УДК 004.75

ИССЛЕДОВАНИЕ ПРОИЗВОДИТЕЛЬНОСТИ РАСПРЕДЕЛЕННОЙ ВИРТУАЛЬНОЙ ПРОГРАММНО-КОНФИГУРИРУЕМОЙ ИНФРАСТРУКТУРЫ ДЛЯ ЗАДАЧ ОБРАБОТКИ БОЛЬШИХ ДАННЫХ

Ю. А. Ушаков, к.т.н., доцент кафедры ГКН, Оренбург, Россия;
orcid.org/0000-0002-0474-8919, e-mail: unpk@mail.ru

Виртуальные программно-конфигурируемые виртуальные инфраструктуры на базе контейнеров прочно вошли в состав базовых механизмов облачных вычислений и используются во множестве задач по созданию распределенных масштабируемых отказоустойчивых систем. Но задачи анализа больших данных в большинстве случаев решаются традиционными распределенными кластерами, требующими первоначального развертывания, осторожного обновления и квалифицированного обслуживания. ***Целью работы*** *является исследование эффективности использования программно-конфигурируемых виртуальных инфраструктур на базе контейнеров и методов их быстрого развертывания по облачным принципам для реализации платформ автоматизации распределенной обработки больших данных. Приведены схемы развертывания архитектуры Hadoop, Spark на базе кластеров Docker Swarm и Kubernetes. Также проведена разработка критериев для оценки производительности распределенных вычислений для обработки больших данных и проведено экспериментальное исследование эффективности работы.*

Ключевые слова: *большие данные, распределенные вычисления, анализ данных*

DOI: 10.21667/1995-4565-2021-77-58-67

Введение

Большие данные и работа с ними проникли во все области жизни. Все больше задач управления, логистики, автоматизации, принятия решений, маркетинга, научных исследований требуют анализа больших данных. Для быстрого и точного анализа данных в распределенных многоуровневых системах хранения и обработки больших данных требуются новые подходы, большие вычислительные ресурсы и объемы хранилищ. Традиционные подходы на основе Hadoop решений и механизмов распределенной обработки Map-Reduce уже не удовлетворяют многим современным задачам по скорости и оперативности выполнения. Кроме того, традиционные подходы требуют, в большинстве случаев, выделенных вычислительных ресурсов, стабильную высокопроизводительную сеть (вплоть до специализированных кластерных решений). Решения на базе Hadoop и сопутствующих продуктах не обладают простотой развертывания и управления, требуют значительного уровня знаний и опыта для их использования в собственной инфраструктуре [1].

Использование облачных провайдеров для больших данных в виде SaaS, PaaS, Serverless имеет преимущество в виде отсутствия предварительного развертывания и настройки, а также избавляет от забот по поддержанию инфраструктуры, но имеет высокую стоимость и привязывает процессы к конкретным провайдерам [2].

Альтернативой содержания собственной инфраструктуры и использованию готовых облачных сервисов является развертывание собственных сервисов на основе IaaS решений. С одной стороны, это снимает необходимость в создании собственной высокопроизводительной инфраструктуры и дает возможность оперативно масштабировать решения, добавлять специализированные узлы с графическими и тензорными вычислителями, твердотельными сверхскоростными дисками NVMe, а также запрашивать сегменты с 40 – 100 Гб/с сетью или низколатентные Infiniband/Fiberchannel соединения. С другой стороны, это не избавляет от первичного развертывания и настройки всего необходимого программного обеспече-

ния, синхронизации узлов, предоставления интересов взаимодействия внешним источникам данных, исследователям, сервисам [3]. При работе с вычислительными инфраструктурами с внешним управлением необходимо также учитывать вопросы безопасности хранения и передачи данных, а также сетевые ограничения на связность узлов и наличие брандмауэров.

Работа посвящена исследованию разработанных ранее алгоритмов автоматизации развертывания облачных инфраструктур [4] для создания платформы автоматизации распределенных вычислений для обработки больших данных [5] в условиях ограничений облачных платформ. Также проведена разработка критериев для оценки эффективности работы платформы автоматизации распределенных вычислений для обработки больших данных и проведены результаты экспериментального исследования эффективности работы.

Средства и методы развертывания приложений

В инфраструктурах виртуальных машин, арендуемых по принципу IaaS, возможности внедрения механизмов удаленного управления на этапе заказа и включения ограничены SSH ключами. Большинство провайдеров, предоставляющие API для автоматизации этого процесса (Vmware vCloud Director API, Amazon API, Cloudstack API, OpenStack API, ISPsystem API) обеспечивают только SSH доступ, хотя отдельные системы или провайдеры могут предоставить образы с предустановленными системами управления конфигурацией (Salt, Puppet, Ansible, Chief, SaltStorm). Каждая из этих систем позволяет автоматизировать множество действий по установке и настройке программ, но создание работающего кластера требует больших усилий по написанию предварительных скриптов [6].

Также существуют подходы, основанные на использовании менеджеров кластеров, таких как Apache Mesos, TrinityX, Linux-HA Heartbeat [7]. Они требуют при первичной настройке даже больше знаний и опыта, чем системы управления конфигурацией. В исходном виде в IaaS их использование не оправдано задачей, хотя и возможно.

В то же время существуют системы кластерной контейнеризации, которые позволяют развертывать произвольное количество узлов кластера, виртуальные наложенные сети, распределенные хранилища в полуавтоматическом режиме сразу после установки. К тому же, системы контейнеризации не исключают использование средств управления, а дополняют их, а также могут запускать их на узлах в автоматическом режиме. Самым популярным на сегодняшний день средством кластерной контейнеризации является Kubernetes (K8s), также встречаются промышленные использования коммерческой версии K8n RedHat OpenShift, открытой системы Rancher, Apache Mesos Marathon, а также второй по популярности системы кластеризации Docker Swarm.

Выбор системы создания и управления кластером должен начинаться с определения задач и требований по отказоустойчивости, управляемости и совместимости с программным обеспечением. Рассмотрим основное используемое программное обеспечение для систем обработки больших данных.

1. Система импорта данных в Data Lake. В этой области представлен самый широкий спектр сервисов, таких как коннекторы (connectors) и средства импорта для сервисов syslog, SNMP, NetFow, аналитики веб-сайтов, событий, аудита безопасности, «сырого» трафика, фото, видео, текста, документов и т.д. Большинство сервисов импортирования – это коннекторы к выбранной системе организации Data Lake или к потоковым обработчикам.

2. Система потоковой обработки неструктурированных и слабоструктурированных данных. Включают как открытые инструменты (Java +Hadoop, Apache Spark, Apache Flink, Apache Drill, Apache Kafka Streams, Apache Storm, Apache Samza, Logstash), так и облачные инструменты для выноса первичной обработки на внешних провайдеров, например Azure.

3. Система организации хранилища DataLake. Традиционные подходы включают в себя распределенные хранилища на основе Hadoop (pure HDFS, HBase, Hive, Asterix), OpenStack Swift, Cassandra, Presto, Elasticsearch, Yandex ClickHouse, MariaDB ColumnStore, MongoDB.

Также используются внешние хранилища на основе Amazon S3, Amazon Redshift, Amazon FireHouse, Google BigData, Azure DataLake и прочие.

4. Системы анализа и визуализации циклических данных (системы мониторинга), такие как Prometheus+Grafana, InfluxDB, Elasticsearch+Kibana, R Studio.

5. Программные системы для обработки больших данных включают в себя как встроенные языки и утилиты в каждый из используемых инструментов (например, Python или встроенные noSQL диалекты), так и внешние системы, такие как нейросетевые фреймворки (tensorflow, keras, pytorch, caffe), фреймворки машинного обучения, фреймворки работы с данными (numpy, pandas, scikit, mathploit).

Также нельзя забывать о необходимости физического хранения данных для выбранных DataLake инструментов, способов доступа к этим данным по сети, способам обеспечения репликации, доступа к данным из контейнеров.

Системы распределенного хранения и потоковой обработки

Системы хранения данных на узлах IaaS для доступа к ним из контейнеров довольно ограничены. Большинство систем постоянного хранения Kubernetes и Docker Swarm требуют общего внешнего сетевого хранилища. Например, Kubernetes Persistent Volume по умолчанию умеют работать с NFS, iSCSI/FC, loud-provider-specific storage, а также с сетевыми файловыми системами типа Cepp, GlusterFS, Cinder и подобными. Особенностью контейнеров является то, что контейнер может запуститься на произвольном узле и доступ к хранилищу должен быть организован через общий ресурс или адрес, доступный всем узлам. Хранение данных на тех же локальных узлах, которые используются для вычислений, не подходит из-за неопределенности места запуска контейнера. Запуск Docker-версии хранилища типа GlusterFS/Ceph сопряжен с накладными расходами на синхронизацию, к тому же они не синхронизированы с масштабированием узлов, поэтому данные могут быть потеряны. Почти все системы организации хранения данных имеют встроенную репликацию и устойчивы к потере части узлов. Кроме того, для снижения сетевых задержек данные могут размещаться сразу на тех узлах, где они будут более востребованы.

Приведем пример размещение в контейнерах системы хранения HDFS (рисунок 1). Каждый узел может совмещать различные роли, если позволяет нагрузка. Для размещения Hadoop NameNode выбирается узел, к которому возможен доступ снаружи. Использование узла возможно и для запущенных внутренних клиентов Hadoop, и для внешних Hadoop сервисов. При взаимодействии с внешними потребителями напрямую для Hadoop используются проброшенные порты HDFS и WebHDFS, а также режим сбора адресов кластера по DNS. В этом случае все внутренние DNS записи Docker кластера необходимо вести в выбранном публичном поддомене, а на публичном DNS сервере все «A» записи этого поддомена перевести на публичный адрес узла с NameNode [8].

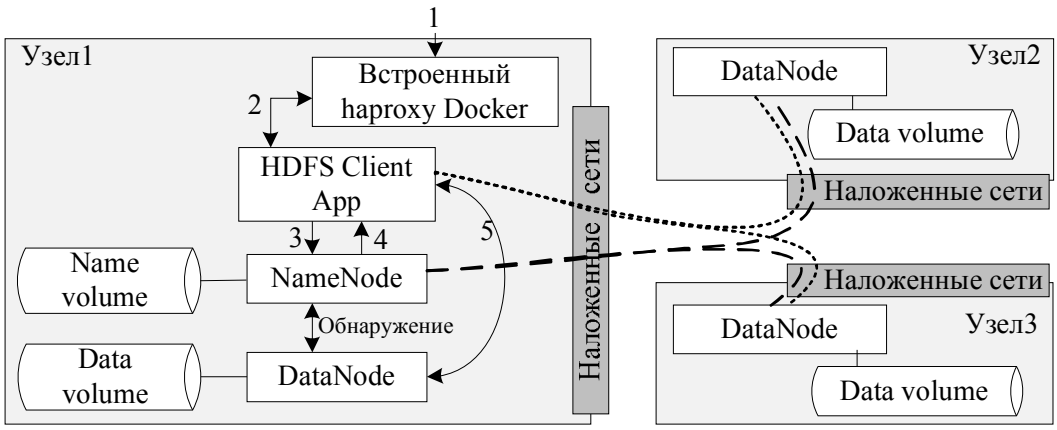

Рисунок 1 – Схема работы кластера HDFS в Docker Swarm cluster
Figure 1 – Scheme of HDFS cluster operation in Docker Swarm cluster

На рисунке 1 цифрами обозначены следующие шаги процесса чтения данных:

1. В приложение выполняется запрос, сопровождаемый чтением данных.
2. Выполняется обратное проксирование запроса в контейнер.
3. Выполняется запрос адресов DataNode с искомой информацией.
4. Производится возврат списка адресов.
5. Выполняется обращение к требуемому адресу и чтение информации.

Работа данной архитектуры через наложенную сеть кластера Swarm или K8n прозрачна, передача данных происходит в зашифрованном виде посредством настроенных механизмов IPSeC.

Архитектура хранения Elasticsearch почти не отличатся от Hadoop, но количество MasterNode должно быть не менее трех, связь производится по внутренним DNS Docker. Архитектура хранения Cassandra реализуется еще проще, здесь каждый контейнер независим, но для настройки кластера необходимо сообщить сервисам об адресах хотя бы нескольких запущенных сервисов. Также может быть использован групповой DNS (разрешение имени контейнера при репликации на несколько узлов возвращает все адреса всех запущенных контейнеров).

Масштабирование на все узлы Swarm кластера выполняется с помощью настройки «deploy: mode: global» или для конкретного количества узлов «deploy: mode: replicated replicas: 2», для Kubernetes используется соответственно «kind: DaemonSet» и «spec: replicas: 2».

Количество узлов в кластере может быть большое, и в случае, если не все узлы необходимо использовать или необходимо выделить конкретные узлы под работу конкретных служб (например, с графическим ускорителем), требуется ограничить конкретный кластер приложений. Для этого на виртуальной машине при добавлении в кластер Swarm или Kubernetes нужно становить одну или несколько текстовых меток в виде «имя=значение», например «accelerator=nvidia-tesla-p100». Затем при запуске контейнера указать ограничения на узлы – в Swarm это «constraints: -"node.label.accelerator==nvidia-tesla-p100"», в Kubernetes «nodeSelector: accelerator: nvidia-tesla-p100» . Это позволяет гибко размещать сервисы только на тех узлах, где это требуется.

Поскольку анализ данных часто использует не только HDFS, но и саму инфраструктуру Hadoop, требуется добавить контейнеры: один ResourceManager, один HistoryServer и по одному NodeManager на каждую виртуальную машину. Сам Hadoop внутри управляется менеджером YARN, при правильном задании переменных в контейнерах сборка кластера произойдет автоматически.

Для использования внешних систем обработки, например Apache Spark, требуется запустить один Master и несколько Slave. При этом их связь будет также осуществляться по заданному имени сервиса Master через внутренний DNS Docker.

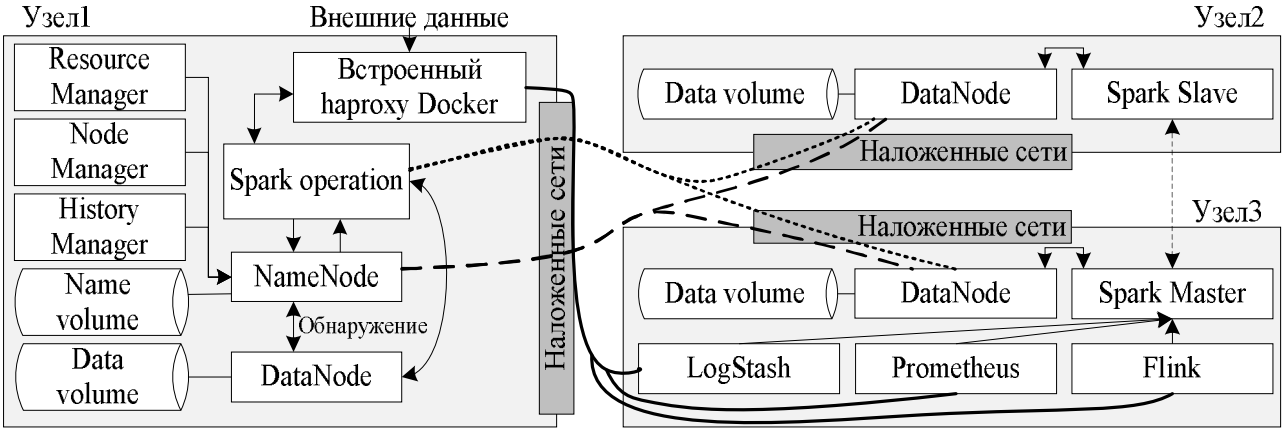

Рисунок 2 – Схема работы кластера Apache Spark + Hadoop в Docker Swarm cluster
Figure 2 – Scheme of Apache Spark + Hadoop cluster in Docker Swarm cluster

Как видно из рисунка 2, структурно отличий в добавлении Spark в систему нет. Поскольку используется Docker, нет несовместимости версий библиотек, проблем с установкой дополнительных программ – все берется из готовых контейнеров с требуемыми версиями программного обеспечения.

Реализация платформы

Алгоритм управления платформой обработки больших данных должен управлять наличием и количеством запущенного программного обеспечения, а также следить за тем, чтобы была связность всего программного обеспечения друг с другом. Каждый новый сервис обработки должен быть заранее протестирован, и для него должен быть создан шаблон развертывания (например, шаблон Docker compose) и шаблон или скрипт тестирования работоспособности. Для управления платформой разработан API на основе реализации OpenAPI. Поскольку почти все действия развертывания выполняются в Docker, а для взаимодействия с сервисами применяется REST API, графическая часть выполнена на JavaScript в виде веб-приложения, работающего с API, а также в виде скриптов командной строки.

Запуск каждого сервиса имеет одинаковую последовательность действий:

1. Прочитать шаблон, выделить необходимые для задания вручную переменные, найти для них соответствие в описании шаблона (тип, допустимые значения, логика совместимости).

2. Сформировать форму для заполнения пользователем.

3. По результатам заполнения формы сформировать JSON шаблон и послать запрос в кластер Docker для запуска сервиса.

4. С помощью скриптов тестирования проверить работоспособность сервиса и сообщить клиенту входные адреса и порты сервиса.

Для основы Hadoop, Spark, Flink, Hive сервисов взят репозиторий Big Data Europe project [9], из файлов сервисов которых были сделаны шаблоны соответствующих сервисов.

Для обеспечения автоматизации действий, связанных с развертыванием компонентов, а также для ограничения и изоляции различных компонентов в многопользовательских средах необходимо проработать вопрос авторизации и разделения области видимости. Самой большой проблемой использования таких инструментов, как Jupyter, Anaconda IE, Zeppelin, является отсутствие инструментов изоляции пользователей и кастомизации персональных подключений к внешним системам. Существующие дополнения, такие как JupiterHub, Anaconda Team, Zeppelin Shiro, позволяют решить только проблему авторизации, отчасти проблему изоляции по пользовательским аккаунтам. Но в случае если платформу используют несколько организаций с различными настройками или даже видами систем хранения и кластерного управления, эти инструменты не позволяют использовать их в таких режимах. В коммерческих облачных платформах (например, в [10]) используется подход запуска отдельного контейнера для пользователя и проксирование в общий интерфейс.

Для реализации более простого, чем ручная настройка, стека Docker для платформы создан небольшой портал, реализующий функции автоматического заказа и контроля элементов. Схема портала показана на рисунке 3.

Основная проблема гранулярного запуска сервисов – необходимость точного соответствия всех номеров версий Java, Python, PySpark, Scala, Spark, Hadoop и остальных компонентов и клиентских библиотек.

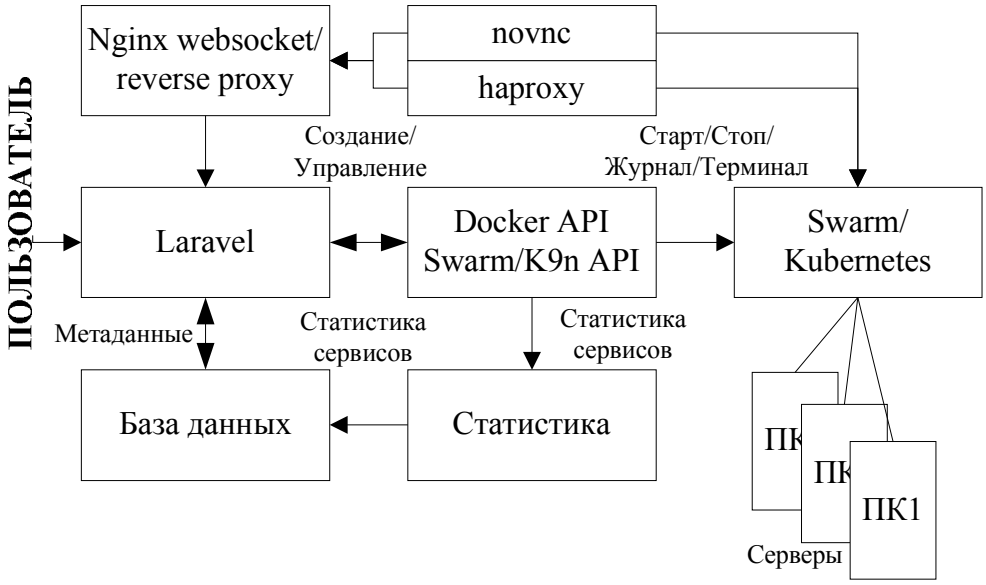

Рисунок 3 – Схема веб-портала управления
Figure 3 – Scheme of web-based management portal

При заказе нужной конфигурации происходит выбор нужных образов. Для python образов запрашивается requirements.txt и происходит сборка образа, который помещается в локальный репозиторий. Затем по шаблону формируется файл Docker-compose и запускается стек нужных сервисов. Вычислительные узлы подключаются к кластеру swarm/kubernetes любыми доступными способами.

Для того чтобы не давать прямой доступ к портам веб-служб и консолям, в novnc помещается токен доступа для контейнеров, к которым необходим доступ через консоль. Для веб-служб (например, мониторинга Spark) и блокнотов формируется правило проксирования порта через laravel-websocket с авторизацией по пользовательскому токену на haproxy кластера docker-swarm. Таким образом, все контейнеры пользователя будут доступны для доступа через веб-интерфейс с предварительной авторизацией в основной системе. Авторизация самих сервисов не включалась.

Для подсистемы хранения создаются статические разделы (volumes), размещенные в файловой системе физического хоста. Для постоянного хранения данных с привязкой к сервису активируется sshfs на хосте управления, и на нем размещается хранилище личных файлов блокнотов с прямым доступом из веб-интерфейса.

Эксперимент

Для эксперимента выбран набор из 5 серверов Intel Xeon E5, на которые установлен Debian 10, docker, docker-swarm. Все узлы заведены в кластер swarm. Один из узлов имеет GPU, для размещения используется параметр generic-resource "gpu=1".

Эксперимент проводился с целью оценки скорости запуска сервисов и оценки размера накладных расходов производительности кластеров при таком подходе.

В таблице 1 показаны результаты измерения запуска сервисов. Каждый запуск проводился 5 раз, измерялось среднее значение времени запуска от передачи команды в API до ответа от сервиса через открытый порт и потребление памяти в простое. Разделы Hadoop и Spark (data volumes) были очищены. Курсивом показано время запуска всего стека до режима функционирования и сумма потребления памяти на одну копию каждого сервиса.

Таблица 1 – Данные о запуске сервисов
Table 1 – Services start-up data

Стек	Время начала, с	Использование памяти в режиме ожидания, Мб
Hadoop 3.0– 1 копия	*166*	*2170*
NodeManager	49	497
ResourceManager	45	698
DataNode	40	219
NameNode	126	313
HistoryServer	12	365
Hue	3	78
Spark– 1 копия	*22*	*755*
Master	6	326
Worker	22	429
Dask– 1 копия	5	366
Master	3	68
Worker	5	134
Scheduler	2	164
Jupiter– 1 копия	*12*	*68*
Zeppelin+Spark client+Hadoop client – 1 копия	*22*	*460*

Как видно из эксперимента, разворачивание собственной инфраструктуры возможно, но требует ресурсов, и запуск полного стека может занимать несколько минут. Отдельно до 20 % процессорного времени тратилось на шифрование в сторону клиентов, которые работали с Jupiter и Zeppelin для обеспечения безопасности веб-сокета.

Отдельно стоит проблема производительности распределенных вычислений в таких средах. Сравним скорость чтения и записи для среды на контейнерах, виртуальной машине KVM и при традиционной установке на одном и том же оборудовании (рисунок 4).

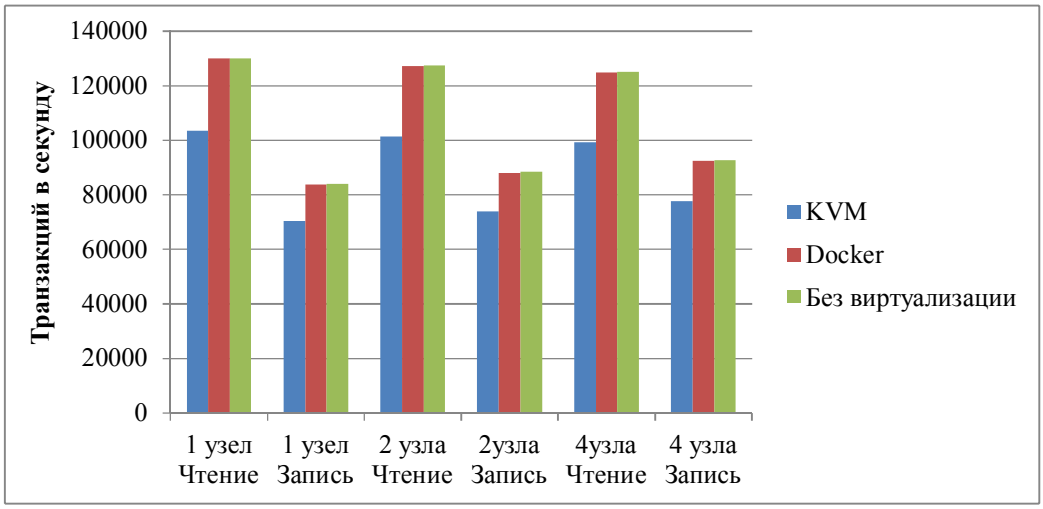

Рисунок 4 – Сравнение пропускной способности Docker-based Hadoop с KVM и решением без виртуализации
Figure 4 – Compare throughput on Docker-based Hadoop vs KVM and bare-metal installation

Как видно из графика, Docker при работе с данными практически не уступает решению без виртуализации. Оценим задержки при взаимодействии между узлами в режиме записи данных (рисунок 5).

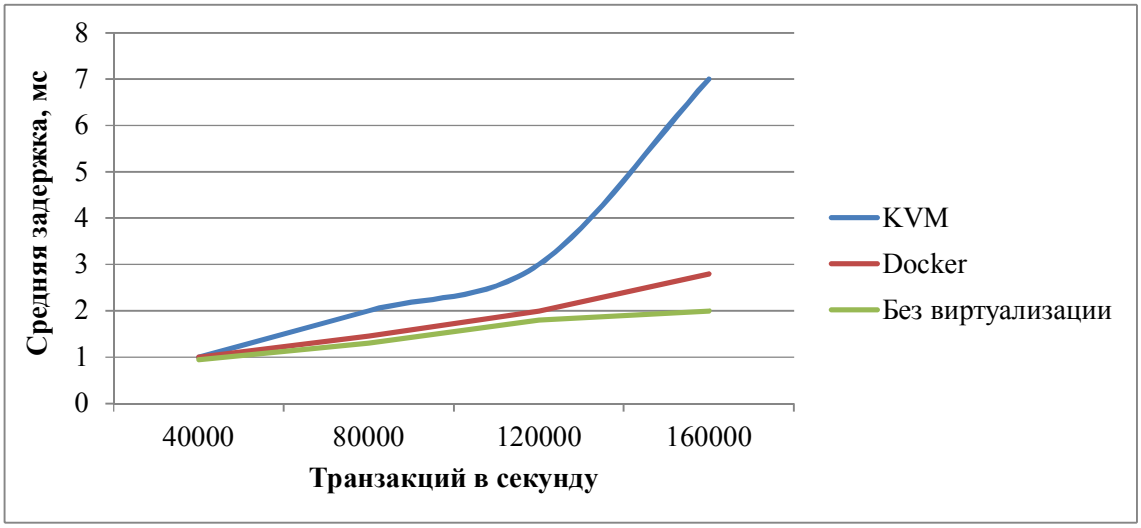

Рисунок 5 – Сравнение задержек Docker-based Hadoop с KVM и без виртуализации
Figure 5 – Compare latency on Docker-based Hadoop vs KVM and bare-metal installation

Из графика видно, что решение без виртуализации на высоких нагрузках имеет преимущество, но подход на основе Docker существенно лучше виртуализации по задержкам (более чем в два раза).

Заключение

В результате исследования разработан прототип платформы обработки больших данных для совместной работы на выделенном оборудовании. Такой подход позволил упростить запуск инструментов хранения, анализа и разработки в области больших данных для образовательных учреждений, студентов, энтузиастов. В результате реализации платформы выявлена жизнеспособность идеи, необходимость реализации внешних инструментов управления и обеспечения безопасности. Также проведена оценка производительности по сравнению с традиционными и виртуализированными средами, которая показала преимущество контейнеризации перед виртуализацией по производительности чтения до 20 %, по записи до 15 %, снижение задержки при высокой нагрузке более чем в два раза.

Исследование выполнено при финансовой поддержке РФФИ (*проект №20-57-53019*).

Библиографический список

1. **Ложкин О.** Методы выполнения запросов к хранилищу данных в Hadoop и Spark // Молодой ученый. 2017. № 14. С. 34-41.

2. **Odun-Ayo I., Ananya M., Agono F., Goddy-Worlu R.** Cloud Computing Architecture: A Critical Analysis. 2018 18th International Conference on Computational Science and Applications (ICCSA). 2018, pp. 1-7. DOI: 10.1109/ICCSA.2018.8439638.

3. **Bolodurina I. P., Parfenov D. I.** Development and research of the autonomous system for providing security and quality of service for multi-cloud platform. 7th Mediterranean Conference on Embedded Computing (MECO). 2018, pp. 1-4. DOI:10.1109/MECO.2018.8406038.

4. **Bolodurina I., Parfenov D., Torchin V., Legashev L.** Development and Investigation of Multi-Cloud Platform Network Security Algorithms Based on the Technology of Virtualization Network Functions. International Scientific and Technical Conference Modern Computer Network Technologies (MoNeTeC). 2018, pp. 1-7. DOI: 10.1109/MoNeTeC.2018.8572059.

5. **Ушаков Ю. А., Бахарева Н. Ф., Полежаев П. Н., Шухман А. Е.** Модель распределенной гетерогенной мультиоблачной вычислительной системы и платформы автоматизации распределенных вычислений для обработки больших данных // Университетский комплекс как региональный центр

образования, науки и культуры: материалы Всерос. науч.-метод. конф. (с междунар. участием). 2019. С. 1980-1991.

6. **Saraswat M., Tripathi R. C.** Cloud Computing: Analysis of Top 5 CSPs in SaaS, PaaS and IaaS Platforms. 2020 9th International Conference System Modeling and Advancement in Research Trends (SMART). 2020, pp. 300-305, DOI: 10.1109/SMART50582.2020.9337157.

7. **Bahareva N., Ushakov Y., Ushakova M., Parfenov D., Legashev L., Bolodurina I.** Researching a Distributed Computing Automation Platform for Big Data Processing. 2020 International Conference Engineering and Telecommunication (En&T). 2020, pp.1-5. DOI: 10.1109/EnT50437.2020.9431254.

8. **Bolodurina I. P., Ushakov Y. A., Parfenov D. I., Cui J.** Cloud processing of security events for VANET distributed sensors. IOP Conference Series: Materials Science and Engineering. 2021, vol. 1069, no 1, pp. 012007.

9. Репозиторий Big Data Europe project [Электронный ресурс]. URL: https://github.com/big-data-europe/docker-hadoop (дата обращения 28.06.2021).

10. Research highlights the opportunities that hybrid and multi-cloud strategies present to the modern enterprise [Электронный ресурс]. URL: https://www.cloudera.com/campaign/economics-of-hybrid-and-multi-cloud.html (дата обращения 28.06.2021).

UDC 004.75

RESEARCH OF DISTRIBUTED VIRTUAL SOFTWARE-DEFINED INFRASTRUCTURE PERFORMANCE FOR BIG DATA PROCESSING TASKS

Y. A. Ushakov, Ph.D. (Tech.), associate Professor, OSU, Orenburg, Russia;
orcid.org/0000-0002-0474-8919, e-mail: unpk@mail.ru

Container-based virtual software-defined virtual infrastructures have become an integral part of the underlying cloud computing engine and are used in a variety of distributed, scalable, resilient systems. But big data analytics tasks are mostly handled by traditional distributed clusters that require initial deployment, careful upgrades, and skilled maintenance. ***The aim of the work*** *is to study the effectiveness of using software-defined virtual infrastructures based on containers and methods for their rapid deployment according to cloud principles for the implementation of automation platforms for distributed processing of big data. The schemes of Hadoop and Spark architecture deployment based on Docker Swarm and Kubernetes clusters are given. The development of criteria for evaluating the performance of distributed computing for processing big data was also carried out and an experimental study of the work efficiency was carried out.*

Key words: *Big Data, Distributed Infrastructure, Data Analyze.*

DOI: 10.21667/1995-4565-2021-77-58-67

References

1. **Lozhkin O.** Metody vypolnenija zaprosov k hranilishhu dannyh v Hadoop i Spark. *Molodoj uchenyj*. 2017, no14, pp. 34-41 (in Russian).

2. **Odun-Ayo I., Ananya M., Agono F., Goddy-Worlu R.** Cloud Computing Architecture: A Critical Analysis. *2018 18th International Conference on Computational Science and Applications* (*ICCSA*). 2018, pp. 1-7. DOI: 10.1109/ICCSA.2018.8439638.

3. **Bolodurina I. P., Parfenov D. I.** Development and research of the autonomous system for providing security and quality of service for multi-cloud platform. *7th Mediterranean Conference on Embedded Computing* (*MECO*). 2018, pp.1-4. DOI:10.1109/MECO.2018.8406038.

4. **Bolodurina I., Parfenov D., Torchin V., Legashev L.** Development and Investigation of Multi-Cloud Platform Network Security Algorithms Based on the Technology of Virtualization Network Functions. *International Scientific and Technical Conference Modern Computer Network Technologies* (*MoNeTeC*). 2018, pp. 1-7. DOI: 10.1109/MoNeTeC.2018.8572059.

5. **Ushakov Y. A., Bakhareva N. F., Polezhaev P. N., Shuhman A. E.** Model' raspredelennoj geterogennoj mul'tioblachnoj vychislitel'noj sistemy i platformy avtomatizacii raspredelennyh vychislenij dlja

obrabotki bol'shih dannyh. *Universitetskij kompleks kak regional'nyj centr obrazovanija, nauki i kul'tury: materialy Vseros. nauch.-metod. konf.* (*s mezhdunar. uchastiem*). 2019, pp. 1980-1991 (in Russian).

6. **Saraswat M., Tripathi R.C.** Cloud Computing: Analysis of Top 5 CSPs in SaaS, PaaS and IaaS Platforms. *2020 9th International Conference System Modeling and Advancement in Research Trends* (*SMART*). 2020, pp.300-305, DOI: 10.1109/SMART50582.2020.9337157.

7. **Bahareva N., Ushakov Y., Ushakova M., Parfenov D., Legashev L., Bolodurina I.** Researching a Distributed Computing Automation Platform for Big Data Processing. *2020 International Conference Engineering and Telecommunication* (*En&T*). 2020, pp.1-5. DOI: 10.1109/EnT50437.2020.9431254.

8. **Bolodurina I. P., Ushakov Y. A., Parfenov D. I., Cui J.** Cloud processing of security events for VANET distributed sensors. *IOP Conference Series: Materials Science and Engineering*. 2021, vol. 1069, no 1, pp. 012007.

9. Repozitorij Big Data Europe project [Jelektronnyj resurs]. URL: https://github.com/big-data-europe/docker-hadoop (data obrashhenija 28.06.2021).

10. Research highlights the opportunities that hybrid and multi-cloud strategies present to the modern enterprise [Jelektronnyj resurs]. URL: https://www.cloudera.com/campaign/economics-of-hybrid-and-multi-cloud.html (data obrashhenija 28.06.2021).