УНИВЕРСИТЕТ ИТМО
НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ноябрь–декабрь 2019 Том 19 № 6 ISSN 2226-1494 http://ntv.itmo.ru/
SCIENTIFIC AND TECHNICAL JOURNAL OF INFORMATION TECHNOLOGIES, MECHANICS AND OPTCS
November–December 2019 Vol. 19 No 6 ISSN 2226-1494 http://ntv.itmo.ru/en/
НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

УДК 004.75 doi: 10.17586/2226-1494-2019-19-6-1086-1093

РЕАЛИЗАЦИЯ ПРОТОКОЛА ОБМЕНА ДАННЫМИ МЕЖДУ ПРОГРАММНЫМИ АГЕНТАМИ В ОБЛАЧНОЙ ИНФРАСТРУКТУРЕ В ГЕОГРАФИЧЕСКИ РАСПРЕДЕЛЕННЫХ ЦЕНТРАХ ОБРАБОТКИ ДАННЫХ

Н.Ю. Самохин[a], А.А. Орешкин[b], А.С. Супрун[a]

[a] Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация
[b] НИЦ «Курчатовский Институт» – ПИЯФ, Гатчина, 188300, Российская Федерация
Адрес для переписки: samon@vuztc.ru

Информация о статье
Поступила в редакцию 16.07.19, принята к печати 03.10.19
Язык статьи — русский

Ссылка для цитирования: Самохин Н.Ю., Орешкин А.А., Супрун А.С. Реализация протокола обмена данными между программными агентами в облачной инфраструктуре в географически распределенных центрах обработки данных // Научно-технический вестник информационных технологий, механики и оптики. 2019. Т. 19. № 6. С. 1086–1093. doi: 10.17586/2226-1494-2019-19-6-1086-1093

Аннотация
Представлена облачная система для географически распределенных центров обработки данных. В основу разработки положен принцип мультиагентности структуры (микросервисы). Для связи агентов между собой предложен специально разработанный протокол взаимодействия, работающий в асинхронном режиме. Асинхронность системы взаимодействия агентов достигается за счет применения специально разработанного алгоритма. Программа, реализующая предложенный алгоритм, написана на языке программирования Python. В данном решении используются реляционные базы данных и системы очередей. Реляционная база данных применяется для хранения запросов и ответов от агентов. Для обмена YAML-сообщениями с идентификаторами этих запросов и ответов применяется брокер сообщений. Апробация разработанного программного обеспечения произведена на макете масштабируемого географически распределенного центра обработки и хранения данных. Получено оригинальное техническое решение, успешно прошедшее контрольно-тестовые испытания и внедренное в действующую облачную инфраструктуру. Показаны особенности применения системы очередей RabbitMQ и системы управления баз данных PostgreSQL в кластерном режиме с шифрованием трафика. Использование разработанной модели представляется перспективным в условиях работы с высоконагруженными распределенными системами.

Ключевые слова
облако, распределенный, центр обработки данных, очереди сообщений, RabbitMQ, PostgreSQL

Благодарности
Исследования выполнены при финансовой поддержке Министерства науки и высшего образования Российской Федерации (Договор № 03.G25.31.0229).

doi: 10.17586/2226-1494-2019-19-6-1086-1093

IMPLEMENTATION OF AGENT INTERACTION PROTOCOL WITHIN CLOUD INFRASTRUCTURE IN GEOGRAPHICALLY DISTRIBUTED DATA CENTERS

N.Yu. Samokhin[a], A.A. Oreshkin[b], A.S. Suprun[a]

[a] ITMO University, Saint Petersburg, 197101, Russian Federation
[b] Petersburg Nuclear Physics Institute named by B.P. Konstantinov of National Research Centre «Kurchatov Institute» (NRC «Kurchatov Institute» — PNPI) , Gatchina, 188300, Russian Federation
Corresponding author: samon@vuztc.ru

Article info
Received 16.07.19, accepted 03.10.19
Article in Russian

For citation: Samokhin N.Yu., Oreshkin A.A., Suprun A.S. Implementation of agent interaction protocol within cloud infrastructure in geographically distributed data centers. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2019, vol. 19, no. 6, pp. 1086–1093 (in Russian). doi: 10.17586/2226-1494-2019-19-6-1086-1093

Abstract
A cloud system for geographically distributed data centers is presented. An approach is based on the principle of multi-agent structure (microservices). A special interaction protocol was developed for agents' communication operating in asynchronous

mode. The asynchronous behavior of the agent interaction system is achieved through the use of a specially developed algorithm. The program that implements the proposed algorithm is written in the Python programming language. This solution uses relational databases and queuing systems. A relational database stores requests and responses from the agents. A message broker is necessary for exchanging YAML messages with identifiers of these requests and responses. The developed software was tested on a prototype of a scalable geographically distributed data center. An original technical solution was obtained that successfully passed a series of tests and was implemented within existing cloud infrastructure. Features of applying the RabbitMQ queuing system and PostgreSQL database management system in cluster mode with traffic encryption are indicated. The use of the developed model appears to be promising oriented for highly loaded distributed systems.

Keywords
cloud, distributed, data center, message broker, RabbitMQ, PostgreSQL

Acknowledgements
The research was carried out with the financial support of the Ministry of Education and Science of the Russian Federation (Contract No. 03.G25.31.0229).

Введение

При создании интегрированной системы управления облачной инфраструктурой географически распределенных центров обработки данных (ГРЦОД) [1] возникла необходимость в разработке специального протокола взаимодействия между агентами, являющимися ее составными частями. Публичные облачные инфраструктуры (облака), например, такие как Microsoft Azure, Amazon AWS, Яндекс Облако, используют в своих архитектурных решениях различные протоколы: для коммуникаций пользователей с облаком, для взаимодействия компонентов облака между собой. Для коммуникаций пользователей со службами облака часто используется REST API для HTTP [2]. REST API, будучи архитектурным стилем взаимодействия компонентов распределенного приложения в сети, в основном используется для синхронной обработки, когда клиент находится на связи с сервером, пока не получит от сервера ответ. В современной облачной инфраструктуре приложения разделяются на небольшие независимые элементы (микросервисы или агенты), которые проще разрабатывать, развертывать и обслуживать. Хорошей практикой считается использовать для взаимодействия и координации между ними очереди сообщений, которые работают в асинхронном режиме [3].

Согласно исследованиям [4, 5], с помощью системы очередей необходимо передавать малый объем информации, либо обмениваться не самой информацией, а сообщениями со своего рода ссылками на нее, так как число байт, которое можно передать через очередь сообщений, конечно.

Публичные облака используют проприетарные системы очередей сообщений. В облаке Microsoft Azure используется различные варианты системы очередей сообщений для своих сервисов: Azure Storage Queues, Azure Service Bus Queues, Azure Service Bus Topics & Subscriptions. В облаке Amazon AWS — система очередей сообщений Amazon Simple Queue Service. В Яндекс Облаке — система очередей сообщений Yandex Message Queue. Из свободно распространяемых систем очередей сообщений отметим следующие: Apache Active MQ, ZeroMq, RabbitMQ, ApacheQpid, Kafka, EagleMQ, IronMQ.

Отметим, что RabbitMQ обладает следующими основными преимуществами перед аналогами: проста в использовании, поддерживает протокол AMQP и кластеризацию (что является важным моментом в распределенных системах). Кроме того, данная система имеет удобный и гибкий API, при желании можно настроить web-интерфейс [6, 7].

С целью оптимизации объемов передаваемой между агентами информации в [8] предложена процедура, в которой основная информация записывается в специальную базу данных (БД), а затем в очередь сообщений отправляется «ссылка» на нужную запись в БД. Эта модель может быть использована и для управления агентами. Для БД могут быть использованы реляционные или NoSQL системы управления базами данных (СУБД). Представляется логичным использование СУБД именно первого типа ввиду необходимости четкого структурирования хранимых данных (запросов и ответов с их метаданными). Наиболее популярными решениями считаются программные пакеты MySQL и PostgreSQL. В ходе разработки предпочтение отдается второму варианту ввиду его многочисленных преимуществ, в особенности в области масштабируемых распределенных систем [9].

Разработка специального асинхронного протокола с одновременным использованием системы очередей и БД обусловлена острой необходимостью обеспечить высокую доступность связи между имеющимися микросервисами. Так, разработчикам системы [1] стандартная модель использования одного экземпляра системы очередей видится недостаточно отказоустойчивой, исходя из чего было принято решение внедрить дополнительный подуровень хранения данных о запросах и ответах, а также обеспечить отказоустойчивость обоих подуровней с помощью организации системы виртуальных кластеров.

В связи с вышеизложенным основными задачами данной работы являются:

1) выбор системы очередей сообщений из свободно распространяемых;
2) выбор БД для процедуры обмена информацией между агентами;
3) обеспечение отказоустойчивости системы очередей сообщений и БД;
4) разработка протокола обмена данными между агентами.

Агенты. Архитектурные решения

Облачная инфраструктура на базе ГРЦОД требует создания системы интегрированного управления (СИУ). СИУ, представленная в работе [1], создана путем разбиения ее на логические функциональные блоки (микросервисы), каждый из которых выполняет свои определенные функции. Функциональный блок реализуется в виде программного демона (агента) в изолированной операционной среде, в отдельной виртуальной машине (ВМ) или контейнере. Обмен данными между такими агентами осуществляется асинхронно через очереди сообщений и реляционную базу данных – это позволяет избежать состояний ожидания ответа, как при синхронном общении, а также добавляет устойчивости протоколу обмена. Агент ожидает прихода запроса в свою входную очередь и, получив запрос, обрабатывает его, результат посылает обратно в очередь агента, пославшего запрос. Запуск нескольких экземпляров одного агента на ЦОДе позволяет повысить отказоустойчивость и пропускную способность агента. На каждом ЦОДе устанавливаются свои экземпляры агентов, работающие независимо от агентов на других ЦОДах.

Использование системы агентов имеет ряд преимуществ, в том числе:

— универсальность: в рамках данной работы для программной реализации агента был использован язык высокого уровня Python, однако может использоваться любой другой язык программирования;

— отказоустойчивость системы: в ситуации, когда один из агентов по различным причинам более недоступен, система продолжает функционировать, так как остальные агенты продолжают работать и исполнять свои функции, в том числе экземпляры-клоны того же агента;

— гибкость: существенно упрощается принцип работы системы горизонтального масштабирования [10, 11].

Логическая структура СИУ и взаимосвязь агентов схематически показаны на рис. 1. В [1] дано описание функций, выполняемых каждым агентом.

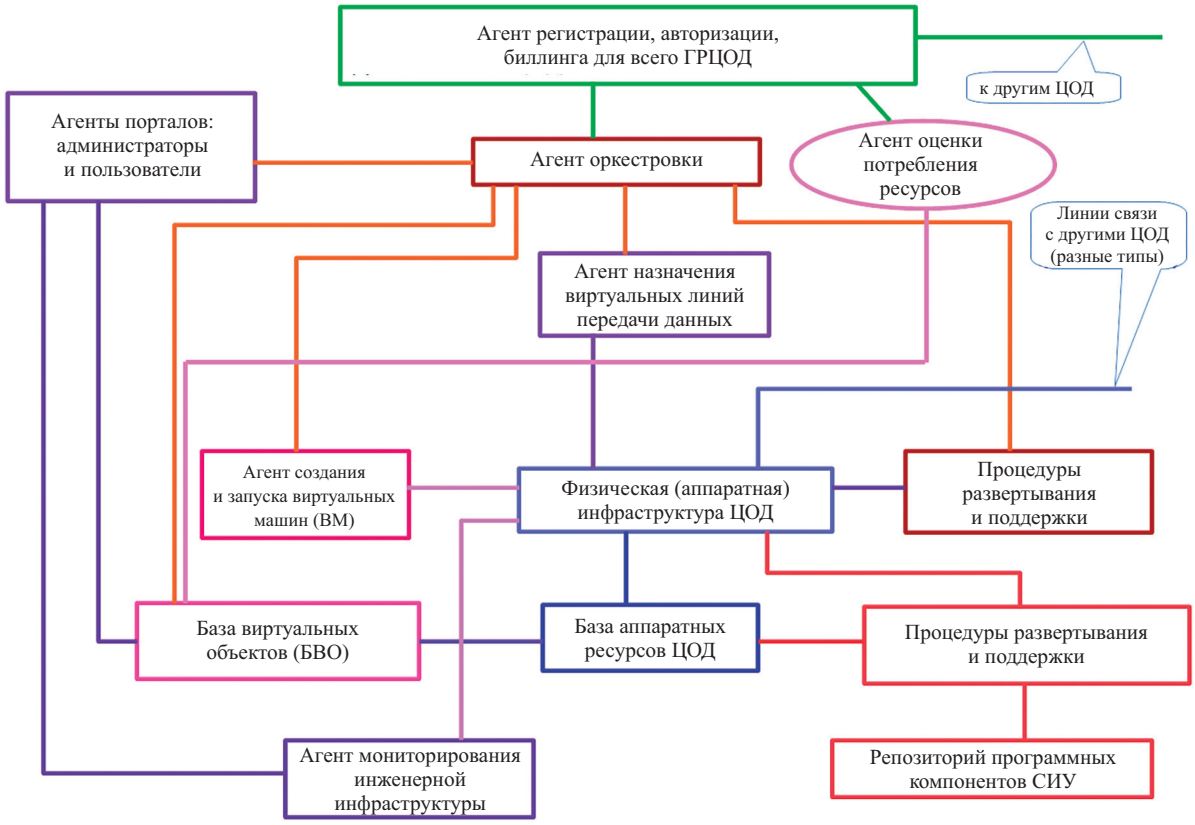

Рис. 1. Логическая схема взаимосвязи агентов в рамках одного центра обработки данных

Очереди сообщений позволяют различным частям системы асинхронно взаимодействовать и обрабатывать операции и включают в себя упрощенный буфер, используемый для временного хранения сообщений, и конечные точки, которые позволяют программным компонентам подключаться к очереди для отправки и получения сообщений. Чтобы отправить сообщение, компонент, называемый отправителем, добавляет сообщение в очередь. Сообщение хранится в очереди, пока другой компонент, называемый получателем, не извлечет это сообщение и не выполнит с ним некую операцию. Использование очередей повышает горизонтальную масштабируемость и надежность: всегда можно увеличить поток сообщений в очередь и добавить больше получателей – обработчиков сообщений, при этом отказ получателей никак не сказывается на работе отправителей. Очереди сглаживают пики нагрузок: они исполняют роль буфера

для получателей. Если для мгновенной обработки всех поступающих сообщений текущих мощностей получателей недостаточно, помещенные в очередь сообщения будут обработаны позже, когда нагрузка уменьшится. Буферизация полезна для сервисов с нестабильной нагрузкой, где не нужна моментальная обработка входящих сообщений [12].

Ввиду ограничения на объем информации, способного быть переданным через систему очередей, используется реляционная база данных для хранения этой информации, через систему очередей же передаются только краткие сообщения, содержащие в себе идентификатор записи в БД с нужным запросом. Таким образом, появляется возможность сформировать сложный запрос, который определенный агент должен выполнить, записать его в БД и через систему очередей сообщить упомянутому агенту, что существует некий запрос, который ждет, что его обработают. После обработки запроса агентом похожим образом можно записать результат в БД и послать уведомление в очередь сообщений.

В качестве БД используется PostgreSQL версии 11, ввиду ее популярности, масштабируемости (включая работу в распределенных системах), а также высокой скорости выполнения операций [9].

Для обеспечения высокой доступности и повышения пропускной способности RabbitMQ создан кластер. Узлы кластера располагаются на различных физических серверах, все узлы кластера равнозначны и имеют доступ к общим очередям. Если выйдет из строя один узел, например, из-за поломки аппаратного сервера, то другие узлы продолжат принимать сообщения из очередей. Так как все узлы кластера имеют доступ ко всем очередям, то обработка сообщений выполняется всеми узлами одновременно, что повышает пропускную способность брокера и обеспечивает балансировку нагрузки [13]. Подключение к кластеру производится по защищенному каналу с использованием SSL.

Для обеспечения высокой доступности базы данных создан кластер по репликационной схеме мультимастер с использованием программного пакета Bucardo, состоящий из нескольких узлов, расположенных на разных физических серверах. В случае выхода из строя одного из узлов по разным причинам, например, поломка физического сервера, кластер будет продолжать работать [14]. Подключение к кластеру производится по защищенному каналу с использованием SSL.

В рамках данной работы предложено на каждом ЦОДе инсталлировать свой экземпляр агента оркестровки и свои экземпляры исполнительных агентов, что обеспечит независимость управления ЦОДом от управления другими ЦОДами.

Важную роль в управлении ЦОДом играет протокол взаимодействия агентов между собой. Он должен обеспечить:

— надежность передачи данных между агентами;

— в случае программного или аппаратного сбоя предоставить детальную информацию о точке сбоя и причинах сбоя;

— независимость функционирования агентов друг от друга;

— удобную структуру передаваемых данных.

Основываясь на приведенных требованиях, был разработан специальный протокол обмена данными между агентами.

Протокол взаимодействия программных агентов

Разработанный протокол обмена между агентами использует два потока данных:

1) краткие сообщения, содержащие информацию об источнике запроса и ссылки на базу данных, посылаемые в очередь сообщений типа AMQP;

2) подробное описание запроса, которое записывается в соответствующую базу данных запросов.

Принимающий агент считывает краткий запрос из своей входной очереди, считывает оттуда уникальный идентификатор и по нему находит в специальной базе данных подробное описание технического запроса, например, на выделение ресурсов. Такая схема позволяет функционировать агентам независимо: если один из агентов аварийно завершил работу или перестал отвечать по любым причинам (программная ошибка, сбой аппаратного сервера, на котором выполняется агент), то в агенте, который послал запрос, сработает таймаут и механизм уведомления для администратора. На всех агентах ведутся журналы (логи) о выполненных действиях агентов. На основе логов и информации из базы данных администратор сможет понять, в какой точке и по какой причине возникла проблема.

Важное значение в протоколе обмена данными между агентами имеет сам формат данных. От него зависит удобство обработки и возможность масштабирования данных. В базе данных для запросов и ответов созданы таблицы запросов и ответов на запросы. Каждому запросу присваивается уникальный идентификатор формата uuid. По нему отслеживается весь путь запроса, начиная от посылки его соответствующему агенту и заканчивая получением ответа на запрос. Для удобства ответу присваивается идентификатор исходного запроса.

Модель разработанного протокола схематически показана на рис. 2.

Тело запроса (и ответа) в БД имеет формат JSON. Тело запроса (и ответа) в очереди сообщений имеет формат YAML. Оба формата позволяют достаточно легко извлекать из себя информацию [15].

Запросы YAML, передающиеся с помощью системы сообщений, содержат в себе такую информацию, как:

— кем послан запрос.
— когда послан запрос,
— ссылка на сам JSON-запрос в БД.

Запросы, хранящиеся в БД, содержат в себе такую информацию, как:

— уникальный идентификатор запроса (uuid),
— дата формирования запроса (UTC),
— тип запроса,
— тело запроса или ответа на него (JSON),
— этап обработки запроса (статус).

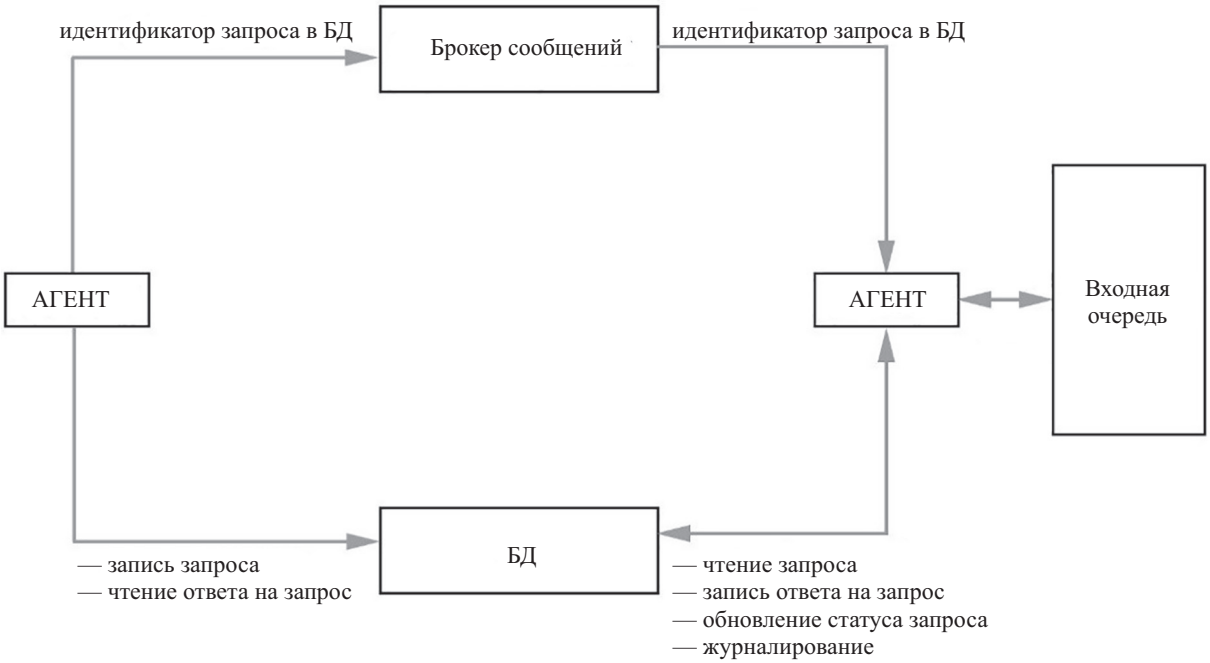

Рис. 2. Логическая модель разработанного протокола

Разработанный специальный механизм обмена данными между агентами использует два асинхронных потока — YAML-запросы через систему очередей AMQP и JSON-запросы через реляционную СУБД.

Экспериментальные исследования

Экспериментальные исследования представленной разработки выполнены на макете ГРЦОД, описание которого дано в [1]. Программа исследований содержит более 40 испытаний, которые выполнялись на основе разработанных контрольно-тестовых задач. Как наиболее важные и связанные с упомянутой темой задачи, можно отметить и взять для рассмотрения следующие:

1) Тест № 1: Проверка создания виртуального хранилища данных (ВХД) на основе подписки по запросу;

2) Тест № 2: Проверка создания ВМ1 и ВМ2 в OpenStack1 (ЦОД 1) и OpenStack2 (ЦОД 2) на основе подписки по запросу;

3) Тест № 3: Проверка функции мониторинга виртуальной инфраструктуры: виртуальных каналов передачи данных, ВХД, виртуальных процессоров;

4) Тест № 4: Проверка удаления ВХД на основе подписки по запросу.

Если рассматривать принцип работы системы на примере первых двух вышеуказанных контрольно-тестовых задач, то процесс начинается с агентов авторизации. Пользователь (клиент) регистрируется с помощью агента регистрации и авторизации. После регистрации клиент должен запросить услуги (ресурсы), к примеру в рамках понятия подписки, если речь идет о модели «ресурсы как услуга». В качестве услуги может быть ВМ, ВХД, виртуальный канал передачи данных (ВКПД) между ВМ. Услуги могут сопровождаться требованиями по качеству обслуживания. Все услуги, входящие в подписку, оформляются в виде одного запроса. В первую очередь запрос на создание услуг (ресурсов) посылается агенту оркестровки. В запросе указывается уникальный идентификатор (uuid) ЦОД, в который нужно перенаправить запрос. Агент оркестровки анализирует запрос, разбивает весь запрос на одиночные запросы, каждый из которых

соответствует одному ресурсу, затем перенаправляет последовательно каждый одиночный запрос соответствующему исполнительному агенту. Для создания ВМ запрос перенаправляется агенту управления ВМ, для создания ВХД – агенту управления ВХД, для создания виртуального канала передачи – агенту управления ВКПД. В случае успешного создания виртуального ресурса исполнительный агент возвращает техническую информацию для доступа к созданному виртуальному ресурсу. В случае неуспешного выполнения запроса исполнительный агент возвращает информацию о причине сбоя [16, 17].

На рис. 3 представлена выдержка из системного журнала одного из агентов, кратко иллюстрирующая получение и обработку поступившего через очередь сообщений запроса (в данном случае на получение данных по утилизации ресурсов, согласно вышеуказанному тесту № 3).

```
Jun 17 04:38:52 monitoring env: message
Jun 17 04:38:52 monitoring env: {host_from: samara-rabbitmq.samara.ru, program_from: {agent: monitoring, module: monitoring-sping},$
Jun 17 04:38:52 monitoring env: call
Jun 17 04:38:52 monitoring env: sping
Jun 17 04:38:52 monitoring env: {host_from: monitoring, program_from: {agent: ma, module: ma_sping}, reply_uuid: ac025104-f6e1-4854$
Jun 17 04:39:54 monitoring env: message
Jun 17 04:39:54 monitoring env: {host_from: samara-ksuv.samara.ru, program_from: {agent: ksuv-multi-dc, module: ksuv-resend-own-dc},
Jun 17 04:39:54 monitoring env: request_uuid: 16498cdb-25c7-49e0-913c-62154413d82b, sping: false, timestamp: '2019-06-17
Jun 17 04:39:54 monitoring env: 04:39:35+00'}
Jun 17 04:39:54 monitoring env: call
Jun 17 04:39:54 monitoring env: Connected to the database to update...
Jun 17 04:39:54 monitoring env: Connected to the database to select...
Jun 17 04:39:54 monitoring env: Collecting data...
Jun 17 04:39:54 monitoring env: {"in": "19892087", "out": "15243542"}
Jun 17 04:39:54 monitoring env: Connected to the database to insert...
Jun 17 04:39:54 monitoring env: Connected to the database to update...
Jun 17 04:39:54 monitoring env: message
Jun 17 04:39:54 monitoring env: {host_from: samara-rabbitmq.samara.ru, program_from: {agent: monitoring, module: monitoring-sping},$
Jun 17 04:39:54 monitoring env: call
Jun 17 04:39:54 monitoring env: sping
Jun 17 04:39:54 monitoring env: {host_from: monitoring, program_from: {agent: ma, module: ma_sping}, reply_uuid: db34b229-f782-44c3$
```

Рис. 3. Выдержка из системного журнала агента мониторинга — получение YAML-сообщения, подключение к базе данных для считывания исходного запроса, обновления статуса запроса, запись ответа на запрос, отправка ответа в ответную очередь.

Для более подробного исследования обратимся к вышеуказанному тесту № 4, в рамках которого происходит удаление существующего виртуального объекта.

Центральный агент управления виртуальными ресурсами подключен к одной из так называемых входных очередей отказоустойчивого кластера RabbitMQ по защищенному каналу связи и находится в состоянии ожидания. Когда в нужную входную очередь приходит сообщение, агент считывает идентификатор запроса (рис. 4, *а*). Имея нужный идентификатор запроса, агент может получить содержимое этого запроса, по защищенному каналу связи обратившись к нужной сущности отказоустойчивого кластера базы данных (рис. 4, *б*).

а

```
2019-04-2509:53:57+00 ksuv-main: Received request body {'host_from':
'128.1.1.240', 'program_from': 'ASR', 'request_uuid'
: 'd020bde5-6718-48a7-80dc-e9fdb6d798a1',
  'sping': !!bool 'false', 'timestamp': '20190425-095357'}
```

б

```
2019-04-2509:53:57+00 ksuv-main: Selecting request from ksuv  request table
'query_schema.requests'
2019-04-2509:53:57+00  ksuv-main: ('4dad6b4a-3bfb-433d-905f-3743698963c4',
{u'client_uuid': u'35843250-05a7-40a0-a217-a733c7bc9c82', u'request':
u'transaction', u'provider_uuid': u'b01db499-4491-424d-9282-854dd60b969e',
u'dc_uuid': u'DC1', u'subscription_uuid': u'af1d817a-a5b9-408a-892d-
71dcde1441d8', u'data': [{u'gw': {u'type': u'nextcloud'}, u'request': u'vhd
delete', u'resource_uuid': u'c5c0d98e-ab5d-4e52-ba27-62e5391593d4'}]}, '0215df6e-
f41c-4bb9-b47a-aae2a5f72e0f')

2019-04-2509:53:57+00 ksuv-main: Updating  reply_queue in  request table
2019-04-2509:53:57+00 ksuv-main: Updated reply_queue in request table= asr

2019-04-2509:53:57+00 ksuv-transaction: Transaction request: single request to
UHD: '[u'vhd', u'delete']'
2019-04-2509:53:57+00 ksuv-transaction: Connecting to database -->
'172.17.20.55','5432','query_db'
2019-04-2509:53:57+00 ksuv-transaction: Inserting request into request table
2019-04-2509:53:57+00 ksuv-transaction: Inserted request_uuid = 79d75dc5-817a-
4bfd-93b7-8f25c85f2611
```

Рис. 4. Выдержка из системного журнала агента управления виртуальными ресурсами: получение YAML-сообщения из очереди (*а*), поиск содержимого запроса в базе данных по его идентификатору (*б*)

Проанализировав полученное содержимое запроса, агент в состоянии определить, какому вспомогательному агенту должен быть перенаправлен данный запрос. В рамках данного теста таким вспомогательным агентом выступает агент управления виртуальным хранилищем, так как запрос содержит необходимые параметры для операций именно с этой сущностью. Таким образом, агент управления виртуальными ресурсами обращается к кластеру RabbitMQ и пересылает запрос в специальную входную очередь агента управления виртуальным хранилищем (рис. 5, *а*). Последний после обработки и выполнения запроса обязан отправить во входную очередь центрального агента краткий результат выполнения исходного запроса. Независимо от успешности выполнения упомянутого запроса, все содержимое ответа записывается агентами в БД в виде записи с идентификатором исходного запроса (рис. 5, *б*).

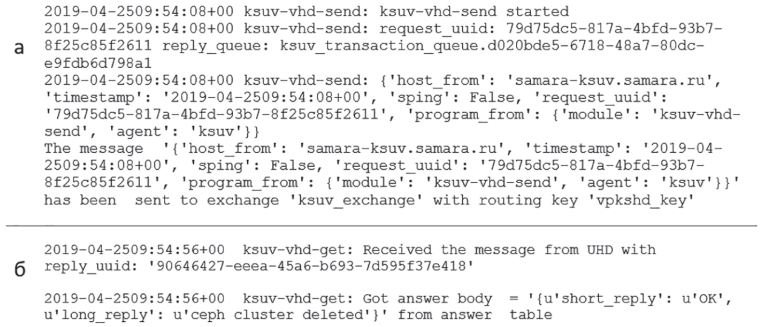

Рис. 5. Выдержка из системного журнала агента управления виртуальными ресурсами: пересылка исходного запроса агенту управления виртуальным хранилищем (*а*), получение ответа от агента управления виртуальным хранилищем и запись результата в базу данных (*б*)

Результаты исследований показали, что применение предложенной модели более чем оправдано в рамках использования мультиагентных систем в облачной инфраструктуре. Разделение канала коммуникации между агентами на два асинхронных потока представляется удобным, в особенности в ситуациях передачи большого объема информации внутри запроса — например, массива параметров создаваемого виртуального ресурса. Кроме того, подобная модель обеспечивает более легкую отладку и более удобный контроль всего процесса (включая журналирование) [16, 17].

Заключение

В данной статье предлагается комплекс решений по реализации протокола обмена данными между программными агентами в облачной инфраструктуре географически распределенных центров обработки данных. Данные решения позволяют существенно расширить функционал систем управления распределенных центров обработки данных в рамках контроля над внутренними процессами.

Приведены основные принципы работы системы с использованием системы очереди сообщений и реляционной базы данных, описан специально разработанный протокол взаимодействия агентов.

Экспериментальные исследования показали работоспособность принятых технических решений.

Предложенная модель имеет ряд преимуществ, в том числе гибкость, универсальность, отказоустойчивость, что потенциально создает перспективы для использования подобного набора подходов и методик в ряде высоконагруженных систем, в особенности географически распределенных.

Получено оригинальное техническое решение, успешно прошедшее контрольно-тестовые испытания и внедренное в действующую облачную инфраструктуру Университета ИТМО.

Литература

1. Хоружников С.Э., Шевель А.Е. Система управления масштабируемым географически распределенным центром обработки данных // Научно-технический вестник информационных технологий, механики и оптики. 2019. Т. 19. № 5. С. 931–938. doi: 10.17586/2226-1494-2019-19-5-931-938
2. De Benedictis A., Rak M., Turtur M., Villano U. REST-Based SLA management for cloud applications // Proc. IEEE 24th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2015). 2015. P. 93–98. doi: 10.1109/WETICE.2015.36
3. Benchara F.Z., Youssfi M., Bouattane O., Ouajji H. A new efficient distributed computing middleware based on cloud micro-services for HPC // Proc. 5th International Conference on Multimedia Computing and Systems (ICMCS). 2016. P. 354–359. doi: 10.1109/ICMCS.2016.7905644
4. Esposito C., Palmieri F., Choo K.-K. R. Cloud message queueing and notification: challenges and opportunities // IEEE Cloud Computing. 2018. V. 5. N 2. P. 11–16. doi: 10.1109/MCC.2018.022171662
5. Salah K., Sheltami T.R. Performance modeling of cloud apps using message queueing as a service (MaaS) // Proc. 20th Conference on Innovations in Clouds, Internet and Networks (ICIN). 2017. P. 65–71. doi: 10.1109/ICIN.2017.7899251
6. Ionescu V.M. The analysis of the performance of RabbitMQ and ActiveMQ // Proc. 14th RoEduNet International Conference - Networking in Education and Research (RoEduNet NER). 2015. P. 132–137. doi: 10.1109/RoEduNet.2015.7311982

References

1. Khoruzhnikov S.E., Shevel A.Ye. Management system for scalable geographically distributed data center. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2019, vol. 19, no. 5, pp. 931–938. (in Russian). doi: 10.17586/2226-1494-2019-19-5-931-938
2. De Benedictis A., Rak M., Turtur M., Villano U. REST-Based SLA management for cloud applications. *Proc. IEEE 24th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2015)*, 2015, pp. 93–98. doi: 10.1109/WETICE.2015.36
3. Benchara F.Z., Youssfi M., Bouattane O., Ouajji H. A new efficient distributed computing middleware based on cloud micro-services for HPC. *Proc. 5th International Conference on Multimedia Computing and Systems (ICMCS)*, 2016, pp. 354–359. doi: 10.1109/ICMCS.2016.7905644
4. Esposito C., Palmieri F., Choo K.-K. R. Cloud message queueing and notification: challenges and opportunities. *IEEE Cloud Computing*, 2018, vol. 5, no. 2, pp. 11–16. doi: 10.1109/MCC.2018.022171662
5. Salah K., Sheltami T.R. Performance modeling of cloud apps using message queueing as a service (MaaS). *Proc. 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, 2017, pp. 65–71. doi: 10.1109/ICIN.2017.7899251
6. Ionescu V.M. The analysis of the performance of RabbitMQ and ActiveMQ. *Proc. 14th RoEduNet International Conference — Networking in Education and Research (RoEduNet NER)*, 2015, pp. 132–137. doi: 10.1109/RoEduNet.2015.7311982

7. Hong X.J., Sik Yang H., Kim Y.H. Performance analysis of RESTful API and RabbitMQ for microservice web application // Proc. 9th International Conference on Information and Communication Technology Convergence (ICTC). 2018. P. 257–259. doi: 10.1109/ICTC.2018.8539409
8. Wang J., Bai X., Li L., Ji Z., Ma H. A Model-based framework for cloud API testing // Proc. 41st IEEE Annual Computer Software and Applications Conference (COMPSAC). 2017. P. 60–65. doi: 10.1109/COMPSAC.2017.24
9. Momjian B. PostgreSQL: Introduction and concepts. AddisonWesley, 2002. 490 p.
10. Chen Y., Paxson V., Katz R. What's new about cloud computing security?. Technical Report UCB/EECS-2010-5. Berkeley, 2010.
11. Ali-Eldin A.H. Capacity scaling for elastic compute clouds. Dissertation. Umea University, Sweden, 2013. 90 p.
12. Vinoski S. Advanced message queuing protocol // IEEE Internet Computing. 2006. V. 10. N 6. P. 87–89. doi: 10.1109/MIC.2006.116
13. Rostanski M., Grochla K., Seman A. Evaluation of highly available and fault-tolerant middleware clustered architectures using RabbitMQ // Proc. Federated Conference on Computer Science and Information Systems. 2014. V. 2. P. 879–884. doi: 10.15439/2014F48
14. Budrean S., Li Y., Desai B.C. High availability solutions for transactional database systems // Proc. 7th International Database Engineering and Applications Symposium (IDEAS). 2003. P. 347–355. doi: 10.1109/IDEAS.2003.1214952
15. Afanasev M.Ya., Fedosov Yu.V., Krylova A.A., Shorokhov S.A. Performance evaluation of the message queue protocols to transfer binary JSON in a distributed CNC system // Proc. 15th IEEE International Conference on Industrial Informatics (INDIN). 2017. P. 357–362. doi: 10.1109/INDIN.2017.8104798
16. Samokhin N.Yu., Khoruzhnikov S.E., Trubnikova V.M., Akhmedzyanova R.R., Bulykina A.B. Information on utilization of data center resources with message broker implementation // Научно-технический вестник информационных технологий, механики и оптики. 2018. Т. 18. № 5. С. 858–862. doi: 10.17586/2226-1494-2018-18-5-858-862
17. Fedchenkov P.V., Khoruzhnikov S.E., Samokhin N.Y., Shevel A.Y. The designing of cloud infrastructure consisting of geographically distributed data centers // CEUR Workshop Proceedings. 2018. V. 2267. P. 32–36.

7. Hong X.J., Sik Yang H., Kim Y.H. Performance analysis of RESTful API and RabbitMQ for microservice web application. *Proc. 9th International Conference on Information and Communication Technology Convergence (ICTC)*, 2018, pp. 257–259. doi: 10.1109/ICTC.2018.8539409
8. Wang J., Bai X., Li L., Ji Z., Ma H. A Model-based framework for cloud API testing. *Proc. 41st IEEE Annual Computer Software and Applications Conference (COMPSAC)*, 2017, pp. 60–65. doi: 10.1109/COMPSAC.2017.24
9. Momjian B. *PostgreSQL: Introduction and concepts*. AddisonWesley, 2002, 490 p.
10. Chen Y., Paxson V., Katz R. *What's new about cloud computing security?. Technical Report UCB/EECS-2010-5*. Berkeley, 2010.
11. Ali-Eldin A.H. *Capacity Scaling for Elastic Compute Clouds*. Dissertation. Umea University, Sweden, 2013, 90 p.
12. Vinoski S. Advanced message queuing protocol. *IEEE Internet Computing*, 2006, vol. 10, no. 6, pp. 87–89. doi: 10.1109/MIC.2006.116
13. Rostanski M., Grochla K., Seman A. Evaluation of highly available and fault-tolerant middleware clustered architectures using RabbitMQ. *Proc. Federated Conference on Computer Science and Information Systems*, 2014, vol. 2, pp. 879–884. doi: 10.15439/2014F48
14. Budrean S., Li Y., Desai B.C. High availability solutions for transactional database systems. *Proc. 7th International Database Engineering and Applications Symposium (IDEAS)*, 2003, pp. 347–355. doi: 10.1109/IDEAS.2003.1214952
15. Afanasev M.Ya., Fedosov Yu.V., Krylova A.A., Shorokhov S.A. Performance evaluation of the message queue protocols to transfer binary JSON in a distributed CNC system. *Proc. 15th IEEE International Conference on Industrial Informatics (INDIN)*, 2017, pp. 357–362. doi: 10.1109/INDIN.2017.8104798
16. Samokhin N.Yu., Khoruzhnikov S.E., Trubnikova V.M., Akhmedzyanova R.R., Bulykina A.B. Information on utilization of data center resources with message broker implementation. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2018, vol. 18, no. 5, pp. 858–862. doi: 10.17586/2226-1494-2018-18-5-858-862
17. Fedchenkov P.V., Khoruzhnikov S.E., Samokhin N.Y., Shevel A.Y. The designing of cloud infrastructure consisting of geographically distributed data centers. *CEUR Workshop Proceedings*, 2018, vol. 2267, pp. 32–36.

Авторы

Самохин Никита Юрьевич — инженер, Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация, Scopus ID: 57196066158, ORCID ID: 0000-0002-4634-8809, samon@vuztc.ru
Орешкин Анатолий Алексеевич — ведущий инженер, НИЦ «Курчатовский Институт» — ПИЯФ, Гатчина, 188300, Российская Федерация, ORCID ID: 0000-0002-2314-0917, anatoly.oreshkin@gmail.com

Супрун Антон Сергеевич — ассистент, Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация, ORCID ID: 0000-0001-8429-9115, asuprun@list.ru

Authors

Nikita Yu. Samokhin — Engineer, ITMO University, Saint Petersburg, 197101, Russian Federation, Scopus ID: 57196066158, ORCID ID: 0000-0002-4634-8809, samon@vuztc.ru
Anatoly A. Oreshkin — Leading Engineer, Petersburg Nuclear Physics Institute named by B.P. Konstantinov of National Research Centre «Kurchatov Institute» (NRC «Kurchatov Institute» — PNPI), Gatchina, 188300, Russian Federation, ORCID ID: 0000-0002-2314-0917, anatoly.oreshkin@gmail.com
Anton S. Suprun — Assistant, ITMO University, Saint Petersburg, 197101, Russian Federation, ORCID ID: 0000-0001-8429-9115, asuprun@list.ru