УДК 004.67
DOI: 10.46548/21vek-2020-0950-0023

АЛГОРИТМ БЕЗОПАСНОЙ ОБРАБОТКИ БОЛЬШИХ ДАННЫХ ОБЛАЧНЫМИ СЕРВИСАМИ С ПОМОЩЬЮ ТЕХНОЛОГИЙ BIG DATA

Мартышкин Алексей Иванович, кандидат технических наук,
доцент, доцент кафедры «Вычислительные машины и системы»
Коростелев Александр Владимирович, аспирант кафедры «Информационные технологии и системы»
Пензенский государственный технологический университет
(440039, Россия, Пенза, проезд Байдукова/ул. Гагарина, д. 1а/11,
e-mails: alexey314@yandex.ru , bezpaniki@hotmail.com)

Аннотация. Статья посвящена рассмотрению некоторых вопросов, связанных с разработкой алгоритма безопасной обработки больших данных облачными сервисами с помощью технологий Big Data. Сегодня обработка больших объемов данных является одной из наиболее актуальных задач. В ходе проведенных работ в статье предложен алгоритм, который позволяет осуществлять обработку больших объемов данных посредством набора инфраструктурных облачных сервисов. Алгоритм позволяет обрабатывать данные из пользовательских защищенных источников, без хранения какой-либо информации после процесса обработки. Он имеет возможность быть встроенным в инфокоммуникационные сети, формируя подсистему виртуального частного облака. Достаточно подробно описаны решенные при написании статьи задачи: проанализирована предметная область аналитики больших данных и облачных систем; выбраны объекты и компоненты инфраструктуры алгоритма для разработки и исследования; спроектирована архитектура процессов алгоритма; выбраны публичный облачный провайдер Amazon Web Services и сервисы для реализации алгоритма; наконец, разработан и протестирован алгоритм на инфраструктурных сервисах Amazon Web Services. Описанная в статье реализация системы является легко масштабируемой облачной распределенной вычислительной системой. Это позволяет использовать отдельные инфраструктурные элементы как сервис. Таким образом, алгоритм позволяет безопасно обрабатывать большие данные, затем выгружая их в хранилища любой заданной защищенности: как облачные, так и местные. Алгоритм обладает гибкими возможностями трансформации и применения как в публичных, так и в гибридных и кросспровайдерных облачных структурах разных уровней защищенности. Предложенная его реализация позволяет оперативно развертывать облачную инфраструктуру.

Ключевые слова: алгоритм, безопасность, большие данные, инфраструктура, масштабируемость, маршрутизация, нереляционная база данных, облачная система, облачные сервисы, реляционная база данных.

ALGORITHM FOR SECURE PROCESSING OF BIG DATA BY CLOUD SERVICES USING BIG DATA TECHNOLOGIES

Martyshkin Alexey Ivanovich, candidate of technical sciences,
docent, associate Professor of sub-department «Computers and systems»
Korostelev Alexander Vladimirovich, postgraduate of sub-department
«Information technologies and systems»
Penza state technological University
(440039, Russia, Penza, Baydukov Proyezd / Gagarin Street, 1a/11,
e-mails: alexey314@yandex.ru , bezpaniki@hotmail.com)

Abstract. The article deals with some issues related to the development of an algorithm for secure processing of big data by cloud services using Big Data technologies. Today, processing large amounts of data is one of the most urgent tasks. In the course of this work, the article proposes an algorithm that allows processing large amounts of data through a set of infrastructure cloud services. The algorithm allows you to process data from user-protected sources, without storing any information after the processing process. It can be embedded in infocommunication networks, forming a virtual private cloud subsystem. Sufficient detail is resolved when writing objectives: analyzed the subject area of big data Analytics and cloud systems; selected objects and infrastructure components of algorithm development and research; designed the architecture of the processes of the algorithm; the chosen public cloud provider Amazon Web Services and services for the implementation of the algorithm; finally, developed and tested the algorithm on infrastructure Amazon Web Services. The system implementation described in this article is an easily scalable cloud distributed computing system. This allows you to use individual infrastructure elements as a service. Thus, the algorithm allows you to safely process big data, then uploading it to storage of any given security: both cloud-based and local. The algorithm has flexible capabilities for transformation and application in both public and hybrid and cross-platform cloud structures of different security levels. The proposed implementation allows you to quickly deploy cloud infrastructure.

Keywords: algorithm, security, big data, infrastructure, scalability, routing, non-relational database, cloud system, cloud services, relational database.

Введение. В современном мире обработка больших объемов данных является одной из наиболее актуальных задач. С ростом информатизации процессов и развитием инфокоммуникационных технологий растет также количество передаваемых и обрабатываемых данных. По результатам исследования около 89% опрошенных ИТ-специалистов назвали использование *Big Data* столь же важным, как и появление Интернет [1]. Параллельно с обработкой информации встает и вопрос её защиты [2]. Безопасность остается одним из главных препятствий при внедрении облачных структур, так необходимых при анализе *Big Data* [1]. Несмотря на увеличивающуюся долю *Big Data* в инвестиционных кампаниях крупных корпораций, научных трудов, связанных с решениями *Big Data* не так много. Большинство материалов представлено на английском языке в зарубежных источниках. Технические материалы и передовые наработки по реализации в наиболее полном виде представлены документацией продуктов крупных компаний, продукты которых создаются для решения подобных задач. Перспективным видится создание простых в пользовании систем аналитики больших данных, доступных пользователям без необходимости изучения сложных технических аспектов. Шагом вперед в этом направлении являются публичные облачные сервисы, которые позволяют использовать продукты на высоком уровне абстракции от технических процессов, обладают исчерпывающей и доступной для понимания документацией, а также просты в настройке.

Целью работы является определение стека технологий для построения процесса безопасной обработки больших данных, определение необходимых компонентов для него, в том числе среди облачных сервисов: публичных и частных. Для достижения указанной цели требуется решить ряд задач: исследовать предметную область; определить требования к алгоритму; исследовать имеющиеся наработки и практики; определить объекты и компоненты архитектуры для реализации алгоритма, исходя из требований; определить возможность применения различных облачных сервисов; реализовать алгоритм и протестировать сценарий несанкционированного доступа к инфраструктурным компонентам. При построении архитектуры алгоритма стоит учесть перспективы его применения не только в рамках конкретной задачи, но и готового к последующему использованию технологии в других вычислительных системах.

Материалы исследования. Для лучшего понимания рассматриваемой проблематики, необходимо понимать предмет исследования и его область применения. Разработка алгоритма велась в рамках построения архитектуры *Business Intelligence* системы для мониторинга и контроля ИТ-инфраструктуры крупных корпораций в соответствии с концепцией *ITSM (IT Service Management)*. Чтобы иметь прозрачность показателей использования ИТ-инфраструктуры, приходится обрабатывать большие объемы данных об утилизации, стоимости и ряде других метрик, поступающих с ресурсов. В результате этого появилась необходимость использования современных подходов в разработке технических решений. Ключевым процессом в системе загрузки данных является *Extract Transformation Load (ETL)* процесс [3]. Задаваясь вопросом, будут ли предполагаемые потоки данных «большими данными», необходимо понять, что подразумевает использование данного термина. Несмотря на то, что термин был введен еще в 2008 году Клиффордом Линчем, четкого определения он не дал [4]. Для понимания термина принято использовать три основных принципа больших данных (три *V*): *Volume* (объем), *Velocity* (скорость), *Variety* (разнообразие) [5].

Как подмечено в работе [6], взрывной рост объемов данных и неспособность традиционных методов аналитики справиться с лавинообразным увеличением поступающей информации связаны с увеличивающейся информатизацией бизнес-процессов и накапливающимися данными, с которыми пользователи и аналитики просто не знали, что делать. Объемы данных увеличивались, но не развивалась теория информации, что привело к тому, что потребность во взаимодействии с появляющимися данными образовалась раньше теоретических возможностей её реализации. Только спустя время проблема больших данных переросла в парадигму и подход к их обработке и аналитике, которые наряду с облачными сервисами на сегодняшний день не только взаимосвязаны – сейчас работа с большими данными практически невозможна без облачных хранилищ и вычислений. Одним из главных преимуществ облачных сервисов является возможность предоставления сервисов на разных уровнях абстракции от физического управления ресурсами (рис.1).

Рисунок 1 – Уровни абстракции над облачной инфраструктурой

Нижний уровень абстракции предполагает использования инфраструктуры как сервиса (*Infrastructure as a Service, IaaS*). Такая модель сравнима с использованием локальных ресурсов, с той разницей что ресурсы физически находятся у провайдера [7]. Провайдером облачной услуги может быть как ИТ-департамент компании в случае развертывания частного облака. Такая форма взаимодействия – пуб-

личное облако. Более высоким уровнем абстракции является предоставление платформы как как сервиса (*Platform as a Service, PaaS*). Это означает, что сущест-вует возможность управлять элементами ИТ-инфраструктуры, не вдаваясь в тонкости настройки, а просто приобретая готовые решения по требованию. Самый верхний уровень абстракции – программное обеспечение как сервис (*Software as a Service, SaaS*). Пользователь получает полноценное программное обеспечение, скрывающее под собой облачную архитектуру и инфраструктуру. Из трех моделей предоставления облачных сервисов наиболее применимой в контексте алгоритма может быть *IaaS* модель либо *PaaS* платформы с возможностью изменением конфигурации машин.

Отметим, что облачные структуры могут быть частными, разворачиваемыми в локальных датацентрах, в публичном облаке (*Public Cloud*) или гибридом облаке (*Hybrid Cloud*). Частное облако предполагает автоматизацию процессов использования ИТ-ресурсов внутри компании, когда пользователи заказывают ресурсы, а их предоставление и оплата происходит автоматически. Положительной особенностью использования частного облака является полная его изоляция, и вследствие этого возможность поддержки критически высокого уровня требований безопасности. Также ресурсы могут быть незначительно более производительными, так как центры обработки данных (ЦОД) компаний находится недалеко от пользователей. Частные облака можно сконфигурировать по требованию пользователя.

Гибкость и оперативная адаптация инфраструктуры на изменения нагрузки, так что в любой момент времени ресурсы отвечают требованиям спроса настолько, насколько возможно, называется эластичностью мощностей [8]. В области обработки больших данных эластичность вычислительных мощностей и хранилищ крайне необходима, так как она удовлетворяет по двум из трех требований к работе с *Big Data – Volume и Velocity*.

Эластичные облачные хранилища могут использоваться для накапливания и хранения разнородных и увеличивающихся объемов данных. Такие легко доступные эластичные репозитории, в которых данные хранятся в натуральном неструктурированном формате, и объем которых можно легко и оперативно увеличивать, называются даталейки (*Data Lake*) [9]. Эластичные вычислительные мощности могут обеспечивать оперативную обработку больших объемов данных. Так как количество вычислительных мощностей может подстраиваться под нужды загрузки, возможно обеспечить быструю загрузку и обработку данных в системе на кластере распределенных серверов.

Анализ функциональных требований. Разумно полагать, что одним из начальных вопросов, возникающих при проектировании алгоритмов работы *ETL* систем, является вопрос к свойствам данных, которые нужно обрабатывать: их размеру, механизму и формату загрузки, а также требованиям безопасности, выдвигаемым к подсистеме. Для начала, стоит оценить объемы данных, поступающих в систему. Так как разработка алгоритма велась при создании системы мониторинга ИТ-затрат, стоит оценить количество обрабатываемых данных. Алгоритм обработки должен иметь минимальную зависимость от объема поступающих данных и должен позволять оперативно обрабатывать постоянно увеличивающиеся объемы данных, либо должен подразумевать возможность масштабирования в зависимости от желания пользователя.

Механизм и формат загрузки должны описывать выкачку данных из источника информации. В систему мониторинга и отчетности об ИТ-инфраструктуре данные могут поступать как постоянным потоком, так и в консолидированном виде архивов данных за временной промежуток и агрегированные по подразделению организации.

Алгоритм обработки больших объемов данных должен иметь возможность обеспечения общих корпоративных требований безопасности. Подсистема должна предусматривать возможность установки дополнительных мер безопасности, таких как выгрузка данных из защищенных источников, разрывание каналов связи источника и целевой БД, отсутствие промежуточных звеньев между источником данных и целевой БД, которые могли бы хранить данные, а также другие возможности повышения уровня безопасности.

Основными функциональными требованиями к алгоритму обработки являются: возможность масштабируемости инфраструктуры; оперативность обработки данных; обеспечение требований безопасности.

Результаты исследования. Приступая к проектированию алгоритма, нужно понимать, какие практики уже были предприняты для обработки больших данных. Одно из наиболее расхожих определений больших данных – разнородные данные большого объема, которые невозможно обработать привычными способами [6]. Стало быть, методика обработки данных в *Big Data* системах принципиально отличается от привычных способов дата-процессинга. Для проектирования компонентов обработки будем исходить из наличия трех стадий обработки: извлечение данных, трансформация и загрузка внутрь системы.

В мире *Big Data* есть наиболее расхожий стек технологий, формируемый внутри экосистемы компании *Apache*. Этот стек технологий и подходов объединяет в себе программные компоненты для процессов работы с большими данными и называется *Apache Hadoop*. Несмотря на широкую распространенность и поддержку сообществом, иногда эти технологии накладывают ограничения на ресурсы. Например, в ходе проработки было выяснено, что файловая система *Hadoop Distributed File System – HDFS* отбирает часть ресурсов вычислительной машины, поэтому ее использование не всегда

оправдано при необходимости высоких нагрузок. Поэтому было принято решение взять за основу компоненты и идеологию *Hadoop* и адаптировать её под свои нужды для наилучшей оптимизации.

Начнем с выбора инфраструктурных объектов для алгоритма на каждой из стадий *ETL* процесса. Ввиду того, что большие данные относительно недавно вошли в стадию активной проработки и реализации, физических инфраструктурных компонентов для реализации *Big Data* систем не так много.

Первой стадией *ETL* процесса является *Extract* – извлечение данных из источника-хранилища. Как было отмечено, распространенной формой хранения данных являются даталейки, у которых, однако, есть разные режимы работы: базовый даталейк, холодный даталейк, глейсер данных, датасвомп. При выборе хранилища для исходных данных стоит выбирать между обычным либо холодным даталейком и избегать попадания в режим свомпа. Конкретный режим работы необходимо выбирать исходя из понимания того, как стоит поступать с данными после их обработки. Если они должны оставаться в исходном хранилище и реплицироваться в систему, то разумнее использовать холодный даталейк [10].

Далее для реализации алгоритма обработки следует изучить стадию трансформации данных. Одной из главных проблем приложений, ориентированных на обработку больших данных, является вычислительная сложность процессов. Как было показано выше, необходимо учитывать постоянно растущие объемы поступающих данных, но важно понимать, что требования к вычислительным мощностям линейно возрастают по отношению к объемам [11]. Традиционная схема клиент-серверной архи-тектуры приложений с одним сервером – обработчиком не подойдет по причине слабой его масштабируемости. Отдельное улучшение вычислительных компонентов одного сервера видится намного сложнее, нежели создание параллельного процесса обработки большого объема данных между несколькими вычислительными ресурсами [3]. Такие кластеры серверов с контейнерами приложений внутри и используются в практике работы с *Big Data*. Кластером серверов могут управлять различные приложения, обеспечивающие эффективное управление нагрузками на сервера, распределение задач и легкость абстракции над машинами. Для трансформации данных исследуемым алгоритмом применяется кластер эластичных серверов. Это позволит эффективнее использовать вычислительные мощности, а также писать разнообразные приложения для задач трансформации данных. Такой подход сделает алгоритм более гибким и откроет возможности для использования его в дальнейшем для многих систем работы с большими данными.

После трансформации данных необходимо загрузить их в систему. Традиционные практики использования реляционных баз данных и *SQL* здесь также не подходят ввиду низкой скорости доступа к данным и сложности работы с комплексными многообразными данными. Альтернативой является использование парадигмы *Not only SQL (NoSQL). NoSQL* базы данных являются распределенными, нереляционными, разработанными для работы с большими хранилищами данных и распределением параллельных потоков работы с данными на нескольких серверах [12]. Распределенные базы данных подразумевают возможность использования нескольких машин под управлением одного сервера БД, подобно кластеру вычислительных машин обработчика данных, а также отделять хранилище от машин – обработчиков запросов. Отказ от реляционной модели БД значит использование связей по иным моделям СУБД нежели первичные ключи к целым таблицам. Это могут быть модели «ключ-значение», графовые модели, документоориентированные СУБД и модели семейства столбцов [13] (рис. 2).

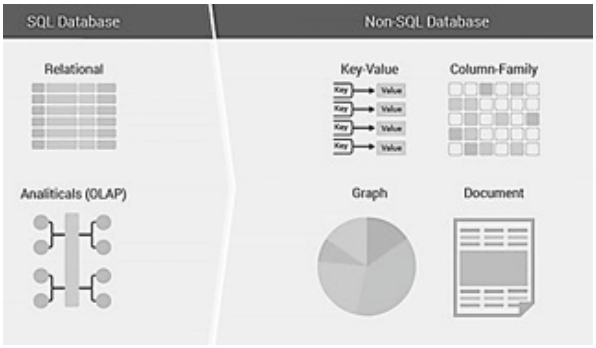

Рисунок 2 – Сравнение подходов к моделям данных в реляционных и нереляционных СУБД

Это позволяет поддерживать высокую разделяемость колонок и, как следствие, горизонтально масштабировать базы данных [12]. Стоит отметить, что *ETL* системе необязательно загружать данные в одну БД. Нереляционные базы данных хорошо подходят для оперативной выгрузки объемов данных, однако поиск конкретных значений лучше удается традиционным решениям БД, поэтому в случае необходимости частой работы с точными значениями данных, рекомендуется использовать реляционные БД. Одним из наиболее частых применений реляционных БД в системах больших данных является управление пользовательскими данными [12]. В таких случаях использования выстраивается гибридная архитектура из нереляционной БД и реляционной, оперирующей малым объемом данных. Такие архитектуры наиболее оперативно производят операции с данными. Сетевые нагрузки также должны быть по возможности оптимизированы. Так как система обработки больших данных должна обладать возможностью встраивания в корпоративные сети, одним из образующих компонентов алгоритма является объединение инфраструктурных объектов под одной выделенной сетью. Внутри сети необходимо использовать балансировку нагрузок на вычислительные мощности машин трансформации данных и машин БД. Возможности использования балансировщиков сете-

вых нагрузок позволяют равномерно распределять нагрузку как на уровне запросов, так и на уровне объектов виртуальной частной сети [14]. В итоге, получаем следующую схему общей инфраструктуры использования алгоритма (рис. 3).

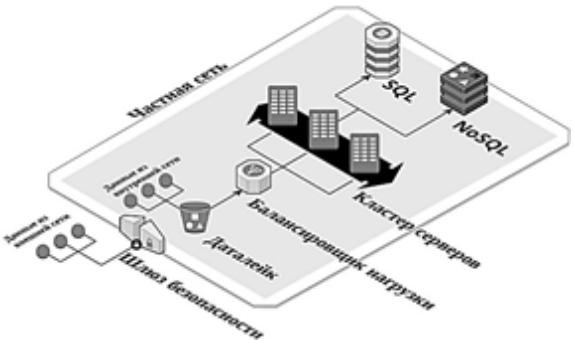

Рисунок 3 – Архитектура алгоритма

1. Данные выгружаются из даталейка, режим работы которого выбирается в зависимости от требований к хранению исходных данных. Для исключения излишней репликации в целях повышения безопасности системы стоит использовать обычный режим работы даталейка. На схеме даталейк представлен корзиной с объектами.

2. Процесс трансформации данных должен происходить на кластере серверов. Работа с контейнером серверов предполагает возможность использования приложений и программ над всем кластером. Над ним должны быть развернуты системы управления кластером машин и распределения потоков между ними. Для конкретной реализации алгоритма необходимо развертывание сервиса *MapReduce*, который может быть заменен на сервисы потоковой обработки данных.

3. Загрузка больших данных должна производиться в *NoSQL* базы данных для обеспечения быстрой работы с информацией остальными компонентами системы. Для создания гибкой системы, способной эффективно работать как с большими данными, так и с точными значениями, рекомендуется использовать гибридные архитектуры БД.

4. Весь процесс обработки должен быть объединён одной частной сетью под управлением балансировщиком нагрузки для равномерной загрузки вычислительных машин. Также необходима работа в частной сети для обеспечения корпоративных требований безопасности.

На каждой стадии процесса обработки данных есть возможности компонентной оптимизации выбранных инфраструктурных объектов. Поэтому для построения наиболее эффективных процессов внутри алгоритма обработки необходимо также выбрать более высокоуровневые компоненты инфраструктуры.

Наиболее популярными подходами к работе с даталейками являются развертывание файловых систем *HDFS* либо *S3*. Первая является частью системы *Hadoop*, вторая является реализацией файловой системы для больших данных от публичного облачного провайдера *Amazon*. Сейчас существуют возможности развертывание озер данных с *S3* файловой системой не только в облаке *Amazon*, но и локально или у других облачных провайдеров. Озеро данных *S3* управляется посредством автоматического программного интерфейса *API*. Помимо наиболее широкого функционала для работы с данными и управлению самим хранилищем, *S3 API* имеет множество возможностей к управлению доступом к данным в озере данных и настройкам безопасности [15]. Чем более полноценно даталейк с исходными данными поддерживает *S3 API*, тем он гибче в настройках, и в том числе в настройках безопасности. Конкретные правила и настройки безопасности для хранилища в рамках применения алгоритма стоит выбирать в зависимости от конкретной реализации алгоритма. *HDFS API* имеет более скромный набор функций. *API* продукта компании *Apache* более ориентировано на работу с данными, методами *PUT* и *GET*, нежели настройки безопасности. Так, у *HDFS API* есть лишь два способа аутентификации: *Hadoop Delegation Token* и *Kerberos SPENEGO*. Из журналирования поддерживается только проверка контрольных сумм [16].

Существуют различные варианты управления кластерами серверов обработки больших данных. Одним из наиболее удобных и эффективных вариантов реализации кластерной обработки больших данных, не требующих потоковых вычислений, является выполнение алгоритма *MapReduce*, разработанного компанией *Google*. Это фреймворк для выполнения процессов и задач, оптимизирующий работу распределенных вычислительных мощностей серверов (нодов) посредством выполнения трех операций: *Map; Shuffle; Reduce* [17].

Алгоритм *MapReduce* успел успешно зарекомендовать себя как фреймворк для параллельной обработки данных на кластерах вычислительных машин, поэтому он отлично подойдет для обработки больших данных. Параллельность и распределение задач делают возможным независимую обработку данных на разных стадиях, при этом не перегружая локальные вычислительные ресурсы конкретной машины. Также это увеличивает доступность системы: при сбое в отдельной воркер-ноде её задача будет передана другой. Отмеченным недостатком является ацикличность работы системы, то есть невозможность обращаться к исходному набору данных дважды на разных стадиях работы. Это ограничивает возможности применения *MapReduce* для машинного обучения, однако таких задач для алгоритма не было обозначено [18].

Внутри машин, используемых в *MapReduce* можно удобно поднимать контейнеры с заранее написанными приложениями. Контейнеризация может осуществляться на различных платформах и позволяет писать различные приложения над виртуальными машинами, а затем *MapReduce* оптимизирует их работу.

Для более оптимальной работы кластеров *MapReduce*, производящих вычисления над общей фай-

ловой системой кластера, существуют надстройки, одна из наиболее удобных, распространенных и применимых для задач обработки больших данных, является *Apache Spark*. Это фреймворк для оптимизации процессов выполнения запросов на серверах, которые при этом производят кэширование в памяти. С помощью *Spark* можно использовать механизм трансформации данных типа «направленный ациклический граф» (*DAG*), что формирует эффективные планы запросов на обработку данных. Разработка приложений, формирующих задачи, с помощью *Spark* также вариативна, так как *Spark* имеет встроенную поддержу *Scala, Python* и *Java,* а также *SQL* запросы к обрабатываемым данным. Такие приложения легко сворачиваются в контейнеры для дальнейшего автоматического развертывания при запуске кластера [19].

Для выбора СУБД необходимо понимать ограничения и требования к БД. Как было выяснено при выборе объектов БД, нам интересны *NoSQL* базы данных, к которым необходимо подобрать управляющий компонент. Одной из наиболее распространенных свободных компонентов управления *NoSQL* является *Druid* [20].

Для выбора инструментов управления сетевыми нагрузками в системах обработки больших данных с первого взгляда нет ничего необычного: есть виртуальная частная сеть, объединяющая машины и хранилища, у каждого есть *IP* адреса, маршрутизацию между которыми осуществляет *NAT*. Однако, в мире больших данных есть специфический программный брокер сообщений *Kafka*, осуществляющий координацию между частями системы. У облачных провайдеров есть специфические сервисы и ресурсы балансировщиков нагрузок, позволяющих распределять нагрузки на вычислительные мощности внутри сети на высоком уровне абстракции.

По полученным результатам выбора компонентов получен стек программных компонентов, представленных на рисунке 4 и позволяющих осуществлять эффективное управление выбранной инфраструктурой.

Рисунок 4 – Архитектура алгоритма на уровне компонент

Выбранные продукты не только эффективны с точки зрения работы с данными, но также и с точки зрения вложений. Они не требуют проприетарных решений, поддерживаются сообществом и имеют широкий набор библиотек.

Как было описано выше, при рассмотрении возможностей алгоритма с точки зрения облачных структур стоит обратить внимание на модели предоставления облачных *IaaS* и *PaaS* сервисов для контроля над инфраструктурой. Эти модели наиболее гибкие для моделирования различных процессов трансформации, а также поддерживают возможности развертывания пользовательских настроек безопасности [7]. При развертывании публичного облака одним из ограничений является невозможность обеспечить критически высокие требования безопасности, но даже несмотря на то, что ресурсы могут физически находиться у провайдера, они могут быть объединены в виртуальную частную сеть, сопряженную с корпоративной. Такие виртуальные сети с облачными ресурсами называются виртуальными частными облаками (*VPC*) [21].

Отдельным неотъемлемым плюсом публичных облачных структур является их надежность. Именно публичные облачные ресурсы позволяют выбрать необходимый уровень доступности датацентра, обеспечены надежностью в 99,(9)% (11 девяток) [15] – для создания подобных ЦОД необходимы огромные капиталовложения.

По вариантам развертывания системы для больших данных, публичные облачные решения также порой более применимы, потому как провайдер может договориться о совместной разработке с *Apache* над *Hadoop* интеграцией, либо самостоятельно сконфигурировать свои продукты так, чтобы они были заточены под легкую интеграцию с *Hadoop*-компонентами [19].

Реализация алгоритма. Сегодня основными игроками на рынке публичных облачных сервисов является тройка: *Amazon Web Services, Microsoft Azure* и *Google Cloud Platform*. Российские компании лишь выходят на этот рынок, их пакет сервисов растет, но пока не является конкурирующими с ними. Крупнейшим облачным провайдером в мире является американская компания *Amazon Web Services*, имеющая более 35% мирового рынка облачных услуг. Являясь лидером отрасли, в портфеле сервисов *Amazon* есть более специализированные сервисы и команды управления инфраструктурой для конкретных задач. Их облачные *IaaS* сервисы обладают способностью автоматического масштабирования при увеличении нагрузок для обеспечения постоянной эталонной производительности инфраструктуры. Немаловажным аргументом при выборе провайдера является доступность автоматического развертывания. В случае с *Amazon*, бесплатно предоставляется возможность развертывания выбранной инфраструктуры посредством скриптов сервиса *AWS CloudFormation.* Он объединяет параметры развертывания облака в один текстовый файл, и затем при запуске скрипта из файлов создается пользовательская облачная среда. При необходимости, алгоритм может поддерживать кросспровайдерную архитектуру.

Основные архитектурные и компонентные эле-

менты уже были описаны и выбраны выше. Необходимо выбрать наиболее подходящие сервисы для каждого из элементов инфраструктуры из списка сервисов *Amazon* так, чтобы они максимально полно поддерживали требования к алгоритму. При реализации исходим из стандартных настроек сервисов, поддерживающих работоспособность алгоритма, но при внедрении алгоритма стоит понимать, что список сервисов может быть расширен за счет использования дополнительных надстроек (резервное копирование, использование нескольких зон доступности и т.п.).

Для исходных хранилищ будет использоваться хранилище *AWS Simple Storage Service (S3)*. Затем данные будут трансформироваться сервисом *AWS Elastic MapReduce (EMR)*, работающим поверх кластера серверов *AWS Elastic Compute Cloud (EC2)*. Это значит, что есть сама единица вычислительной машины *AWS EC2*, но можно заказать сервис *AWS EMR*, который развернет и настроит кластер *EC2* самостоятельно. Дальнейшая загрузка будет производиться в такой же эластичный даталейк *AWS S3*. Реляционная БД может быть представлена сервисом *AWS Relational Database Service (RDS)* под управлением *Druid*. Сетевые нагрузки перенаправляет балансировщик нагрузки *Elastic Load Balancer*. Полная архитектура алгоритма с сервисами *AWS* представлена на рисунке 5.

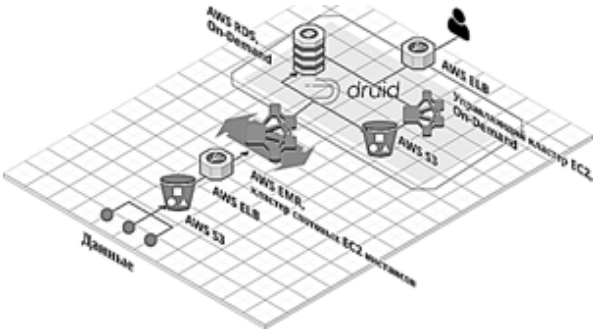

Рисунок 5 – Архитектура алгоритма на уровне компонент внутри облака Amazon Web Services.

Развертывание алгоритма и тестирование. Алгоритм был применен в комплексной системе аналитики больших данных о состоянии ИТ-инфраструктуры. Как видно из рисунка ниже, вся архитектура комплексной системы развернута внутри облака *Amazon Web Services*. Компоненты алгоритма отражены на рисунке 6 в правой части архитектурной схемы и отмечены рамкой.

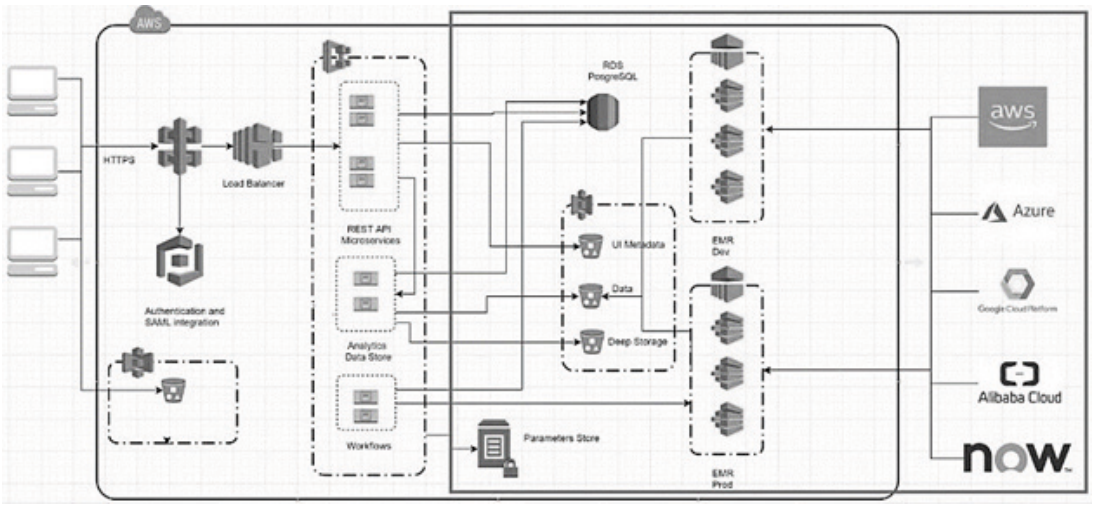

Рисунок 6 – Комплексная архитектура алгоритма.

При внедрении алгоритма развернуто два кластера *EMR* для вариации пользовательских настроек трансформации. Пользователю предлагается потренироваться с настройками трансформации над пробным *EMR* кластером, и убедившись в их корректности, продвигать изменения на основной кластер.

При первоначальном развертывании использовалась консоль *AWS*. После единичного развертывания слепок инфрастуктурных сервисов системы можно использовать посредством *CloudFormation*. Развернуты даталейки *AWS S3* для моделирования источника данных и конечного места загрузки. Затем развернут сервис *AWS EMR,* самостоятельно развернувший кластер *AWS EC2*. При развертывании *EMR* можно сразу указать необходимость журналирования, формат работы, необходимые параметры безопасности внутри машин. Стоит отметить, что помимо настроек безопасности из консоли, могут быть развернуты пользовательские программы и настройки ПО на каждой машине посредством подключения к каждой из них и развертывания настроек самостоятельно либо с помощью контейнеризации. После этого развернуты сервисы *AWS EC2 для Druid*. Затем выставлены права на доступ к данным из источников *S3* для *EC2*.

Доступ по роли к ресурсам осуществляется по трем параметрам: название роли, ключ доступа и секретный ключ. На каждое действие в облаке *Amazon Web Services* (удаление, просмотр данных, управление сервисами и др.) необходимо иметь доступ. При отсутствии доступа подключение и проведение операций над инфраструктурами компонентами невозможно ни через командную строку, ни через

интерфейс консоли *AWS*.

Заключение. В работе предложен алгоритм, позволяющий осуществлять обработку больших объемов данных посредством набора инфраструктурных облачных сервисов. Он позволяет обрабатывать данные из пользовательских защищенных источников, без хранения какой-либо информации после процесса обработки, а также может быть встроен в корпоративные инфокоммуникационные сети, формируя подсистему виртуального частного облака.

При выполнении работы решены следующие задачи:

• произведен анализ предметной области аналитики больших данных и облачных систем;

• произведен выбор объектов и компонентов инфраструктуры алгоритма для разработки;

• спроектирована архитектура процессов алгоритма;

• выбраны публичный облачный провайдер *Amazon Web Services* и сервисы для реализации алгоритма;

• разработан и протестирован алгоритм на инфраструктурных сервисах *Amazon Web Services*.

Предложенная реализация системы является легко масштабируемой облачной распределенной вычислительной системой, что позволяет использовать отдельные инфраструктурные элементы как сервис и при необходимости увеличения мощностей просто докупать их. Таким образом, алгоритм позволяет безопасно обрабатывать большие данные, затем выгружая их в хранилища любой заданной защищенности: как облачные, так и местные. Алгоритм обладает гибкими возможностями трансформации и применения как в публичных, так и в гибридных и кросспровайдерных облачных структурах разных уровней защищенности. Предложенная его реализация позволяет оперативно развертывать облачную инфраструктуру.

СПИСОК ЛИТЕРАТУРЫ:

1. Accenture Big Success with Big Data [Электронный ресурс]. 2014. Режим доступа: https://www.accenture.com/mx-es/insight-big-success-big-data свободный. Яз. англ. (дата обращения: 14.05.2020)

2. Федотов Н.Н. Форензика - компьютерная криминалистика. М.: Юридический мир. – 2007. С. 37-50.

3. David Loshin. ETL (Extract, Transform, Load) // Business Intelligence. – 2nd edition. – Morgan Kaufmann, Burlington, Massachusets, 2012. 400 p.

4. Lynch C. Big data: How do your data grow? // Nature. – 2008. – Т. 455. – №. 7209. – С. 28.

5. Martyshkin, A.I., Salnikov, I.I., Artyushina, E.A. R&D in Collection and Representation of Non-structured Open-Source Data for Use in Decision-Making Systems // Lecture Notes in Electrical Engineering 641 LNEE, PP. 1098-1112.

6. Черняк Л. Большие данные – новая теория и практика // Открытые системы. СУБД. – 2011. – №. 10. – С. 18-25.

7. Bhardwaj S., Jain L., Jain S. Cloud computing: A study of infrastructure as a service (IAAS) //International Journal of engineering and information Technology. – 2010. – Т. 2. – №. 1. – С. 60-63.

8. Herbst N. R., Kounev S., Reussner R. Elasticity in cloud computing: What it is, and what it is not //Proceedings of the 10th International Conference on Autonomic Computing ({ICAC} 13). – 2013. – С. 23-27.

9. Hai R., Geisler S., Quix C. Constance: An intelligent data lake system //Proceedings of the 2016 International Conference on Management of Data. – ACM, 2016. – С. 2097-2100.

10. What is a data lake? // Amazon Web Services [Электронный ресурс]. 2019. Режим доступа: https://aws.amazon.com/ru/big-data/datalakes-and-analytics/what-is-a-data-lake/ свободный. Яз. англ. (дата обращения: 14.05.2020).

11.Чехарин Е. Е. Большие данные: большие проблемы // Перспективы науки и образования. – 2016. – №. 3 (21). – С. 7-11.

12. Moniruzzaman A. B. M., Hossain S. A. NoSQL database: New era of databases for big data analytics-classification, characteristics and comparison // International Journal of Database Theory and Application. Vol. 6, No. 4. 2013. PP. 1 – 14.

13. Что такое NoSQL? // Amazon Web Services [Электронный ресурс]. 2019. Режим доступа: https://aws.amazon.com/ru/nosql/ свободный. Яз. русский (дата обращения: 14.05.2020).

14. Elastic Load Balancing // Amazon Web Services [Электронный ресурс]. 2018. Режим доступа: https://aws.amazon.com/ru/elasticloadbalancing/ свободный. Яз. русский (дата обращения: 14.05.2020).

15. Возможности Amazon S3 //Amazon Web Services [Электронный ресурс]. 2019. Режим доступа: https://aws.amazon.com/ru/s3/features/ свободный. Яз. русский (дата обращения: 14.05.2020).

16. WebHDFS Rest API // Apache [Электронный ресурс]. 2018. Режим доступа: https://hadoop.apache.org/docs/r1.0.4/webhdfs.html свободный. Яз. русский (дата обращения: 14.05.2020).

17. Мартышкин А.И., Сальников И.И., Пащенко Д.В. Этапы сбора и представления больших данных для построения социального профиля человека // Известия Тульского государственного университета. Технические науки. – 2018. – № 9. – С. 617-628.

18. Riesen R., Brightwell R., Maccabe A. B. Differences between distributed and parallel systems // SAND98-2221, Unlimited Release, Printed October. – 1998. – PP. 3-28.

19. Apache Spark в Amazon EMR //Amazon Web Services [Электронный ресурс].2019. Режим доступа: https://aws.amazon.com/ru/emr/features/spark/ свободный. Яз. русский (дата обращения: 14.05.2020).

20. Druid Documentation // Druid [Электронный ресурс]. 2018. Режим доступа: http://druid.io/docs/latest/design/index.html свободный. Яз. англ. (дата обращения: 14.05.2020).

21. Sotomayor B. et al. Virtual infrastructure management in private and hybrid clouds // IEEE Internet computing. – 2009. – Т. 13. – №. 5. – С. 14-22.

Статья публикуется при поддержке гранта РФФИ «Конкурс на лучшие проекты фундаментальных научных исследований» (Грант № 19-07-00516 А).

Статья поступила в редакцию 05.05.2020
Статья принята к публикации 10.06.2020