УДК 004.7:004.75:004.9:004.738.5

А.А. Урсатьев

Некоторые программные среды аналитики больших данных

Изложены концептуальные вопросы построения сред обработки данных – кластерных систем на программной платформе *Hadoop*. Описана инфраструктура *HadoopMapReduce* для организации параллельных распределенных вычислений над данными и показано эволюционное преобразование платформы *Hadoop* применительно к интерактивным и потоковым динамическим нагрузкам.

Викладено концептуальні питання побудови середовищ обробки даних – кластерных систем на програмній платформі *Hadoop*. Описано інфраструктуру *HadoopMapReduce* для організації паралельних розподілених обчислень над даними і показано еволюційне перетворення платформи *Hadoop* стосовно інтерактивних і потокових динамічних навантажень.

Введение. Возрастающие объемы данных, насущная потребность извлечь из них необходимую информацию и новые знания вынуждают разработчиков аналитических систем уделять внимание как кардинальному совершенствованию традиционных технологий и средств их обеспечения, так и созданию новых, прогрессивных аппаратных и программных сред аналитики.

Стремительный рост данных означает, что общее их число в цифровой вселенной возрастет экспоненциально с 4,4 Зб в 2013 г. до 44 Зб (44 триллиона гигабайт или 44×1012 Гб) в 2020 г. при среднегодовом темпе роста 40 процентов [1]. Исследования [1–3] сходны в одном – объем данных в мире стремительно расширяется и будет продолжать расти в обозримом будущем в геометрической прогрессии.

Источниками данных, производимых мировым сообществом, могут быть современные научные исследования [2], производственная и социальная деятельность, бизнес и другие сферы вплоть до автоматических систем, например, радиочастотной идентификации (*RFID*) и Интернета вещей [1, 4]. Ожидается[1], что использование этих данных позволит создать прирост национальных и мировой экономик, существенно повысит эффективность функционирования и конкурентоспособность организаций частного и государственного секторов. Многочисленные пулы данных, которые хранятся и могут быть проанализированы – теперь являются составной частью отраслей и секторов мировой экономики [5].

Таким образом, возникает проблема использования потенциала данных, обусловленная не столько значительным их числом, сколько в неспособности переработать старыми методами новый объем больших данных. Под термином *Big Data* [5] подразумевают набор данных, объем которых выходит за пределы возможностей типовых программных средств баз данных (БД) для хранения и обработки. Это определение субъективно, так как не ставится вопрос, каковым должен быть набор данных, чтобы считать его большими данными (*Big Data*), т.е. объем данных не определяется в тера- или петабайтах. *IDC*[2] определяет технологии *Big Data* как новое поколение технологий и архитектур, предназначенных для экономичного извлечения значений из очень больших объемов самых разнообразных данных, обеспечивая высокую скорость захвата и анализа [6].

Программные среды аналитики больших данных

Предпринята попытка классификации больших наборов данных, которая позволила бы

[1] *McKinsey Global Institute* (*MGI*) – экономический мозговой центр международной консалтинговой *McKinsey & Company*.

[2] *IDC – International Data Corporation* – аналитическая фирма, специализируется на исследованиях рынка информационных технологий (*http*://*www.idc.com*/).

соотнести существующие технологии с ожидаемыми результатами их обработки [7]. Автор этой работы делит подходы к *Big Data* на три группы (рис. 1): *Fast Data* (Быстрые Данные), *Big Analytics* (Большая Аналитика) и *Deep Insight* (Глубокое Проникновение, т.е. способность глубоко проникать в суть).

Обработка для *Fast Data* не предполагает получения новых знаний, ее результаты соотносятся с априорными знаниями и позволяют судить о том, как протекают те или иные процессы, она позволяет лучше и детальнее увидеть происходящее, подтвердить или отвергнуть какие-то гипотезы. Задачи, решаемые средствами *Big Analytics* и заложенными в них соответствующими технологиями, служат для преобразования зафиксированной в данных информации в новое знание. На этом среднем уровне не предполагается наличие искусственного интеллекта при выборе решений или каких-либо автономных действий аналитической системы – она строится по принципу *обучения с учителем*. Иначе говоря, весь ее аналитический потенциал закладывается в нее в процессе обучения.

Высший уровень, *Deep Insight*, предполагает обучение без учителя (*unsupervised learning*) и использование современных методов аналитики, а также различные способы визуализации. На этом уровне возможно обнаружение знаний и закономерностей, априорно неизвестных [7, 8].

Рассмотрим отдельные системы обработки данных (см. рис. 1). Одна из них, широко известная, это программная инфраструктура с открытым исходным кодом *Hadoop* – де-факто стандарт технологий для работы с *Big Data*.

Hadoop широко используется в сфере финансов, средствах массовой информации, правительственных кругах, здравоохранении, информационных услугах, розничной торговле и ряде отраслей промышленности. Обеспечивает совместное распределенное хранение и параллельную обработку в кластере стандартных машин как структурированных и неструктурированных данных размерности терабайт, петабайт и выше, что значительно превышает возможности существующих технологий, основанных на традиционных СУБД и хранилищах данных [9 –19].

Кластер *Hadoop* хорошо масштабируется путем добавления узлов и может состоять из многих сотен и тысяч серверов. Распараллеливание данных и вычислений по серверам кластера позволяет увеличить в целом быстродействие обработки. Одно из его ценовых преимуществ – то, что он полагается на внутреннюю избыточную структуру данных – поддерживается несколько рабочих копий данных – и развертывается на стандартных серверах, а не дорогих специализированных системах хранения данных. Другое – то, что в *Hadoop* предусмотрено дублирование на случай выхода из строя узлов. Этим достигается высокая надежность и отказоустойчивость. В основу продукта положены два компонента:

- отказоустойчивая распределенная файловая система *Hadoop Distributed File System* (*HDFS*), обеспечивающая хранение файлов путем их распределения по узлам кластера *Hadoop*;

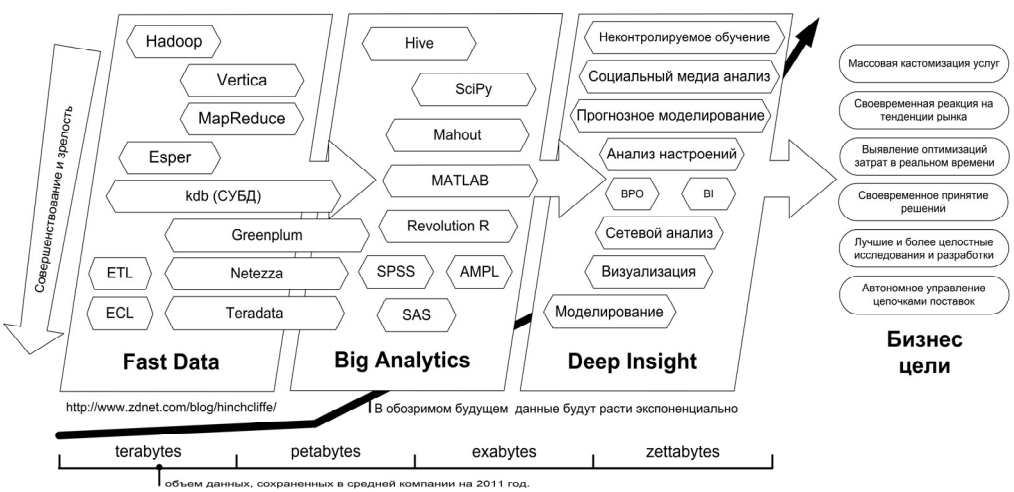


Рис. 1. Распространенные средства обработки *Big Data* и их развитие

• *Hadoop MapReduce* – программная платформа (*framework*)[3] – модель, определяющая структуру системы в рамках парадигмы *map*/*reduce*[4], согласно которой приложение разделяется на большое количество одинаковых заданий, выполняемых на узлах кластера и сводимых в конечный результат.

В *HDFS* вместо таблицы файловых дескрипторов и области данных, присущей обычным файловым системам, используется специальный сервер – сервер имен *NameNode* (метаданные), а собственно данные распределены по серверам данных *DataNode*. Данные разбиваются на блоки, каждый из которых попадает на отведенные ему места в пуле серверов. Для каждого файла сервер имен хранит его путь, список блоков и их реплик. Обычно в *Hadoop* хранится не менее трех копий. Если *HDFS* обнаруживает исчезновение блока, она автоматически восстанавливает резервную копию, снова доводя их число до прежнего значения.

Модель программирования *MapReduce* ориентирована на параллельную масштабируемую обработку больших массивов данных. *Hadoop* адресуется без индексации сразу ко всем данным, что дает преимущества при работе с гигантскими объемами, распределенными по сотням и тысячам серверов. Несмотря на относительно невысокую скорость выполнения операций на узлах, общая производительность параллельных систем высока. Объем одновременно анализируемых данных в реляционных СУБД, напротив, ограничивается методами индексации, так как РСУБД создавались в расчете на ускорение и упрощение обработки запросов.

Классическая конфигурация кластера *Hadoop* состоит из одного сервера имен *NameNode*, одного мастера *MapReduce* (*JobTracker*) и набора рабочих машин, на каждой из которых одновременно запущен сервер данных *DataNode* и вычислительные процессы – *TaskTracker* (*Workers*). Информация о том, на каких машинах (рабочие узлы – *worker node*) расположены блоки данных, позволяет запустить на них же вычислительные процессы и выполнить предварительную обработку данных локально, т.е. без передачи их по сети (рис. 2). Именно эта идея положена в основу парадигмы *MapReduce*, реализованной в *Hadoop*.

Hadoop предоставляет инфраструктуру ПО для выполнения заданий *MapReduce* в виде серий задач *map* и *reduce*. Задачи *map* вызывают функции *map* для обработки наборов данных, задачи *reduce* – направлены на обработку промежуточных результатов, сгенерированных функциями *map*, и формирование окончательных выходных данных.

Каждая, инициируемая *MapReduce*, задача – это две фазы:

• *map* – выполняется параллельно и локально над каждым блоком данных. Вместо того, чтобы доставлять терабайты данных к программе, определенная пользователем программа копируется на рабочие узлы – серверы с данными и выполняет их обработку;

• *reduce* – дополняет *map* агрегирующими операциями над обработанными данными, формируя результат.

Задачи *map* и *reduce* выполняются изолированно одна от другой, что обеспечивает параллельность и отказоустойчивость вычислений.

В инфраструктуре *MapReduce* выполнение задания управляется двумя видами процессов (см. рис. 2):

• один главный процесс *JobTracker* координирует выполнение заданий в кластере и назначает задачи *map* и *reduce* для выполнения процессами *TaskTrackers*;

• подчиненные процессы *TaskTracker*, выполняющие назначенные задачи, периодически передают результаты в *JobTracker*.

Помимо планирования заданий и координации всех задач, выполняемых в кластере (включая инструкции процессам *TaskTracker* по запуску задач *map* и *reduce*), главный процесс *JobTracker* осуществляет мониторинг выполнения задач в кластере, их повторный запуск

[3] Цель фреймворка – предоставить программисту удобную среду, каркас, для проекта с большим и хорошо расширяемым функционалом.

[4] *MapReduce* – модель программирования, объединяющая формирование наборов из больших данных на узлах кластера с их обработкой, служит для организации параллельных распределенных вычислений.

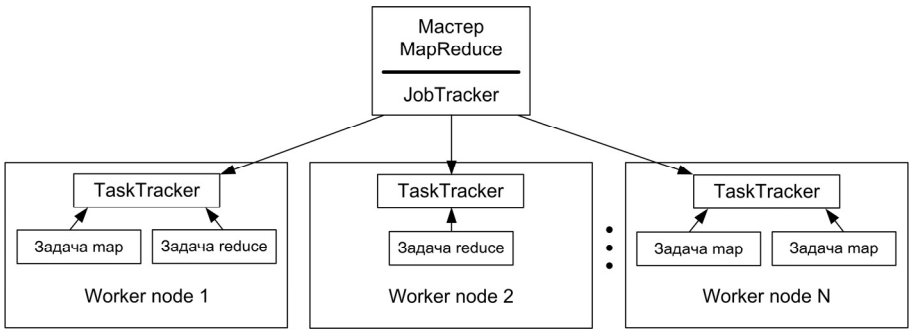


Рис. 2. *MapReduce*, реализованный в *Hadoop*

после сбоев, упреждающее выполнение медленных задач и др.

На процессе *JobTracker* также лежит обязанность управления вычислительными ресурсами в кластере (включая обработку списков работоспособных узлов, доступных и занятых слотов *map* и *reduce*), выделение доступных слотов для соответствующих заданий и задач согласно выбранной политике планирования [10].

Стандартный *MapReduce* спроектирован так, что все результаты – как конечные, так и промежуточные – записываются на диск. В итоге время считывания и записи на диск, увеличенное в *n* число раз предусмотренным решением задачи, зачастую превышает время собственно вычислений. В настоящее время вместо *MapReduce* применяют новое производительное вычислительное ядро – *Spark*. Он использует ту же идею локальности данных, однако выносит большинство вычислений в память вместо диска и имеет более эффективную и удобную модель программного ядра [11]. Компания *Cloudera* [12] предлагает коммерческую поддержку *Spark*, а *Hortonworks* [13] решила сделать упор на альтернативное вычислительное ядро – *Apache Tez* [14] и *Apache™ Storm* [15].

Storm, как и *Spark* – альтернатива пакетному *MapReduce*, – вносят в экосистему *Hadoop* эффективные средства оперативной обработки. Однако предназначение проектов разное – *Storm* представляет собой классическую распределенную систему обработки сложных событий (*Complex Event Processing*, *CEP*), выполненную в экосистеме *Hadoop*, а *Spark* – это интерактивно-аналитическая система, предназначенная для задач, использующих многопроходные обработки (машинное обучение, граф-анализ, глубинный анализ данных).

В *Storm* по аналогии с потоком в системе водоснабжения (*plumbing*) используют метафору с трубами (*spout*) и задвижками (*bolt*) для создания рабочих процессов (*workflows*). Топология в *Storm* – это программно реализованная схема преобразования потока, непрерывного и потенциально бесконечного набора кортежей. Для создания топологии – некоторой конструкции обработки входящего потока – разработчик должен реализовать методы, заданные в интерфейсах *труб* и *задвижек*, отражающие специфические для конкретной задачи атомарные преобразования, формирующие целостный непрерывный процесс преобразования входящего потока сырых данных. Задания запускаются и по мере выполнения завершаются, а *топология*, будучи единожды созданной, функционирует постоянно, пока ее не остановит системный администратор [16].

Ограничения *HadoopMapReduce*. Инфраструктура *Hadoop*, первоначально разрабатываемого для выполнения заданий *MapReduce*, берет на себя все сложные аспекты распределенной обработки: параллелизм, планирование, управление ресурсами, взаимодействие между машинами, обработку отказов ПО и оборудования и др. Вместе с тем, с появлением альтернативных моделей программирования (новых средств обработки на основе *Spark*, *Tez*, *Storm* и др.) возрастает потребность в поддержке отличных от *MapReduce* парадигм программирования, которые могли бы эффективно использовать те же кластеры и общие ресурсы. Ограничения инфраструктуры *MapReduce*, связанные с поддержкой рабочих нагрузок, отличных от *MapReduce*, не единственные. Другие – диктуются масштабируемостью и использованием ресурсов кластера. Проявляются они при создании масштабных кластеров, размером свыше 4 тыс. узлов, и при значительном

числе одновременно выполняемых задач [10]. Большие кластеры *Hadoop* выявили узкое место масштабируемости – наличие одного процесса *JobTracker*. Этот вид ограничений вынуждает создавать и поддерживать кластеры меньшего размера и меньшей мощности.

Вычислительные ресурсы каждого ведомого узла (*worker node*) в *HadoopMapReduce* делятся на фиксированное число слотов *map* и *reduce*. Слоты не взаимозаменяемы. Узел не может в любой момент выполнять больше задач *map*, чем имеется слотов *map*, даже если задачи *reduce* не выполняются. Это затрудняет эффективное использование кластера, поскольку при заполнении всех слотов *map* и необходимости в дополнительных нельзя использовать любой свободный слот *reduce* и наоборот. Это утверждение справедливо для кластеров любых размеров [10].

Итак, множество обязанностей, порученных одному процессу, приводит к серьезным проблемам масштабируемости, особенно в большом кластере, где процессу *JobTracker* приходится постоянно отслеживать тысячи процессов *TaskTracker*, сотни заданий и десятки тысяч задач *map* и *reduce*. В свою очередь процессы *TaskTracker* обычно выполняют всего около десятка задач.

Новое поколение *Hadoop YARN*. Эти недостатки устранены в вычислительной платформе *Hadoop YARN* (*Yet Another Resource Negotiator*), в которой процесс *JobTracker*, в обязанности которого входило управление ресурсами кластера и координация задач, существенно разгружен путем делегирования некоторых функций процессам *TaskTracker* на многочисленных узлах кластера (см. рис. 2) и модификацией вычислительных ресурсов. Вместо *JobTracker* введен менеджер ресурсов кластера (*ClusterResourceManager*), полностью ответственный за отслеживание в кластере работоспособных узлов и доступных ресурсов, а также за назначение их задачам (рис. 3). В архитектуре *YARN ClusterResourceManager* работает на выделенной машине как мастер–демон (*MasterDaemon*), распределяющий ресурсы кластера между конкурирующими приложениями. Решения о распределении он принимает с учетом совместного использования ресурсов, безопасности и множественной аренды (например, согласно приоритету приложения, вместимости очереди, спискам управления доступом, местоположению данных и др.).

На машинах кластера запущены менеджеры узла (*NodeManager*) как более универсальная и эффективная версия процесса *TaskTracker*. Вычислительные ресурсы приведены к виду, когда вместо фиксированного числа слотов *map* и *reduce* на узлах кластера *NodeManager* имеют динамически создаваемые контейнеры ресурсов. Размер контейнера зависит от объема содержащихся в нем ресурсов, например, памяти и процессора, а число контейнеров в узле определяется параметрами конфигурации и общим объемом ресурсов узла.

Ранее мониторинг и координирование всех задач приложения выполнял все тот же процесс *JobTracker*. Теперь эти функции переданы мастеру приложения (*ApplicationMaster*). Каждому заданию соответствует отдельный короткоживущий процесс *ApplicationMaster* для управления выполнением задач только в данном задании. Этот процесс и задачи его приложения выполняются в контейнерах ресурсов, управляемых *NodeManager*.

Таким образом, координация жизненного цикла задания распределяется между всеми доступными машинами кластера, благодаря че-

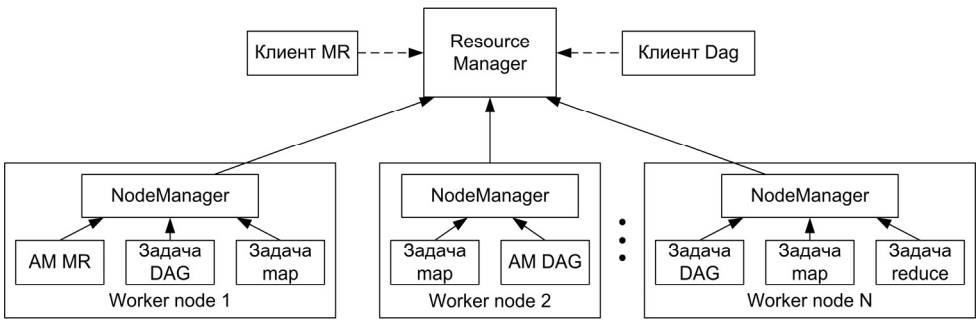


Рис. 3. *MapReduce*, реализованная в *Hadoop YARN*

му больше заданий выполняется параллельно, а масштабируемость резко возрастает.

С другой стороны, *ApplicationMaster* может выполнять любой тип задач внутри контейнера. Например, *MapReduce ApplicationMaster* запрашивает контейнер для запуска задач *map* или *reduce*, в то время как *DAG ApplicationMaster* запрашивает контейнер для запуска задач, представляемых *Tez* в виде ориентированного ациклического графа, и направленных на создание высокопроизводительных интерактивных приложений и пакетную обработку данных в реальном времени процесса. Для *ResourceManager*, *NodeManager* и контейнера не важен тип приложения или задачи. Весь специфицированный для конкретной инфраструктуры код приложения помещается в его *ApplicationMaster*, чтобы *YARN* мог поддерживать любую распределенную инфраструктуру. Предполагается разработка специализированных *ApplicationMaster* для выполнения иных конкретных задач.

Этот универсальный подход позволяет выполнять в одном кластере *Hadoop YARN* различные рабочие нагрузки: *MapReduce*, *Hive*, *Storm*, *Spark*, *Tez*/*Impala* и многое другое.

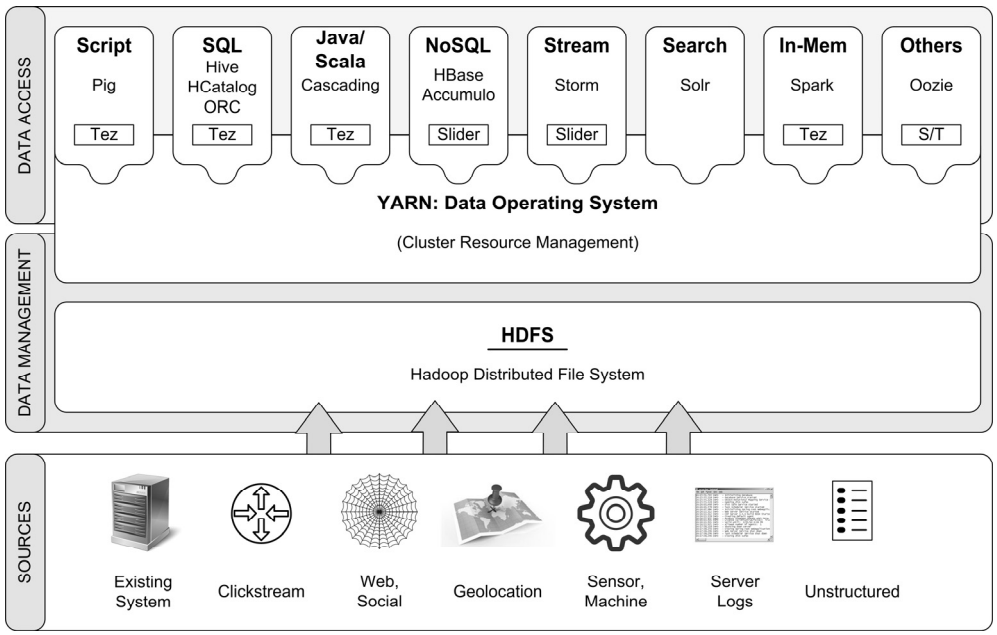
YARN – это абсолютно новая архитектура кластера *Hadoop*, изменяющая способ реализации и выполнения распределенных приложений в кластере стандартных машин, поскольку она предлагает очевидные преимущества в масштабируемости, эффективности и гибкости в сравнении с классическим механизмом *MapReduce* в первой версии *Hadoop*. Выигрыш от использования *YARN* получают как малые, так и большие кластеры. Для конечного пользователя эти изменения почти незаметны, поскольку новый механизм позволяет выполнять немодифицированные задания *MapReduce*, используя те же программные интерфейсы *MapReduce* и интерфейсы командной строки *CLI*.

С появлением *YARN* подход *MapReduce* превратился в распределенное приложение *MRv*2 – реализацию классического механизма *MapReduce* (*MRv*1), выполняемого поверх *YARN* [10].

***Enterprise Hadoop* – экосистема проектов**. *Apache Hadoop*® – проект с открытым исходным кодом для распределенного хранения и обработки на аппаратном обеспечении больших массивов структурированных и неструктурированных данных [17]. Многочисленные проекты *Apache Software Foundation* реализуют услуги, необходимые для развертывания, интеграции и работы с *Hadoop* [9, 12, 13, 17, 19].

Архитектурным центром *Apache Hadoop* сегодня является *YARN*, взявший на себя вынесенные из модуля *MapReduce* функции по управлению ресурсами и планированию заданий. Он гибко и на высоком уровне абстракции управляет кластером, как мультиарендной системой, несущей одновременно разнопрофильные и разнокалиберные нагрузки. Модуль ответствен за взаимодействие с вычислительными ресурсами и предоставляет их по запросу другим программам с учетом текущих возможностей, приоритетов, расписаний и ограничений. Таким образом, с одной стороны, *YARN* – связующее звено между коллекций данных в хранилище и средствами их аналитической обработки, с другой – высокоэффективный механизм, отделяющей компоненты управления ресурсами от компонентов обработки, позволяет выполнять в одном линейно масштабируемом кластере *Hadoop YARN* (*Hadoop* 2.0) разные рабочие нагрузки, достигая высокой эффективности вычислений (рис. 4). Тем самым обеспечивается управление подключаемой архитектуры для обеспечения широкого спектра методов доступа к данным, хранящимися на платформе *Hadoop*, с предсказуемыми уровнем производительности и обслуживания, планированием заданий и управлением кластером. Расширяя возможности *MapReduce*, *Hadoop* допускает использование распределенных приложений, выполненных с другими программными моделями. Проекты *Spark*, *Tez* и *Storm* полностью используют возможности *YARN*, привнося в *Hadoop* динамизм, ранее труднодостижимый в пакетной парадигме. Кроме того, *Hadoop YARN* предоставляет компоненты и *API*, необходимые для разработки приложений различных типов, что обеспечивает независимым поставщикам и разработчикам приложений доступ к слою данных *Hadoop*.

Платформа написана на языке *Java* и работает в среде *Linux*.



HadoopData Management – управление данными в слое хранения, масштабируемом линейно; связующая и управляющая ресурсами компонента *YARN*; *Sourses* – источники данных.

Рис. 4. Линейно масштабируемый кластер *Apache Hadoop YARN*

YARN также обеспечивает гибкость для новых и разрабатываемых методов доступа к данным, таким как *Apache Solr* (см. рис. 4) для масштабируемого полнотекстового [5], фасетного (предопределенного) и геопространственного поиска в больших массивах данных, распределенных на множество узлов, с улучшенной обработкой документов сложного формата (*PDF*, *Word*, *HTML*) и кластеризацией результатов поиска, поддержкой языка запросов структурного поиска так же хорошо, как и поиска по тексту.

***Доступ к данным* (*Data Acces*)**. Несмотря на то, что *Apache Hive* – наиболее широко принятая технология доступа к данным, есть много специализированных средств. Например, *Apache Pig* предоставляет возможности сценариев, *Apache Storm* – обработку в реальном времени, *Apache HBase* – колоночную (*columnar*) структуру хранения *NoSQL* и *Apache Accumulo* – контроль доступа сотового уровня. Все эти средства аналитики могут работать по одному набору данных и ресурсов благодаря *YARN* и таким промежуточным системам как *Apache Tez*™ и *Spark*™, обеспечивающим быстрое время анализа большего количества данных, и *Apache Slider* – для приложений длительного выполнения.

Следует отметить, что *Hadoop* продолжает привлекать новые программные решения вычислительных ядер для запуска на существующей платформе разнообразия задач обработки данных, так как организации желают эффективно хранить свои данные в едином хранилище и взаимодействовать с ними в разных направлениях. Они хотят *SQL*, потоковую обработку, машинное обучение наряду с традиционной пакетной обработкой и другими процессами в том же кластере. Пример тому – *Apache*™ *Storm* [15], который добавляет надежные в режиме реального времени возможности обработки больших объемов данных в *Hadoop Enterprise* [19].

Storm – производительное ядро для аналитики, машинного обучения и постоянного мониторинга операций реального времени с возможностью обрабатывать более миллиона записей в секунду на узел в кластере скромной размерности. Некоторые из конкретных новых возможностей для бизнеса включают в себя: управление в реальном времени, обслуживание клиентов, монетизацию данных, аналитику безопасности киберугроз. К примеру, в области телекоммуникаций – блокировать (исключить) нарушения безопасности и сетевые отключения, с другой стороны – оптимизировать распределение полосы пропускания и обслуживание клиентов; в промышленности – оптимизировать цепочки поставок и сократить время простоя производства; на транспорте – мониторинг перевозчиков, диагностическое обслужи-

[5] Полнотекстовый поиск – поиск документа в базе данных текстов на основании содержимого этих документов.

вание и оптимизация маршрутов, ценообразования; *Web*-прстранство – устранение ошибок приложений и персонализация информационного наполнения и др. [15, 16].

Tez представляет расширяемую основу программного ядра для создания высокопроизводительных пакетных и интерактивных приложений, координируемых *YARN* в *Apache Hadoop*. Это улучшает *MapReduce*-парадигму, значительно повышая быстродействие и сохраняя ее способность к масштабированию в заявленных *Apache Hadoop* объемах.

Поскольку *Tez* – есть расширяемым, т.е. предоставляет примитивы для создания распределенных приложений и механизмы обработки, предназначенные для проблемно-ориентированных приложений, он обеспечивает свободу организации вычислительного процесса и настраиваемую архитектуру выполнения заданий обработки данных с тем, чтобы создавать высокооптимизированные приложения. Это позволяет представлять стратегию вычислений в виде направленных ациклических графов (*directed acyclic graph*, *DAG*)[6]. Получив задание, сформулированное на языке таких стратегий, *Tez* может осуществить манипуляции над графами и, обладая доступом к сведениям о ресурсах, передать в *YARN* задачи так, чтобы решение было максимально эффективным, заранее рассчитывая нужные последовательности выполнения и определяя необходимые уровни параллелизма. Но поскольку распределенная обработка данных динамична, то трудно определить оптимальные методы перемещения данных заранее. Более подробная информация, доступная в процессе работы, может помочь оптимизировать дальнейший план выполнения. Для каждой вершины графа, представляющего вычислительную стратегию, *Tez* включает в себя поддержку подключаемых модулей управления для сбора информации во время выполнения и динамического изменения графа потоков данных. Используя реальную информацию о данных и ресурсах, при необходимости граф потоков данных динамически переконфигурируется, оптимизируя дальнейший план выполнения задания и производительность системы [14, 16].

Таким образом, *Tez* – обобщенная структура программирования потока данных, построенная на *Hadoop YARN* и обеспечивающая мощное и гибкое программное ядро для выполнения произвольных *DAG*-задач обработки данных как для пакетных, так и интерактивных сценариев использования. *Tez* принят к использованию в *Hive*™, *Pig*™ и другими структурами в экосистеме *Hadoop* [13].

Spark – быстрый, мощный обработчик данных в *Hadoop*. Представляет собой простую и выразительную модель программирования, поддерживающую широкий спектр приложений, используемых в исследованиях и аналитике: итерационные алгоритмы в машинном обучении, потоковую обработку и другое, в том числе *ETL*.

Hive – инфраструктура хранилища, построенного в рамках *MapReduce*, позволяет легко выполнять суммирование данных и осуществлять специальную обработку запросов, в частности формировать *Ad-Hoc* выборки, через интерфейс *SQL* для наборов данных, хранящихся в *HDFS*. Де-факто считается стандартом применение интерактивных запросов на *SQL*-подобном языке к большим данным. В режиме аналитики картина прямо противоположная – наибольшая нагрузка создается сравнительно редкими, но тяжелыми выборками (*select*) сотен тысяч и миллионов записей, часто с группировками и расчетом итоговых значений. Использование программного решения *Tez*, привнесшего ускорение более чем на порядок, сделало пригодным *Hive* для интерактивной аналитики. *Spark* также обеспечивает время анализа, более быстрое, чем *MapReduce*, значительного количества данных, улучшая производительность бизнес-решений [18].

Apache Hive позволяет работать с данными привычным способом, используя *SQL*-подобный язык запросов *HQL* (*Hive query language*)

[6] Направленный ациклический граф (ориентированный ациклический граф) – граф, в котором отсутствуют направленные циклы, т.е. пути, начинающиеся и кончающиеся в одной и той же вершине [*https*://*en.wikipedia.org*/*wiki*/*Directed_acyclic_graph*#cite_*note*-1].

так, как работали бы с обычной реляционной базой. Однако создание сложных *MapReduce*[7]-программ на языке *Java* требует навыков, значительных затрат времени и ресурсов. В большинстве организаций есть сотрудники, имеющие опыт работы с системами управления реляционными БД (*Relational Database Management System*, *RDBMS*) и языком структурированных запросов (*SQL*). *Hive* открывает экосистему *Hadoop* для неспециалистов программирования, предоставляя *SQL*-подобные возможности, а также функциональность, аналогичную функциональности БД. Позволяет разработчикам БД и аналитикам использовать кластер *Hadoop* без знания языков *Java* и *MapReduce*. *SQL*-запросы преобразуются в задания *MapReduce*, что означает снижение производительности некоторых запросов в сравнении с традиционными *RDBMS*.

В связи с последним замечанием *Hive* не предназначена для выполнения запросов с малой задержкой в режиме почти реального времени. Еще одно ограничение – что хранилищем по умолчанию служит база данных *Apache Derby*, не готовая к работе на уровне предприятия или в производственных условиях [20].

Еще одно обстоятельство, о котором следует помнить при работе с *Hadoop*: *Hive* – система *Schema on Read* (при чтении), в то время как *RDBMS*, как правило, являются системами *Schema on Write* (при записи). Традиционные *RDBMS* проверяют схему при записи данных. Данные, не соответствующие структуре, отклоняются. *Hive* не заботится о структуре базы данных и не проверяет схему при загрузке данных в базу. Она, напротив, проверяет схему только после выполнения запроса к базе данных [20].

Hive работает с текстовыми файлами, со сжатыми текстовыми файлами (*Gzip*, *Bzip*), массивами, словарями, объединениями (*union*). Имеет большое количество встроенных функций для работы с коллекциями, датами, строками, с текстовыми форматами обмена данными *JSON*. Работает с математическими функциями (округление, логарифмы, корни, тригонометрия) и с функциями агрегации (*sum*, *min*, *max*, *avg*, …). Можно использовать кастомные функции, а также мэпперы и редьюсеры (*python*, *java*) [21].

Pig, так же, как и *Hive*, с помощью *YARN* (см. рис. 3 и 4) способствует эффективному использованию *MapReduce* для анализа больших массивов данных. *Apache Pig* – это высокоуровневый процедурный язык *Pig Latin*, предназначенный для доступа к слабоструктурированным распределенным наборам данных, хранящимся в *HDFS*. Создает более простую абстракцию программы обработки данных в рамках *MapReduce* – сценарий (*scripting*) на *Pig Latin*, который во внутренней реализации *Pig* трансформирует последовательность операторов (инструкций), задающих процедуру решения задачи в серию заданий *MapReduce*. Они автоматически распараллеливаются и распределяются между узлами кластера с тем, чтобы реализовать *SQL*-подобный интерфейс для приложений *Hadoop*. *Pig*[8] – это мощный инструмент для выполнения запросов данных в кластере *Hadoop* [22].

Pig разработан для выполнения продолжительных циклов (*long series of data operations*) операций над данными, что делает его идеальным для трех случаев применения к большим данным [22]:

- *extract-transform-load* (*ETL*) – извлечение, преобразование и загрузка конвейерных данных;
- исследование исходных (сырых) данных;
- итерационная обработка данных.

ETL* – функции с использованием *Hive* и *Pig. Известно, что загрузка данных в БД традиционными средствами *ETL* требует значи-

[7] *MapReduce* – технология, которую программисты на *Java*, *C++* и *Python* могут освоить относительно быстро. Но без знания *Java* почти невозможно изучить *MapReduce*.

[8] *Pig* – мощный инструмент для выполнения запросов данных в кластере *Hadoop*. Этот язык настолько мощный, что по оценкам компании *Yahoo*! сценарии *Pig Latin* генерируют 40–60 процентов всей рабочей нагрузки в ее кластерах *Hadoop*. С учетом того, что в *Yahoo*! имеется 100 тыс. центральных процессоров и примерно на 50 процентов из них запущен *Hadoop*, *Pig* занимает внушительную долю [20].

тельных затрат времени. Стандартные инструменты для извлечения, преобразования и загрузки не позволяют эффективно справляться с *BigData*. Средства доступа к данным *Apache Hive* и *Apache Pig*, включенные в экосистему *Hadoop*, вполне пригодны для выполнения задач извлечения, преобразования и загрузки данных различных типов. В отличие от многих традиционных инструментов, которые хорошо подходят для обработки структурированных данных, *Hive* и *Pig* можно использовать, судя по изложенному материалу, для загрузки и преобразования неструктурированных, структурированных или полуструктурированных [9] данных в *HDFS* [20, 21, 23].

Но изначально следует определить, в каком виде целесообразно представлять в хранилищах большие и уникальные по своей сути данные. Если допустить, что некто примет решение о том, какие данные необходимы, а какие можно отбросить, назовет схему загрузки (например, *звезда* или в полунормализованном формате и др.), то данным будет нанесен существенный урон, так как после этой процедуры хранилище данных становится репозиторием, а исходные (сырые) данные не хранятся и, следовательно, не могут быть извлечены. Это и есть ключевое ограничение концепции *ETL*.

Вместе с тем, в *Hadoop*, в отличие от реляционных БД, в которых схемы должны быть определены до помещения данных на хранение, можно относительно просто сбросить данные в *HDFS* и в дальнейшем позаботиться о схеме. Проверка схемы, разработка структуры БД проводится только после выполнения к ней запроса [20]. В этой связи предложена другая концепция – *ELT* (извлечение, загрузка и преобразование) [24]. Она стала более распространенной вследствие появления технологии *Hadoop*, а также по причине значительного уменьшения стоимости оборудования и систем хранения. По-прежнему извлекаются данные из ряда источников, но затем они не преобразуются, а сначала загружаются необработанные данные в *HDFS*. Зачастую для процесса загрузки не требуется схема, и данные могут долгое время оставаться необработанными. Когда данные потребуются, нужно будет создать схему, преобразовать данные и определить, как их анализировать.

Преимущество заключается в том, что необработанные данные могут оставаться в хранилище долго, и кто-то другой может использовать их на свое усмотрение, а не так, как, например, много лет назад было решено денормализовать и спроектировать систему.

Достаточно подробно изложены все этапы от импорта данных, проектирования схемы до выполнения запроса к БД *Hive* в работе [24]. Конечно, у *Hive* есть ограничения, но экосистема *Hadoop* – отличный инструмент, чтобы приступить к созданию БД, заполнению таблиц, преобразованиям и интеграции данных.

Apache HBase – масштабируемая БД *Hadoop*, хранилище *BigData* [25, 26]. *HBase*, колоночная БД, поддерживает структурированное хранение данных очень больших таблиц – миллиарды строк и миллионы столбцов, что делает ее хорошим выбором для хранения мульти-структурированных или разреженных (*multi-structured or sparse data*) данных. Последнее хорошо сочетается с широким спектром источников данных различного назначения. *HBase* обеспечивает произвольный прямой доступ к *BigData* в операциях чтения/записи в режиме реального времени. Ее особенности – линейная и модульная масштабируемость, автоматический и настраиваемый *Sharding* таблиц (разбиение больших таблиц на логические части по выбранным критериям) и др. Модель доступа к данным имеет следующие ограничения:

- поиск ряда по одному ключу;
- доступны только операции по одной строке;
- не поддерживаются транзакции.

Apache это *HBase* с открытым исходным кодом, нереляционная[10], по образцу базы данных

[9] Полуструктурированные данные подразумевают логическую схему и формат, который может быть понятным, но не дружественным к пользователю. Примером могут быть логии, собираемые в журналах [23].

[10] *NoSQL* (*not only SQL*, не только *SQL*) нереляционные БД, распределенные, с открытым исходным кодом и горизонтально масштабируемые. Как и колоночные БД, *NoSQL* в

Google Bigtable. Так же как *Bigtable* обеспечивает распределенное хранение данных с поддержкой файловой системы *Google File System*, так и *Apache HBase* использует распределенную систему *Hadoop – HDFS*.

HBase – распределенная нереляционная *DBMS*, эффективно позволяющая работать с отдельными записями в реальном времени, служит существенным дополнением к инфраструктуре *Hadoop*.

Apache Accumulo [28] – ориентированное на высокий уровень безопасности распределенное, масштабируемое хранилище данных для здравоохранения, финансов, предпринимательства, государственных учреждений и других организаций со строгими требованиями к защите информации и персональных данных граждан, высокопроизводительная система информационного поиска. В этом – ее направленность и основное отличие от *HBase*. Создана по тому же принципу, что и *Google BigTable*: *NoSQL* хранилище пар «ключ – значение» (*key – value*) и СУБД для управления большими объемами различных структур данных. В *BigTable* ключами для доступа к данным служат три позиции:

- ключ строки (*row key*),
- ключ столбца (*column key*),
- временная метка (*timestamp*), используемая для управления версиями и сборки мусора.

Запрос к системе с определенным ключом вернет соответствующие данные. Положенное в основу *Google BigTable* сокращение времени обработки запросов, зависящее в немалой степени от выполнения операций чтения/записи на дисковые массивы, оптимизация производительности систем хранения и доступа к большим информационным объемам, достигается в *NoSQL* использованием алгоритмов древовидных структур и механизмов их реализации. *LSM-Trees* (*Log-Structured Merge-Trees*, *LSM*) – деревья слияния со структурой (*log*)-журнала – являются структурой данных, обеспечивающей низкую стоимость операции индексирования и практически одинаковую высокую скорость операций чтения и записи при последовательном и произвольном доступе к данным (запись данных осуществляется быстрее чтения) [29]. Краткое изложение концепции *LSM-Trees* и ее работу можно найти в русскоязычных изданиях, например [30, 31].

В *Apache Accumulo* добавлены две уникальные особенности [32]. Одна из них состоит в наличии механизма (*iterator framework*) программирования пользователем функциональности[11] (кодирования функций, таких как фильтрация, агрегации и др.), встраиваемой в разные фазы *LSM-Trees*, что допускает изменение пары «ключ–значение» на различных этапах управления данными *BigTable*. Вторая – безопасность сотового уровня, позволяющая разграничивать права доступа к данным на уровне ячейки таблицы (*cell-level*), содержимого отдельных записей (*fine-grain*). Распределенная таблица, содержащая упорядоченные ряды пар «ключ–значение», показана на рис. 5. Ключ (*Key*) состоит из идентификатора строки (*Row ID*), колонки и ключа временной метки (*Timestamp*). Ключи колонки (*Column*) состоят из отдельных столбцов: семьи столбцов (*column family*), квалификационных (*qualifier*) и видимость колонки (*visibility*). Поле видимости ко-

сравнении с транзакционными РСУБД более производительны при некоторых видах интенсивных операций, например, предоставление веб-страниц или потоковая передача данных. Популярность *NoSQL*-решений растет, и связано это в первую очередь с новыми задачами по обработке больших объемов данных, требованиями по доступности и масштабируемости.

Горизонтальное масштабирование существующих традиционных РСУБД обычно трудоемкая, дорогостоящая и эффективная только до определенного уровня задача. В то же время многие *NoSQL*-решения проектировались исходя из необходимости масштабироваться горизонтально и делать это «на лету». Поэтому эта процедура обычно проще и прозрачнее в *NoSQL*, чем в РСУБД.

Производительность БД на одном узле, а не в кластере также важна. Для многих задач отказ от соблюдения требований *ACID*, предъявляемых РСУБД, позволяет иногда добиваться большей производительности на одном узле, чем традиционным решением [27].

[11] Механизм программирования на стороне сервера для кодирования функций, таких как фильтрация и агрегации в пределах шагов по управлению данными (область, из которой данные считываются или записываются на диск).

лонки используется для защитного признака сотового уровня.

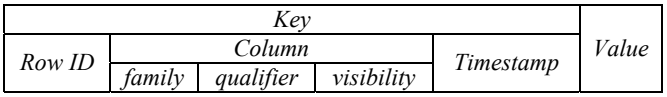


Рис. 5. Структура пары «ключ–значение» в *Apache Accumulo*

Итак, имеется возможность присвоить каждой ячейке таблицы данных определенную метку. В каждом ключе есть секция под названием «видимость колонки», где и хранятся метки, обеспечивающие детальный контроль доступа к данным. Во время запроса пользователи должны предоставить учетные данные, проверяемые в отношении хранящихся меток доступа. В каждой записи пользователи видят только те поля, которые они уполномочены видеть. Полномочия передаются с каждым запросом, чтобы контролировать, какие данные возвращаются пользователю.

Например, внешнему серверу можно дать доступ только к заданным ячейкам хранилища, снабженным соответствующими метками на основе свода правил. Метки доступа в *Accumulo* полностью проблему безопасности не решают. Это – механизм для снабжения каждого фрагмента данных сведениями о правах, которые нужно иметь, чтобы увидеть их. Защитные метки позволяют по мере необходимости объединять данные без дорогостоящего их перемещения и изоляции.

Эта технология представляет собой прорыв в хранении данных, устраняя необходимость физически изолировать данные различных уровней безопасности, обеспечивая логическую развязку на протяжении всего процесса хранения данных и выполнения запросов.

Accumulo первоначально разработана в Агентстве национальной безопасности (АНБ) США, прежде чем в 2011 г. была передана некоммерческой организации *Apache Software Foundation* в качестве инкубационного проекта с открытым исходным кодом и положена в основу хранилища и СУБД *Apache Accumulo*. В ней использован ряд других наработок *Apache*: платформа распределенного хранения данных *Hadoop*, менеджер конфигурации распределенных приложений *Zookeeper* и инструментарий разработки сервисов *Thrift*. Благодаря ее происхождению в разведывательном сообществе, *Accumulo* обеспечивает чрезвычайно быстрый доступ к данным (способен хранить большие таблицы записей, отсортированных по рекордному числу *IDs* идентификаторов, и предоставляет механизмы для быстрого получения одной или нескольких записей из любой таблицы), в то же время контролирует доступ к отдельной ячейке среди миллиардов строк и миллионов столбцов. Это известно как (*as fine-grained*) точное управление доступом к данным [28, 33].

Реализованный специалистами АНБ подход к безопасности на основе меток подобен решению с открытым кодом, выпущенному в 2000 г., – *Security Enhanced Linux* (*SE Linux*). Посредством *SE Linux* администраторы систем могут создавать наборы правил, детально описывающие, какие действия на компьютере может выполнять каждая программа. Компания *Red Hat* интегрировала *SE Linux* в свой дистрибутив *Red Hat Enterprise Linux*.

Apache Mahout™: масштабируемость машинного обучения и интеллектуального анализа данных. Библиотека – 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 сложных вопроса: машинного обучения (*ML – Machine Learning*) и распределенной обработки больших данных.

1. *The Digital* Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things. EMC Digital Universe with Research & Analysis by IDC. – http://www.emc.com/leadership/digital-universe/2014-iview/index.htm.April2014
2. *Big Data* // Nature. – 2008. – **455**, N 7209. – C. 1–136. – http://www.nature.com/nature/journal/v455/n7209/index. html
3. *John F. Gantz*, *David Reinsel*. The Digital Universe in 2020: Big Data, Bigger Digital Shadows, and Biggest Growthin the Far East. Dec. 2012. – http://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf

4. *Michael Chui*, *Markus Loffler*, and *Roger Roberts*. "The Internet of Things", McKinsey Quarterly, March 2010. 22.
5. *Big* data: The next frontier for innovation, competition, and productivity / J. Manyika, M. Chui, B. Brown et al. – May 2011. – www.mckinsey.com/~/media/McKinsey/dotcom/Insights%20and%20pubs/MGI/Research/Technology%20and%20Innovation/Big%20Data/MGI_big_data_full_report.ashx
6. *Carl W. Olofson*, *Dan Vesset*. Big Data: Trends, Strategies, and SAP Technology. – http://www.itexpocenter.nl/iec/sap/BigDataTrendsStrategiesandSAPTechnology.pdf, Aug. 2012. External Publication of IDC Information and Data.
7. *Hinchcliffe Dion*. The enterprise opportunity of Big Data: Closing the "clue gap". – http://www.zdnet.com/article/the-enterprise-opportunity-of-big-data-closing-the-clue-gap/
8. *Черняк Л*. Большие Данные – новая теория и практика // Открытые системы. – 2011. – № 10 – http://www.osp.ru/os/2011/10/13010990
9. *CLOUDERA*. Hadoop and Big Data – http://www.cloudera.com/content/cloudera/ en/about/hadoop-and-big-data.html
10. *Кава Адам*. Введение в *YARN* – *http*://*www.ibm*. com/*developerworks*/*ru*/*library*/*bd-yarn-intro*/, 11.11.2014 (Впервые опубликовано 01.04.2014).
11. *Apache* Spark – http://cloudera.com/content/cloudera/en/products-and-services/cdh/ spark.html
12. *CDH* – 100% Open Source Distribution including Apache Hadoop – http://www. cloudera.com/content/cloudera/en/products-and-services/cdh.html
13. *HORTONWORKS*. What is Apache Hadoop®? – http://hortonworks.com/hadoop/
14. Apache Tez – http://hortonworks.com/hadoop/tez/
15. *Apache* Storm – http://hortonworks.com/hadoop/storm/
16. *Николаенко А*., *Волков Д*. Новые инструменты *Hadoop* // Открытые системы. – 2014. – № 10. – *http*://*www.osp.ru*/os/2014/10/ 13044382/
17. *MAPR*. What is Apache™ Hadoop®? – https://www.mapr.com/products/apache-hadoop
18. *Hadoop*: что, где и зачем – *http*://*habrahabr.ru*/*post*/240405/, 16 окт. 2014.
19. *What* Is Apache Hadoop? – https://hadoop.apache.org/
20. *Джамак П*. Создание библиотеки данных при помощи *Hive*. – *http*://*www.ibm.com*/*developerworks*/*ru*/*library*/*bd-hivelibrary*/, 11 окт. 2013.
21. *Никулин А*. Hive vs Pig. На что мне столько ETL? – *http*://*habrahabr.ru*/*post*/223217/, 23 мая 2014.
22. *Джонс М*. Тим. Обработка данных при помощи Apache Pig. – *http*://*www.ibm.com*/*developerworks*/*ru*/*library*/l-*apachepigdataquery*/, 20.11.2012.
23. *Френкс Б*. Укрощение больших данных: как извлекать знания из массивов информации с помощью глубокой аналитики. – М.: Манн, Иванов и Фербер, 2014. – 352 с.
24. *Джамак П*. *Hive* как инструмент для *ETL* или *ELT*. – *http*://*www.ibm.com*/*developerworks*/*ru*/*library*/*bd-hive-tool*/, 14 мая 2014.
25. *Apache* HBase. – http://hortonworks.com/hadoop/hbase/, May 2010.
26. Apache HBase. – http://hbase.apache.org/, May 24 2016.
27. *Бодров И*. Сильные и слабые стороны *NoSQL*. – *http*://*www.jetinfo.ru*/*stati*/*silnye-i-slabye-storony-nosql*, Jet Info. – июль 2012. – № 6.
28. *Apache* Accumulo. – http://hortonworks.com/hadoop/accumulo/ – http://accumulo.apache.org/
29. *The* Log-Structured Merge-Tree (LSM-Tree) / P. O'Neil, E. Cheng, D. Gawlick et al. // Acta Informatica. – 1996. – **33**, Issue 4. – P. 351–385.
30. *Межов А*. SSTable и LSM-Tree. – *http*://*www.mezhov.com*/2013/09/*sstable-lsm-tree.html*, 24 сент. 2013.
31. *Зубинский А*. *NoSQL* СУБД, Ч. II, «*KVS*» // Компьютерное Обозрение. – *http*://*ko.com.ua*/*nosq*l_*subd_chast_vtoraya_kvs*_103598, 29 января 2014.
32. *Sen Ranjan*, *Farris Andrew*, *Guerra Peter*. Benchmarking Apache Accumulo BigData Distributed Table Store Using Its Continuous Test Suite. – Int. Congr. on Big Data, 2013. – http://sqrrl.com/media/Accumulo-Benchmark-10312013-1.pdf
33. *Джексон Джоаб*. АНБ реализует меточную модель безопасности в Больших Данных. Служба новостей IDG, Нью-Йорк // Сети*network world*. – 2011. – № 04. – *http*://*www.osp.ru*/*nets*/2011/ 04/13010801/

Поступила 30.03.2016
Тел. для справок: +38 044 526-4159 (Киев)

UDC 004.7:004.75:004.9:004.738.5

A.A. Oursatyev

SomeFrameworks forAnalytics Big Data

The need to extract data from new information forces the developers of the analytic systems to pay attention on radical improvement of the traditional processing technology and to create the advanced analytics environments.

The conceptual issues of data media construction, in particular, on the *Hadoop* cluster system software platform is presented. The *HadoopMapReduce* infrastructure is described for the parallel distributed computing on the data and the evolutionary transformation of *Hadoop* platform using the infrastructure and streaming dynamic loads, as well as *HadoopMapReduce* infrastructure constraints. It is shown that an introduction of *YARN* (*Yet Another Resource Negotiator*) on the computing

Hadoop platform allows to perform the different workloads in a linearly scalable cluster *Hadoop YARN* (*Hadoop* 2.0), achieving calculations of the high efficiency. *Framework*$_{s,}$ *Spark*, *Tez and Storm* use the possibility of *YARN* .

The components that make a total *Hadoop* 2.0 de facto the standard technology for working with Big Data are analyzed. These are the constructions ***Hive*** for design-oriented interactive queries to *SQL*-like language *HQL* (*Hive query language*) and working with large data storage; ***Pig*** – a high-level procedure language *Pig Latin*, designed for accessing the semi-distributed lennym datasets; ***HBase*** – distributed non-relational *DBMS*, working effectively with the individual records in real time; ***Apache Accumulo*** – oriented on a high level of safety distributed, scalable data repository with the strict requirements of the information and personal data protection.

The problems of large data efficiently various types download of *Hadoop* ecosystem using *Hive* and *Pig*. A comparative analysis of *ELT* (*extract-load-transform*) and *ETL* (*extract-transform-load*) concept is presented. The first one is widely spread due to the emergence of *Hadoop* technology.

●