MapReduce* в основном также обрабатывает информацию, сохраняемую на жестких дисках. В обоих случаях это влечет за собой существенные накладные расходы, связанные с репликацией данных, дисковым вводом/выводом (*disk I/O*) и сериализацией[3], которые могут доминировать время выполнения приложения. Признавая эту проблему, исследователи разработали специализированные структуры для некоторых приложений, требующих повторного использования данных. Тем не менее, эти программные структуры

[3] **Сериализация** (в программировании) – процесс перевода какой-либо структуры данных в виде последовательности байтов, например сохранения состояния *Java*-объекта для передачи по сети или сохранения на надежном носителе (например, в виде файла на диске) для последующего использования. *Java Serialization API* предоставляет разработчикам *Java* стандартный механизм управления сериализацией объектов [39].

поддерживают только определенные шаблоны вычислений и не обеспечивают абстракции для более общего случая повторного использования результатов вычислений [40].

В разделе настоящей работы о *Hadoop* уже упоминалось об использовании производительного вычислительного ядра *Spark* [11], которое представляет работающую в оперативной памяти программную конструкцию, способную конкурировать с *MapReduce*. *Apache Spark* – вычислительная платформа с открытым исходным кодом, аналогичная *Hadoop*. Несмотря на сходство, *Spark* представляет собой новую среду, обладающую рядом отличительных свойств, направленных на решение в вычислительном кластере задач, в которых рабочий набор данных многократно используется в параллельных операциях. Для оптимизации задач этого типа наборы данных временно помещаются в оперативную память для сокращения времени доступа к ним, и вводится понятие кластерных вычислений в памяти.

Сохранение данных в памяти повысило производительность на порядок величины. Благодаря сокращению операций чтения/записи с диска обеспечивается еще более высокое быстродействие, затраты времени на итерацию в *Spark* значительно меньше, чем у *MapReduce.* Кроме того, ускорение достигается путем хранения информации о каждом операторе в оперативной памяти, что также дает важное преимущество – все процессы, связанные с данными, происходят на одном и том же кластере, в одном и том же приложении. Все это способствует привнесению интерактивности при обработке данных.

Во-вторых, для сохранения масштабируемости и отказоустойчивости, присущих *Map Reduce*, в *Spark* вводится отказоустойчивая абстракция – устойчивые распределенные наборы данных *RDDs* (*resilient distributed datasets*), по сути, абстракция распределенной памяти данных, позволяющая программировать и выполнять параллельные вычисления в оперативной памяти на кластерах. *RDDs* обеспечивают параллельную модель программирования, формально они позволяют распределять существующую коллекцию записей по набору узлов и служат только для чтения. Эти коллекции неизменяемых *read-only* объектов, распределённых по узлам кластера, устойчивы, так как в случае потери части набора данных они восстанавливаются. Создаваться *RDDs* могут только через детерминированные операции[4] [41], называемые *трансформациями* (*transformations*), над данными во внешней системе хранения, например в файловой системе *HDFS* или любого источника данных в формате входящих данных *Hadoop InputFormat*, либо над другими *RDDs*. Эти преобразования (*трансформации*) проводятся над всеми элементами коллекции, в противоположность изменениям на уровне ячейки (*fine-grained*) в традиционных кластерных системах, благодаря тому, что в основу *RDD* положены крупноструктурные (*coarse-grained*) преобразования разделов [40].

Отказоустойчивость системы вычислений вследствие сбоя опирается на механизм восстановления, базирующийся на родословных *RDDs*, содержащих достаточно информации о том, как они были получены из других наборов данных, чтобы определить нужные разделы данных в хранилище. Механизм восстановления аналогичен используемому в вычислениях *MapReduce*, отслеживающих зависимости среди *DAG* задач. Тем не менее, в этой системе информация о родословной теряется по завершении работы, и требуется использование реплицирующей системы хранения, чтобы передавать данные между вычислениями. В отличие от этого, *RDDs* применяют родословную, чтобы вычислять и сохранять данные (*persist in-memory data*) в оперативной памяти, эффективно передавая их между вычислениями и не требуя привлечения затратных операций репликации данных и дискового ввода/вывода. *RDDs* используют *API* на основе *coarse-grained* преобразования, позволяющего им эффективно восстановить данные [40].

[4] *RDD* поддерживают два вида операций: *трансформации*, которые создают новый набор данных из существующих, и *действия* (*actions*), которые возвращают значение основной программе (программе-драйверу) после запуска вычисления на наборе данных.

Одна из важнейших возможностей *Spark* – кэширование данных в памяти. При сохранении *RDD*, каждый узел хранит определенные разделы данных и использует их в других вычислениях с этим же набором данных или данными, полученными от него. Это позволяет выполнять последующие действия более чем на порядок быстрее. Кэш в *Spark* отказоустойчив – если раздел *RDD* теряется, он будет автоматически пересчитан с использованием преобразований, посредством которых был создан. По умолчанию, каждый восстановленный *RDD* может быть пересчитан после каждого запуска вычислений на наборе данных. Тем не менее, *RDD* можно также сохранить в памяти, позволяя ему быть повторно и эффективно используемым в параллельных операциях, сокращая время доступа при последующем использовании. Кроме того, набор данных может быть сохранен путем другого уровня хранения на жестком диске. Кэширование в *Spark* – ключевой инструмент для итерационных алгоритмов и быстрого интерактивного использования [40].

Spark поддерживает распределенную обработку в памяти, так что создавая итерационные алгоритмы, можно не выписывать результат, устанавливаемый после каждого прохода данных. Таким образом, *Spark* создает возможность циклической обработки наборов данных, размещенных в памяти. Отход от пакетного режима *MapReduce* поддержан новым, не имеющим аналогов способом представления данных, – абстракцией *RDD*, допускающей эффективное использование повторных данных в широком диапазоне применений. *RDDs* позволяют явно сохранять промежуточные результаты в памяти, распределять и перераспределять данные на различных стадиях параллельных вычислений (рис. 6) и манипулировать ими с помощью набора операций [41], открывая путь к итерационным процессам, невозможным в случае *DAG*. Организация данных в виде *RDD* оказывается эффективной как в скорости обработки, так и в обеспечении высокой надежности, требуемой при распределенной работе на кластерах [38]. Вследствие этого также оптимизируется решение итеративных задач и интерактивного анализа данных при частой необходимости писать множество запросов к какому-то набору данных.

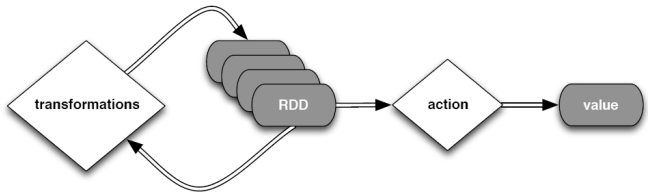

Рис. 6. Работа с *RDDs*

Все преобразования *RDDs* в *Spark* отложены или «ленивы» (*lazy*), т.е. операции *трансформации* над *RDDs* не приводят сразу к каким-либо вычислениям, а только создают очередной шаг на схеме, обещая выполнить операции только после запуска вычислений на наборе данных (рис. 6). Вместо этого они запоминают преобразования, применяемые к какому-то базовому набору данных, и создают их новые наборы из существующего тогда, когда результат должен быть возвращен в пользовательскую программу. Это позволяет оптимизировать необходимые расчеты, в случае необходимости восстановить потерянные разделы данных и вернуть результат вычислений в основную программу вместо большого набора данных. Это положение иллюстрируют рис. 6 и скрипт, приведенный ниже, – для решения задачи про логи[5] [18, 42]:

```
sc = ...                                          # создаём контекст (SparkContext)
rdd = sc.textFile("/path/to/server_logs")     # создаём указатель на данные
rdd.map(parse_line) \                                 # разбираем строки и переводим их в удобный формат
      .filter(contains_error) \                        # фильтруем записи без ошибок
      .saveAsTextFile("/path/to/result")        # сохраняем результаты на диск
```

В этом примере реальные вычисления начинаются только с последней строчки: *Spark* видит, что нужно материализовать результаты, и для этого начинает применять операции к данным. При этом здесь нет никаких промежуточных стадий – каждая строчка поднимается в память, разбирается, проверяется на признак

[5] Файл регистрации, протокол, журнал или лог (*log*) – файл с записями о событиях в хронологическом порядке. *Log*-файл веб-сайта – это текстовый файл, в котором регистрируются все запросы к сайту, а также все ошибки, связанные с этими запросами, хранится информация о посещениях, параметрах посещений сайта и др.

ошибки в сообщении и, если такой признак есть, тут же записывается на диск [42].

Spark нацелен (мотивирован) на ускорение работы двух типов приложений: итерационных алгоритмов и интерактивных инструментов анализа данных. Но *Spark* превосходит *Hadoop* не только в машинном обучении, но и в традиционных приложениях обработки данных. Он предлагает стандартные библиотеки – интегрированные в *Spark* модули (рис. 7) для аналитики больших данных [16, 43–45], в том числе:

- *MLLib* – машинное обучение,
- *GraphX* – работа с графами,
- *Spark-SQL* – интерфейс,
- *SparkStreaming* – потоковая аналитика.

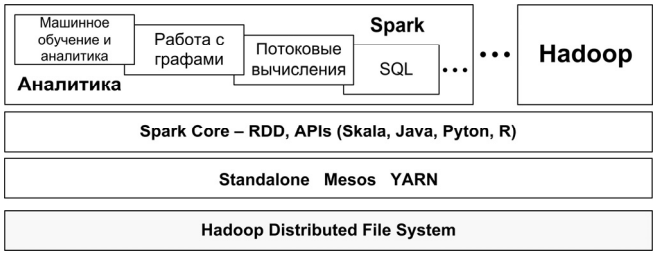

Рис. 7. *Spark* – интерактивно-аналитическая система

Все названные компоненты (см. рис. 7.) находятся поверх ядра (*Spark-core*). *MLLib* – простая и масштабируемая реализация машинного обучения. Библиотека содержит общие алгоритмы обучения, в том числе классификации, кластеризации и коллаборативной фильтрации, дополнительные инструменты в виде набора утилит [46]. *MLLib* обладает отличной интегрируемостью благодаря возможности подключения к развитым прикладным интерфейсам (доступен в *Java*, *Scala*, *Python* и *SparkR*) и простым развертыванием – библиотека инсталлируется на кластере *Hadoop* и работает с уже имеющимися данными. Обеспечивает высокую скорость обучения в сравнении с *MapReduce* и тестированием.

GraphX – компонент *Spark* для работы с графами. На высоком уровне *GraphX* расширяет *RDD* путем введения новой абстракции: направленный мультиграф со свойствами, прикрепленными к каждой вершине и ребру. Для поддержки вычислений графов, *Graphx* предоставляет набор основных операторов (например, *subgraph* – возвращает граф, содержащий только вершины, удовлетворяющие определенному условию; *joinVertices* – присоединяет новые вершины; *aggregateMessages* – агрегация сообщений к указанной вершине и др.). Кроме того, *GraphX* включает в себя растущую коллекцию алгоритмов на графах для упрощения аналитики соответственных задач [47], например, *PageRank*[6], *Personalized PageRank* – алгоритмы ссылочного ранжирования; *Shortest Path* – проблема нахождения кратчайшего пути; *Graph Coloring* – маркировка (раскраска) вершин графа.

Прикладной интерфейс *GraphX* обеспечивает работу с графами и графопараллельными вычислениями. Модуль обладает такими важными преимуществами как гибкость («бесшовность» работы как с графами, так и с коллекциями данных) и высокая скорость работы.

Spark SQL предназначен для структурированной обработки данных и реляционных запросов. Он вводит новую абстракцию данных под названием *DataFrames*, которая обеспечивает быстрые интерактивные запросы к структурированным и частично структурированным данным. *DataFrame* – распределенная коллекция данных, организованных в поименованных колонках. Это концептуально эквивалентно таблице в реляционной базе данных или кадра данных в *R* / *Python*, но с более богатой внутренней оптимизацией. *DataFrames* могут быть построены из широкого спектра источников: структурированных файлов данных, таблиц во внешних базах данных или существующих *RDD*. Эта функция объединяет лучшие практики создания таблиц данных.

DataFrames и *SQL* обеспечивают общий способ доступа к различным источникам данных и интеграцию *SQL*-запросов со всеми элементами фреймворка. Стандарт подключения к данным на сторонних носителях: соединение через *JDBC* или *ODBC*. По этим же интерфейсам обеспечивается также соединение для ин-

[6] *PageRank* (*PR*) — это числовая мера авторитетности страницы сайта для поисковой системы *Google*, рассчитываемая от количества и качества ссылок на эту страницу – как внешних, так и внутренних.

струментов бизнес-аналитики (*BI*). *DataFrame API* доступен в *Scala*, *Java*, *Python* и *R* [48].

SparkStreaming предоставляет пользователю возможность создавать и запускать приложения в потоковом режиме, а так же без существенных изменений в коде может осуществлять пакетную обработку.

Модуль *SparkStreaming* – расширение ядра *Spark API*, предоставляющее масштабируемую, отказоустойчивую потоковую обработку реальных данных с высокой пропускной способностью. Источниками данных могут быть различные системы: распределенная система передачи сообщений *Kafka* [49], платформа для работы с потоковыми данными *Kinesis* [50], *TCP*-сокеты или социальные сети, например *Twitter* и др. Эти данные можно обрабатывать с помощью сложных алгоритмов, скажем, машинного обучения, работы с графами и графопараллельными вычислениями, позволяющими выполнять расчеты в режиме реального времени (или близком к реальному времени). Полученные результаты могут быть помещены в файловые системы, базы данных и информационные панели (*live dashboards*), отражающие существующее (текущее) состояние вещей, близкое к реальному времени измеряемых процессов [51].

SparkStreaming принимает потоки входящих данных и делит их на пакеты, которые затем обрабатываются вычислителем *Spark* (рис. 8). Преимущество пакетной модели *Spark* – то, что она позволяет восстановить все промежуточные состояния и результаты на момент сбоя [52–54].

Рис. 8. Структура модели обработки потоковых данных

Здесь использована новая модель обработки – дискретизированные потоки *D-Streams*. Ключевая идея *D-Streams* структурировать потоковые вычисления на малых временных интервалах как серию не меняющих своего состояния объектов – пакетных вычислений (*deterministic batch computations*). Из поступающих данных, полученных в течение каждого интервала времени, формируется входной набор данных, обрабатываемый с помощью известных параллельных операций [40, 41].

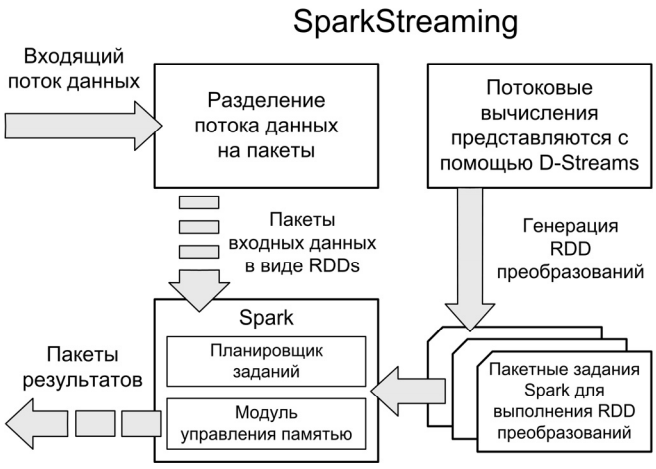

Рис. 9. Структура модуля *SparkStreaming*

На рис. 9 модуль *SparkStreaming* делит входящий поток данных на пакеты и сохраняет их в памяти *Sparks* [53]. Затем он запускает потоковое приложение для создания рабочих заданий *Spark* по обработке пакетов. При таком допущении – дискретизации входных потоков – можно автоматически обеспечить быстрый параллельный механизм восстановления[7] после сбоев, используя структуры данных, называемые *RDDs*. В кластерных системах потоковой обработки имеется и вторая, не менее важная, чем восстановление после сбоев или отказов узлов, проблема отставания (*stragglers*), вызванная медленными узлами[8]. Обе проблемы неизбежны в больших кластерах, поэтому потоковые приложения должны быстро восстанавливаться от последствий их возникновения.

[7] В *D-Stream* используется эффективная абстракция хранения (*storage abstraction*) – результаты сохраняются в отказоустойчивых распределенных наборах данных *RDDs* [53]. *D-Stream*, как и собственно *Spark* применяют детерминированные преобразования в гораздо более грубой (*coarse-grained*) степени разбиения на *RDD*-разделы, что обеспечивает простое восстановление при сбоях и облегчает учет использования системных ресурсов. В традиционных системах пакетной обработки, таких как *Hadoop*, промежуточный результат обработки данных сохраняется на диск, что приводит к значительным временным затратам [40, 53].

[8] В реализации [54] использован простой порог для обнаружения отставших узлов: всякий раз, когда задача выполнялась в 1.4X раза дольше среднего времени выполнения задач на данном этапе работ, ее отмечали как медленную.

D-Stream обрабатывают отставания, используя упреждающее исполнение команд, в то время как предыдущие системы потоковой обработки не поддерживали их. Этот тип спекулятивных (*speculation*) вычислений[9] трудно применить в режиме последовательной обработки операций (*record-at-a-time*), но просто с детерминированными задачами при разложении вычислений на короткие задачи [54].

Это, как уже отмечалось, повышает эффективность в сравнении с традиционными схемами репликации и резервного копирования, при допустимых задержках (*tolerates latency*) [40, 53].

Реализация принципа дискретизации в модуле *SparkStreaming* дало возможность обрабатывать более 60 млн записей в секунду на 100 узлах со сквозной субсекундной[10] задержкой (*sub-second end-to-end latencies*). Пропускная способность узлов *SparkStreaming* сравнима с коммерческими потоковыми базами данных, в то же время модуль предлагает линейную масштабируемость до 100 узлов и работает в два–пять раз быстрее, чем *Storm* и *S4* системы, а также в отличие от них предлагает восстановление после сбоев за доли секунды [40, 54].

Поскольку *D-Streams* использует ту же модель обработки и структуры данных *RDDs*, как и в пакетных заданиях, с присущей им интерактивностью, то это позволяет пользователям запускать нерегламентированные запросы (*ad-hoc queries*) на потоках или смеси потоковых и исторических данных. На практике это означает, что пользователи получают единый высокоуровневый *API* для объединения ранее разрозненных вычислений [40, 54].

Scala – язык, на котором реализован *Spark* и использует его в качестве среды разработки приложений для обработки данных. В отличие от *Hadoop*, *Spark* и *Scala* образуют тесную интеграцию, при которой *Scala* может легко манипулировать распределенными наборами данных как локальными коллективными объектами. *Spark* предоставляет интерфейсы программирования приложений для языков *Java*, *Scala*, *Python* и *R*, предлагает более 80 операторов высокого уровня, которые делают его легко доступным для построения параллельных приложений. Использовать их возможно в интерактивном режиме из оболочек *Scala*, *Python* и *R*.

Scala [55] – это достаточно новый, но уже успевший стать популярным, мультипарадигмальный язык в том смысле, что он гладко и удобно поддерживает языковые функции, характерные для императивных, функциональных и объектно-ориентированных языков.

На титульной странице сайта, на фоне Швейцарских Альп невдалеке от Женевского озера, где расположена Федеральная политехническая школа (*EPFL*), которая в 2003 г. благодаря М. Одерски (*Martin Odersky*) и его исследовательской группе дала жизнь *Scala*, гордо реют слова – главная сущность разработки, квинтэссенция *Scala*: «Объектно-ориентированное сливается с функциональным. Это есть лучшее из обоих миров. Построить элегантные иерархии классов для максимального повторного использования кода и расширяемости, осуществлять их поведение, используя функции высшего порядка или что-нибудь между ними».

В *Scala* используется чистая объектно-ориентированная модель. Ее единообразие состоит в том, что любое значение является объектом, а любая операция – вызовом метода. Это также функциональный язык, так как в нем каждая функция есть значение. Функциональные языки[11] обеспечивают лаконичный синтаксис для определения анонимных функций, работают с функциями высшего порядка, позволяют функциям быть вложенными, и поддерживают кар-

[9] Техника организации вычислительного процесса в параллельных процессорах на основе опережающего распределения команд по конвейерам.

[10] Временной интервал менее 1с.

[11] Программирование в функциональном стиле опирается на вычисление выражений, формируемых посредством комбинирования функций, а не на выполнение команд, т.е. вместо перечисления последовательности действий, необходимых для получения результата, описывается, что хотят получить. Функциональные языки используют высокоуровневую программную модель, вследствие чего программы становятся проще в разработке и поддержке.

рирование (*currying*[12]). Фактически, *Scala* совмещает лучшие черты обоих типов языков программирования.

Scala – аббревиатура от *Scalable Language* – расширяемый язык. Масштабируемость его – результат тесной интеграции[13] упомянутых парадигм программирования: объектно-ориентированной и функциональной концепции языков. *Scala* обеспечивает уникальное сочетание языковых механизмов, которые позволяют легко добавлять новые языковые конструкции в виде разнообразных библиотек и тем самым благоприятствует простому и быстрому созданию масштабируемого программного обеспечения [55–57]. Это позволяет легко выражать самостоятельные компоненты, использующие библиотеки *Scala*, не пользуясь специальными языковыми конструкциями.

Scala известен как краткий, читаемый язык программирования. Будучи достаточно выразительным языком, он предоставляет мощный инструментарий для создания простых и изящных программ [58, 59]. Вместе с тем, многие популярные библиотеки позволяют пользователям сделать код еще более простым и читаемым благодаря использованию *DSL*[14]. В мире программного обеспечения под *DSLs* подразумевается набор исполняемых выражений на языке, специфичном для данной области, удобочитаемые, понятные и, как правило, используемые экспертами, пребывающими в этой предметной области долгое время [60].

DSLs используют инструментарий определенной библиотеки. Эти выражения обычно касаются области применения библиотеки, а значит, позволяют сделать код простым и понятным для экспертов, знакомых с этой областью. Внутренние *DSLs*, написанные в *Scala*, стали мощным инструментом в руках *Scala*-программистов. В оригинальной конструкции *Scala* было уделено большое внимание тому, чтобы синтаксис позволил программистам создавать естественные *DSLs* [61, 62].

Scala-программы во многом похожи на *Java*-программы, и могут бесшовно взаимодействовать с *Java*-кодом, выполняться непосредственно на *Java Virtual Machine* (*JVM*). *Scala* использует огромный каталог существующих *Java*-библиотек наряду с *Java*-программами. Это позволяет ей выполнять почти все, что работает на виртуальных *Java*-машинах [45, 55].

Запускается *Spark*-приложение как массив независимых процессов на кластере (рис. 10), координацией которых занимается объект *SparkContext* в программе пользователя (*driver-program*). В режиме кластера, *SparkContext* может работать в паре с одним из нескольких менеджеров кластера (*Cluster Manager*), который выделяет ресурсы для приложений [63, 64]. *Spark* может работать как сам по себе, так и посредством нескольких существующих менеджеров кластера. В настоящее время система поддерживает три менеджера кластера: *Standalone* – автономный или собственный менеджер кластера, входящий в *Spark*; *Apache Mesos* – основной менеджер кластера, который может запускать *Hadoop MapReduce*-задачи, и *Hadoop YARN* – менеджер ресурсов в *Hadoop* [63].

Mesos обеспечивает эффективную платформу распределения ресурсов и изоляции распределенных приложений. Другой вариант использования – запуск *Spark*-приложений на *YARN*. Оба подхода позволяют *Spark* сосуществовать с *Hadoop* в общем пуле узлов (см. рис. 7). Единожды присоединившись (см. рис. 10) с помощью *driver-program*, *Spark* получает исполнителей (*executors*)[15] на *Worker Node*$_s$ в кластере. Они обрабатывают вычисления и сохраняют данные для приложения. *SparkContext* отсылает процессы (*task*$_s$), выполняющие назначенные задачи, к исполнителям для обработки.

[12] *Currying* (каррирование) – преобразование функции от многих аргументов в функцию, берущую свои аргументы по одному.

[13] В статически типизированных языках, к которым относится *Scala*, эти парадигмы до сих пор были почти полностью разделены.

[14] *DSL* (*Domain Specific Languages*) – язык, специфичный для предметной области или предметно-ориентированный язык, имитирующий термины, идиомы и выражения, используемые среди специалистов в целевой области.

[15] *Executor* – процесс, порожденный приложением (*driver-program*), который в свою очередь запускает *task*$_s$ и хранит данные в памяти или на диске. Каждое приложение имеет собственных исполнителей.

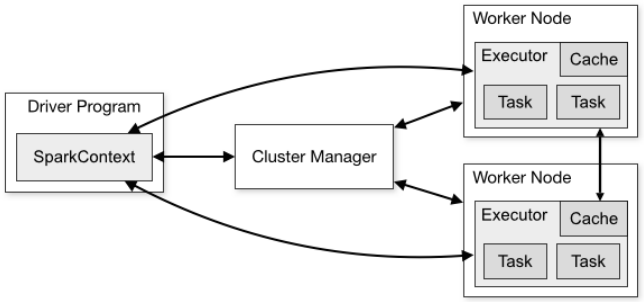

Рис. 10. Запуск *Spark*-приложения на кластере

Таким образом, *Apache Spark* [65] – производительное вычислительное ядро – новое, еще более совершенное и быстрое средство обработки больших массивов данных как в *Hadoop*, так и с другими кластерными вычислительными системами [66]. Решение *Spark* рассчитано не только на профессиональных разработчиков – оно может стать инструментом для специалистов по данным (*data scientist*) с предоставлением современного языка *Scala*, реализованного для платформ *Java* и *.Net*.

Apache Mahout, начиная с версии 0.10.0 (апр. 2015 г.), содержит интерактивную математическую среду (*Mahout Samsara*) машинного обучения (*ML*), включая расширенную версию *Scala*. Библиотека машинного обучения *ML MLLib*, поддерживает универсальную масштабируемую линейную алгебру и включает в себя многие современные алгоритмы [46, 67, 68] машинного обучения (табл.). Начиная с этого выпуска, *Mahout* должен работать на *Spark* или новом аналитическом инструменте для работы с данными под названием *H2O.* [69, 70]. В *Hadoop MapReduce* версии алгоритмов *Mahout* еще сохраняются, но новых разработок на нем не проводится. Для *Mahout H2O* представляет собой распределенную масштабируемую систему машинного обучения. Сочетание *H2O* и *Spark* позволяет пользователям комбинировать быстрые масштабируемые алгоритмы машинного обучения *H2O* с возможностями *Spark*. *Sparkling Water* (*Spark* + *H2O*) пользователи могут управлять вычислениями из *Scala/R/Python* и применить пользовательский интерфейс *H2O Flow*[16], обеспечивая идеальную платформу для разработчиков приложений в области статистических вычислений. *H2O* разрабатывается компанией *H2O.ai* и использует наиболее популярные *Open Source* – продукты, такие как *Apache Hadoop* и *Spark* [70].

Помимо коллаборативной фильтрации – одной из самых популярных и простых в применении возможностей *Mahout* для реализации рекомендательных систем – в нем есть несколько алгоритмов классификации, большинство из которых (за исключением вероятностного наискорейшего спуска) написаны для исполнения на *Hadoop*. Как и в случае с классификацией, *Mahout* предлагает многочисленные алгоритмы кластеризации, каждый из которых имеет свои особенности. Например, алгоритм *K-Means* красиво масштабирует, но требует указать количество кластеров, которые желательно использовать, тогда как для алгоритма кластеризации *Дирихле* требуется выбрать модель распределения и указать количество кластеров. Кластеризация имеет немало общего с классификацией, и иногда их можно использовать совместно, так что кластеры становятся частью классификации. Более того, значительная часть работы по подготовке данных для классификации – та же, что и для кластеризации – например, преобразование исходных данных в последовательность файлов и затем в разреженные векторы [68]. Большинство алгоритм сокращения размерности (*Dimensionality Reduction*), например, *Singular Value Decomposition – SVD* и другие на основе *Scala* доступны через ядро библиотеки *Mahout Math-Scala* для всех платформ. Алгоритм *Collocations* реализован с использованием *MapReduce* [67].

Математическая среда *Mahout Samsara* или *Scala & Spark Bindings* [71, 72], введенная *Apache*, была вызвана необходимостью создания семантически дружественных условий для линейной алгебры. Работа с векторными, матричными и тензорными структурами данных, как с одним типом данных, предполагает наличие существенных свойств, необходимых

[16] *H2O Flow* представляет собой ноутбук-стиль с открытым исходным кодом пользовательского интерфейса для *H2O*. Это веб-интерактивная среда, которая позволяет сочетать выполнение кода, текста, математики, графики и богатые средства массовой информации в одном документе, похожем на *IPython* ноутбуков.

Таблица. Алгоритмы *Mahout*

Алгоритм	Реализация			Краткое описание и пример использования
	Одна машина	*Map Reduce*	*Spark*	
Коллаборативная фильтрация				
User/ Item-Based Collaborative Filtering – алгоритмы рекомендательных систем	x	– / x	x	Прогнозы, исходя из накопленной информации об интересах и вкусах пользователей. Рекомендации по посещениям (сайты, документы, новости) и товарам; системы на основе рейтингов; выявление трендов и пр.
MatrixFactorization with ALS – факторизация матриц с чередованием наименьших квадратов	x	x	x	Матричные модели, во много раз превосходящие классические методы выработки рекомендаций, позволяют включение дополнительной информации, такой как временные эффекты и уровень доверия
MatrixFactorization with ALS on Implicit Feedback – то же, но с учетом обратной связи	x	x	x	Матричные модели, построенные с возможностью учитывать неявную обратную связь
Классификация				
Stochastic Gradient Descent, SGD – логистическая регрессия, решение методом вероятностного наискорейшего спуска	x		x	Быстрый, простой и последовательный классификатор, приспособленный к оперативному обучению в сложных условиях, используемый для рекомендации рекламы пользователям, классификации текста по категориям
Naive Bayes / Complementary Naive Bayes –наивный Байесовский классификатор		x	x	Наивный Байесовский классификатор – двухкомпонентный процесс, который находит признаки, связанные с определенным документом и категорией, и затем использует эту информацию для прогнозирования категории новых, еще не знакомых данных
Random Forest – случайные леса		x	x	Случайные леса – объединения деревьев решений; сочетают в себе много деревьев решений с тем, чтобы уменьшить риск переобучения
Hidden Markov Models – скрытая модель Маркова	x	x		Последовательные и параллельные реализации классического алгоритма классификации, предназначенные для моделирования реальных процессов, когда основной исходный процесс неизвестен
Кластеризация				
Canopy Clustering – навесная кластеризация (устаревший)	x	x		Быстрый алгоритм кластеризации, часто используемый в качестве отправной точки для создания других алгоритмов кластеризации
k-Means Clustering, Fuzzy k-Means – метод *k*-средних кластеризации и нечетких *k*-средних	x	x	x	Группирует элементы в *k*-кластеры, основываясь на расстоянии от этих элементов к центроиду, или центру тяжести предыдущей итерации
Dirichlet Filtering – кластеризация Дирихле	x	x		Подход к кластеризации на основе модели, когда членство определяется по тому, вписываются ли данные в базовую модель. Полезен при наличии параллелизма или иерархии в данных
Spectral Clustering – спектральная кластеризация		x		Использование графического метода определения членства в группе. Как и все алгоритмы группирования, полезен при изучении новых крупных наборов данных
Сокращение размерности				
Singular Value Decomposition SVD, – разложение на сингулярные значения	x	x	x	Снижение уровня шума в больших матрицах вследствие сокращения их размерности. Применяется для автоматического выбора характеристик перед выполнением кластеризации, рекомендаций и классификации
Разное				
Collocations – коллокации		x		Нахождение устойчивых или фразеологических словосочетаний. Поиск в тексте статистически интересных фраз

для быстрого прототипирования алгебраически определенных математических задач. *Mahout Samsara* должна помочь создать собственную математику, обеспечивая некоторые готовые решения реализации алгоритма, и в случае их недостаточности позволить строить свои модели, проверять свои возможности и применять конкретные правила и ансамбли. Математические пакеты должны взять все это на себя, а пользователю предоставить возможность только сформулировать задачу.

Другое востребованное свойство – удобство той же среды в качестве языка программирования. *Apache Mahout*, так же как и *Apache Spark*,

смещают все свои алгоритмы с *Java* на язык *Scala*, т.е. движутся в одном направлении. Различия в результате ошеломляющие, так как *Scala* обеспечивает более краткий способ записи математических программ. Старый код, занимавший тысячи строк, сокращается менее чем до 100. Дискретность чрезвычайно улучшилась. Эффективность высока [73]. Математическая среда *Samsara* построена по образу базового пакета в *R*[17] [74]. Возможно, математику, написанную в *R*-версии легче понять и поддерживать, чем те же выражения, написанные в других основных процедурных или функциональных средах. Это способствует удобству пользования программистами, хорошо знакомыми с основными матричными примитивами *R* [72]. *Samsara*-библиотека линейной алгебры для *Mahout* написана на языке *Scala* и имеет хороший *R*- или *Matlab*-подобный синтаксис для базовых операций линейной алгебры, операторы используют подобные обозначения: % * %, *colSums*, *nrow* и др. Более того, эти операции могут быть распределены [75]. Уравнение, решение которого ранее занимало порядка десяти–двадцати строк текста, решается одной командой *solve*: *val w = solve* (*drmXtX*, *drmXty*) [76].

Mahout имеет довольно полнофункциональный *Scala API* и *DSL* линейной алгебры. Включение *Scala DSL* (*Domain Specific Language*) – очередной шаг в укреплении и расширении математической библиотеки линейной алгебры (*Math Library*) *Mahout*, которая представляет собой не только ценность для проектов *Mahout*, включая обнаружение аномалий, выработку рекомендаций, кластеризацию и прочее, но может быть полезным инструментом для решений вне его [77].

[17] *R* – язык программирования для статистической обработки данных и работы с графикой, свободная программная среда вычислений с открытым исходным кодом. *R* поддерживает широкий спектр статистических и численных методов, обладает хорошей расширяемостью с помощью пакетов. Совмещение базового пакета с численными методами называют языком формул – именно поэтому в *R* проще обрабатывать статистические задачи. Еще одна его особенность – это графические возможности создания качественной графики, которая может включать в себя математические символы [74, 78].

Тесная связь, сращивание *Scala* и *Spark* для *Mahout* (*Samsara*) представляет собой комбинацию *DSL Scala* и распределенного алгебраического оптимизатора (рис. 11) выражений, подобных этому:

$$G = BB^T - C - C^T + s_q s_q^T \xi^T \xi, \qquad (1)$$

которое характеризуется сосредоточенными (*in-core*) и распределенными вычислениями. Выражение (1) в *Mahout-Samsara* должно быть записано пользователем так:

$$val\ g = bt.t\ \%\ *\ \%\ bt - c - c.t + \\ + (s\_q\ cross\ s\_q)\ *\ (xi\ dot\ xi). \qquad (2)$$

Samsara самостоятельно, не требуя контроля пользователя, оптимизирует выражение (2) и распределит задачи по узлам кластера. Теперь пользователь может не заботиться о логических и физических планах выполнения работы и не должен разбираться с кодом на логическом уровне, подобно тому, как это будет осуществляться с *R*.

Другая идея состоит в представлении распределенного логического выражения для обработки на кластере. По мере добавления узлы подключаются к обработке заданного ранее выражения, т.е. *Samsara* самостоятельно учитывает размер кластера при вычислениях. Выражение, записанное ранее, будет обрабатываться и на новых узлах.

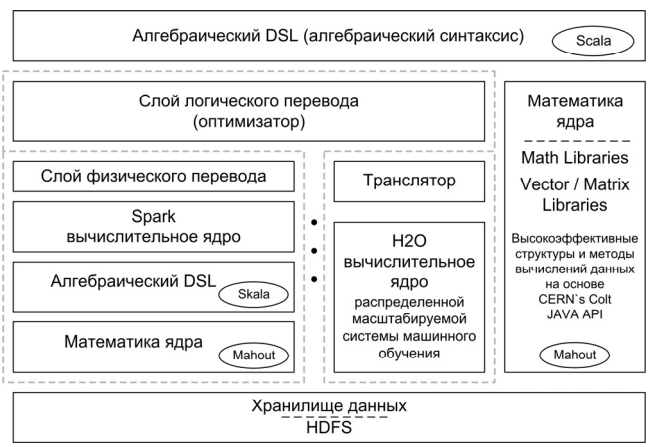

Рис. 11. Математическая среда *Mahout Samsara*

Линейная алгебра работает со скалярами, векторами и матрицами (включая многочисленные специализированные типы, как, например,

многомерные матрицы[18]) и распределенными строками матриц (*distributed row matrix*, *DRMs*) [71, 72].

DRM – новая абстракция, введенная в *Apache Mahout* для удобства представления и обработки матриц. Суть абстракции заключается в том, что матрица записывается в виде отдельных индексированных строк или блоков строк, сохраняемых и обрабатываемых на разных узлах кластера. Чаще всего матрицы разбивают именно на блоки, т.е. выделяют несколько строк одновременно, так как это позволяет обрабатывать больше элементов. В таком представлении – это распределенная коллекция пар. Хороший пример представления матрицы в виде *DRMs* можно увидеть в [76]. Противоположностью *DRMs* служат нераспределенные (*in-core*), обычные матрицы или векторы, которые хранятся в памяти виртуальной *Java*-машины и не распространяются по узлам.

Одним из основных элементов *Mahout Samsara* является алгебраический *DSL Scala*, также называемый *синтаксическим сахаром* [81]. Этот *DSL Scala* предоставляет пользователю тот самый удобный алгебраический синтаксис.

Существенная часть *Mahout Samsara* – оптимизатор выражений (см. рис. 11). Он осматривает представленное пользователем выражение в целом и пытается найти способы упростить его, используя известные физические операторы. Например, можно понять, что выражение, подобное этому $dlog(X * X + 1).t$ %*% $dlog(X * X + 1)$, по сути (в упрощенном псевдокоде физических операторов) эквивалентно приведенному ниже:

$$self\_square(X.map(x => log(x * x + 1))).$$

Оптимизатор автоматически сократит задачу до оптимального уровня. Таким образом, значительно упрощается работа со сложными задачами, а десятки и сотни строк кода заменяются несколькими математическими выражениями [81].

Mahout-Samsara поставляется с интерактивной оболочкой, которая запускает распределенные операции на кластере *Spark*. Это упрощает формулирование задач и позволяет пользователям настраивать алгоритмы с совершенно новой степенью свободы.

Для вычислений *Scala & Spark Bindings* использует вычислительное ядро. В роли этого ядра на данный момент может выступать *Spark* или *H2O*. Последний представляет собой распределенную масштабируемую систему машинного обучения. Его внутренняя архитектура включает в себя распределенный математический процессор (вычислительное ядро *H2O*) и, поверх алгоритмов, отдельный слой пользовательского интерфейса. Интеграция в *Mahout* требует только вычислительного ядра. Планируется также использовать вычислительные ядра других сред, например *Apache Flink*. Вместе с ядром *Spark* используется его внутренний алгебраический *DSL*.

Некоторые задачи не нуждаются в предварительной оптимизации исходных выражений и могут быть решены напрямую через математику ядра *Mahout*. *Samsara* самостоятельно определит ресурсы, требуемые для решения задачи.

Фактически, с помощью *Scala & Spark Bindings* пользователь может реализовать алгоритмы *Spark*, но при этом значительно упрощается работа с векторами и матрицами, что есть важным преимуществом для алгоритмов машинного обучения. Так, например, большинство алгоритмов коллаборативной фильтрации основаны на операциях с матрицами и их элементами.

Отметим, что *Samsara* ускоряет не выполнение алгоритма, а реализацию любого алгоритма пользователем. Таким образом, основной целью создания *Samsara* является упрощение работы для пользователя.

В версии 0.11.2 *Apache Mahout* (март 2016) *Mahout Algorithms* включает в себя новые высокоскоростные (10x) алгоритмы [34], реализованные на *Mahout-Samsara*. Работают они на *Spark* 1.3+ и некоторые на *H2O*.

Заключение. В завершение работы отметим, что приведенные в ней программные вычислительные среды для обработки больших данных, несомненно, отражают тенденцию стрем-

[18] В отличие от многих других вычислительных сред, математика *Mahout* изначально была ориентирована на плотные и разреженные структуры данных [72, 79, 80].

ления информационных технологий к новому поколению технологий, формирующих третью платформу [82]. Совокупность *Open Source* разработок привела к резкому увеличению количества специализированных решений, продиктованных потребностями индустрии, и развитию рынков решений под отдельные отрасли. Это будут самые ценные, трансформирующие отрасль, решения.

Если *Fast Data* – Быстрые Данные – новые технологии преобразования и обработки информации значительно быстрее, чем когда-либо прежде, то *Big Analytics* – Большая Аналитика – во-первых, это тот уровень, где качественные отличия между традиционными и большими данными становятся более очевидными: данные разных форматов, в том числе неструктурированные, вызвавшие интенсивное развитие нереляционных БД и др. Во-вторых, большая аналитика – это превращение информации в знания с использованием комбинации существующих и новых подходов [7]. *Apache Mahout* привносит в большую аналитику автоматизированное обучение, находя скрытые тенденции в противоположность неосмысленным или непродуманным идеям. Действительно, в интеллектуальных системах управления информацией, скорее всего, не будут полагаться на пользователей, придумывающих «умные вопросы к компьютеру», а будут автоматически применять решение об отправке, например автоматического уведомления пользователю или системе о возможности или риске угрозы, в случае если новые наблюдения показывают достаточный интерес к чему-либо, чтобы оправдать подобные реакции.

Вместе с тем, мощные, но не сфокусированные инструменты большой аналитики, т.е. не направленные на извлечение определенного типа результата, не будут достаточными, чтобы извлечь все преимущества больших данных. Проникновение в суть явлений должно быть связано с конкретными целями, чтобы иметь высокие уровни воздействия.

Автор выражает благодарность за помощь в подготовке фрагментов материала настоящей работы В. Духновскому.

34. *What is* Apache Mahout? – http://mahout.apache.org/
35. *Воронцов К.В.* Коллаборативная фильтрация. – http://www.machinelearning.ru/wiki/images/9/95/Voron-ML-CF. pdf, 6 нояб. 2013.
36. *Алгоритм* коллаборативной фильтрации. – http://habrahabr.ru/post/80955/, 16 янв. 2010.
37. *Apache Mahout*. – http://hortonworks.com/hadoop/mahout/, March 2010.
38. *Черняк Л.* Альтернативы MapReduce для реального времени // Открытые системы. – 2014 – № 5. – http://www.osp.ru/os/2014/05/13041818/
39. *Сериализация* в Java. – https://habrahabr.ru/post/60317/, 24 мая 2009.
40. *Resilient* Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing / M. Zaharia, M. Chowdhury, T. Das et al. – NSDI 2012, апр. 2012. – https://people.csail.mit. edu/matei/publications/
41. *Spark* Programming Guide. Spark 1.5.2. – http://spark.apache.org/docs/latest/programming-guide.html
42. *Intro* to Apache Spark. – http://stanford.edu/~rezab/sparkclass/slides/itas_workshop.pdf, (15. 08. 2014).
43. *Dinsmore T. W.* Apache Spark for Big Analytics (Updated for Spark Summit and Release 1.0.1) // The Big Analytics Blog. – http://thomaswdinsmore.com/2014/01/02/apache-spark-for-big-analytics/, (01. 02. 2014).
44. *Overview* – Spark 1.5.2 Documentacion – Apache. – http://spark.apache.org/docs/latest/
45. *Джонс М. Тим.* Spark, альтернатива для быстрого анализа данных. – http://www.ibm. com/developerworks/ru/library/os-spark/, 12.07.2012.
46. *Machine* Learning Library (MLlib) Guide. – http://spark.apache.org/docs/latest/mllib-guide.html
47. *GraphX* Programming Guide. – http://spark.apache.org/docs/latest/graphx-programming-guide.html
48. *Spark SQL* and DataFrames – Spark 1.5.2 Documentation. – http://spark.apache.org/docs/latest/sql-programming-guide.html
49. *Apache* Kafka. – http://kafka.apache.org/
50. *Amazon* Kinesis. – https://aws.amazon.com/ru/kinesis/
51. *Spark* Streaming Programming Guide. – http://spark.apache.org/docs/latest/streaming-programming-guide. html
52. *Spark FAQ*. – http://spark.apache.org/faq.html
53. *Discretized* Streams: An Efficient and Fault-Tolerant Model for Stream Processing on Large Clusters / Matei Zaharia, Tathagata Das, Haoyuan Li et al. – Univ. of California, Berkeley. – http://people.csail.mit.edu/matei/papers/2012/ hotcloud_spark_streaming. pdf
54. *Discretized* Streams: Fault-Tolerant Streaming Computation at Scale / Matei Zaharia, Tathagata Das, Haoyuan Li et al. – Univ. of California, Berkeley http://people.csail.mit.edu/matei/papers/2013/sosp_spark_streaming.pdf
55. *Scala.* – http://scala-lang.org/
56. *Contributed* Libraries and Tools. – http://www.scala-lang.org/old/node/1209.html#libraries

57. *Awesome* Scala. – https://github.com/lauris/awesome-scala
58. *The Scala* Program. Lang. – http://www.scala-lang.org/old/node/25.html
59. *Creating* Domain Specific Languages with Scala – Part 1. – http://blog.scalac.io/2015/05/07/encog-dsl.html
60. *Hunger M.* Domain-Specific Lang. – http://programmer.97 things. oreilly.com/wiki/index.php/Domain-Specific_ Languages (23.12.15).
61. *DSLs* – A powerful Scala feature. – http://www.scala-lang.org/old/node/1403
62. *The Scala* Program. Lang.. – http://www.scala-lang.org/old/ node/25.html
63. *Spark* 1.5.2 Cluster Mode Overview. – http://spark.apache.org/docs/latest/cluster-overview.html
64. *По материалам*: Cloudera. Оптимизация заданий Apache Spark. Ч. 1. – http://datareview.info/article/optimizatsiya-zadaniy-apache-spark-chast-1/, 20.05. 2015.
65. *Apache Spark.* – http://spark.apache.org/
66. *18 essential* Hadoop tools. – http://www.kdnuggets.com/2014/08/18-essential-hadoop-tools.html. Авг. 2014.
67. *Mahout* 0.10.1 Features by Engine. – https://mahout.apache.org/users/basics/algorithms.html
68. *Ингерсолл Г.* Apache Mahout: масштабируемое машинное обучение для всех. – https://www.ibm.com/developerworks/ru/library/j-mahout-scaling/
69. *What* is Apache Mahout? Release Notes. – http://mahout.apache.org/
70. *Sparkling Water*. – http://www.h2o.ai/product/sparkling-water/
71. *Scala & Spark* Bindings. – http://mahout.apache.org/users/sparkbindings/home.html
72. *Lyubimov D.* Mahout Scala Bindings and Mahout Spark Bindings for Linear Algebra Subroutines. – http://mahout.apache.org/users/sparkbindings/ ScalaSparkBindings.pdf
73. *Dunning Ted.* Why Apache Mahout is shifting its all algorithms from Java to Scala, i.e. are Apache Spark and Apache Mahout are moving in one direction? – http://www.quora.com/Why-Apache-Mahout-is-shifting-its-all-algorithms-from-Java-to-Scala-i-e-are-Apache-Spark-and-Apache-Mahout-are-moving-in-one-direction, 18 апр. 2015.
74. *A Free* Software Project. – https://cran.r-project.org/doc/html/interface98-paper/paper_2.html
75. *Ferrel Pat*. Mahout on Spark: What's New in Recommenders. – https://www.mapr.com/blog/ mahout-spark-what%E2%80%99s-new-recommenders, 12 авг. 2014.
76. *Grigorev Alexey*. Apache Mahout Samsara: The Quick Start. – http://www.itshared.org/2015/04/apache-mahout-samsara-quick-start.html, April 2015.
77. *Friedman Ellen.* Advances in Apache Mahout: Highlights for the 0.9 Release. – https://www. mapr.com/blog/advances-apache-mahout-highlights-09-release#.Vebs-rWTWT4, 19 Febr. 2014.
78. *Делзелл К.* Необходимо ли вам изучать язык *R*?. – http://www.ibm.com/developerworks/ru/library/bd-learnr/, 24.10.2014.
79. *Заботнев М.С.* Методы представления информации в разреженных гиперкубах данных. – http://www.olap.ru/basic/theory.asp
80. *Подгорский С.* Написание МКЭ расчетчика в менее чем 180 строк кода. – https:// habrahabr.ru/post/271723/, 1 дек. 2015.
81. *Lyubimov D.* Mahout 0.10.x is coming. – http://www.weatheringthroughtechdays. com/, апр. 2015.
82. *Gens F*. The 3rd Platform: Enabling Digital Transformation. IDC. – http://www.idc.com, Nov. 2013.

Поступила 13.07.2016
Тел. для справок: +38 044 526-4159 (Киев)
E-mail: *aleksei@irtc.org.ua*

UDC 004.7:004.75:004.9:004.738.5

A.A. Oursatyev

Some Frameworks for Big Data Analytics and Machine Learning

The machine Learning (Machine Learning, ML) and distributed processing of the large data collections on Apache Mahout with the automatic search ability for relevant laws are considered. Its realization through the use of MapReduce paradigm and framework Spark is compared. The representation of data and mechanisms to restore their failures, the method of calculation and the ability to cache data in memory are considered. The latter is a key tool for fast interactive use. Spark is implemented on Scala. It combines the best features of functional and object-oriented programming languages, and uses it as an application of the environment development. It provides the application programming interface for the Java language, Scala, Python and R, invites more than 80 high-level operators that makes it easily accessible for the construction of a parallel applications.

Interactive mathematical environment Mahout Samsara ML includes an extended version of Scala. Mahout Samsara or the Scala & Spark Bindings are necessary for creation the semantically friendly conditions for epy linear algebra, and is built in the image of the base package in R. The linear algebra works with scalars, vectors, matrices and distribution lines of the matrices (distributed row matrix, DRMs). DRM is a new abstraction, introduced in Apache Mahout for the representation and processing matrices convenience. One of the main elements of Mahout Samsara is algebraic DSL Scala and expressions optimizer. ML Mllib, supports the scalable universal linear algebra and includes many modern algorithms.

●