УДК 519.6

Модель обеспечения отказоустойчивости контейнерных виртуальных сервисов в центрах обработки данных

Рыбалко А.А.[1]*, Наумов А.В.[2]**

[1]EMC Информационные Системы Си-Ай-Эс,

ул. Беговая, 3, стр. 1, Москва, 125284, Россия

[2]Московский авиационный институт (национальный исследовательский университет), МАИ, Волоколамское шоссе, 4, Москва, А-80, ГСП-3, 125993, Россия

**e-mail: aar@mai.ru*

***e-mail: naumovav@mail.ru*

Аннотация

Работа посвящена актуальной теме увеличения надёжности функционирования пула распределённых сервисов с применением технологии контейнерной виртуализации. В рамках решения задачи предложен вариант реализации инфраструктуры виртуальных контейнеров, функционирующих в условиях изменяющейся нагрузки. В состав инфраструктуры включены средства развёртывания, сопровождения, отслеживания событий виртуальных контейнеров и реакции на них. Предложения в части инфраструктуры дополнены системными решениями в отношении реализации сервисов, а именно, описанием технологии децентрализованного обмена данными между виртуальными контейнерами и протоколов маршрутизации сетевых пакетов в условиях множественной аренды виртуальных контейнеров разным организациям.

Ключевые слова: контейнерная виртуализация, отказоустойчивость системы, гипервизор, децентрализованная сеть.

Введение

Виртуализация как идея представления вычислительного ресурса в форме, абстрагированной от аппаратной составляющей, давно известна и реализована целым рядом технологий [16,17]. В последние годы, за счёт появления аппаратного ускорения технологий виртуализации на широко распространённой платформе x86, виртуализация бурно развивается по разным направлениям и успешно применяется для решения множества практических задач [18]. Одна из таких задач - построение изолированной среды функционирования прикладных приложений, таких например, как системы дистанционного обучения [23,24], разработка компьютерных имитационных комплексов для отработки приемов управления широким классом летательных аппаратов и др. [22,23]. На ранних этапах развития виртуализации решалась задача изоляции вычислительной среды исполнения приложений: каждая такая среда должна получать свой ресурс и использовать его изолированно, то есть без влияния со стороны других сред, функционирующих на той же аппаратной платформе. Переход к изоляции отдельных приложений позволяет в теории значительно повысить эффективность использования аппаратного обеспечения: сократить число простаивающих систем, упростить миграции приложений между различными системами, управлять и балансировать нагрузочными параметрами

[1,19]. Инструменты виртуализации могли бы претендовать на универсальность, если бы изолируемые приложения не требовали сетевого взаимодействия, не стремились использовать удаленные ресурсы, что ограничило бы количество обращений к разнообразным системным службам, вызовам или файловым системам (ФС). Далее, рассуждая про виртуализацию, мы чаще всего подразумеваем так называемую полную виртуализацию (ПВ), то есть некоторую технологию, которая в рамках создания изолированной вычислительной среды эмулирует весь необходимый набор аппаратной поддержки. Такая среда нуждается во внешнем управлении (гипервизоре), который гарантирует доступность ресурса и его проекцию на реальное устройство. Соответственно, при ПВ наряду с очевидными плюсами появляются и издержки, такие как временные задержки на доступ к виртуальным машинам (ВМ) и избыточному потреблению ресурсов серверов при обращении к программному обеспечению (ПО) внутри ВМ. Что, от части, объясняется необходимостью эмуляции полноценного окружения для каждой из множества операционных систем (ОС) в ВМ.

Сравнительно новый взгляд на виртуализацию обеспечивается виртуализацией на уровне операционной системы, она же "контейнерная виртуализация" (КВ). Этот метод виртуализации подразумевает использование одного ядра хостовой ОС для создания множества независимых параллельно работающих операционных сред [20].

Таким образом, КВ лишена издержек, характерных для ПВ. Но при этом есть и ограничения по применению, ограничения универсальности. Как видно

непосредственно из самой идеи КВ, область ее перспективного применения – это формирование среды для функционирования ресурсоемких приложений. Причём зафиксированное ядро ОС определяет зачастую набор ПО, которые может выполняться в рамках программных сред КВ. То есть в рамках КВ разнообразие вычислительных сред не обеспечить, зато эффективно распределять ресурсы между ресурсоемкими приложениями плюс крайне эффективно обеспечивать отказоустойчивость – можно. Применение такого подхода особенно оправдано, если требуется запуск множества однотипных сервисов, доступность которых важна для большого числа потребителей [19].

Таким образом актуализируется задача поиска новых подходов к построению и функционированию виртуальной рабочей среды ресурсоемких приложений, обладающей как высокой доступностью, так и отказоустойчивостью. Основой для решения данной задачи может служить технология контейнерной виртуализации(КВ) [2]. При этом необходимо реализовать балансировку нагрузки и сетевого взаимодействия между распределёнными компонентами ПО при высокой интенсивности обращений.

Большинство коммерческих средств виртуализации предлагает использовать готовые шаблоны и типовые образцы архитектур [3] с расчётом на дальнейшую подстройку параметров среды на ходу под изменение условий работы. В архитектуру заранее не закладываются элементы балансировки нагрузки, позволяющие достичь гибкости при повышении частоты запросов к виртуальному ПО, а также не учитывается модель сетевого взаимодействия компонентов

архитектуры за рамками "плоской сети" взаимодействия. Это приводит к увеличению времени отклика и необходимости перестроения созданных архитектур уже после запуска систем в работу.

В статье рассматривается методика построения устойчивого к вредоносным воздействиям и высоким нагрузкам ПО на базе требований потребления ресурсов КВ, разделяющих ПО на значимые компоненты - сервисы, с организацией высокой доступности, балансировки нагрузки и сетевого взаимодействия. Прикладной задачей моделирования, в частности, является построение облачных архитектур с возможностью аренды ресурсов различным контрагентами (далее - тенанты), изначально рассчитанной на масштабирование под высокие нагрузки, с сохранением надёжности работы и скорости отклика на запросы пользователей.

Анализ архитектуры сервисов на базе контейнерной виртуализации

В основу разрабатываемой модели обеспечения отказоустойчивости контейнерных виртуальных сервисов в случае нагрузок или воздействия злоумышленника положены следующие требования:

- создание распределенных архитектур с количеством изолированных сервисов (приложений или компонентов приложений), превышающим стандартные лимиты (не более 1024 объектов) на хост ПВ[9,21];
- построение инфраструктуры с учётом "мульти-аренды" (multitenancy) - аренды ресурсов разными контрагентами для развёртывания сервисов;

- необходимость обеспечения надежной изолированности наборов сервисов разных тенантов;
- проектирование сетевой топологии для связи между сервисами, а также сбора статистики и передачи команд настройки для обеспечения отказоустойчивости работы сервисов;
- сбор статистики и децентрализованная передача информации в заданной топологии.

Выбор платформы КВ для построения модели обеспечения отказоустойчивости обосновывается ее преимуществами по сравнению с ПВ :

1. Меньшее потребление ресурсов, и, как следствие, более высокая плотность контейнеров на сервер: до 32000 контейнерных процессов на один хост [10];
2. Возможность работы с более высокими нагрузками;
3. Повышенная скорость развёртывания копий контейнеров из шаблона;
4. Упрощённая схема взаимодействия контейнеров с управляющей операционной системой

Указанные преимущества позволяют как изолировать приложения друг от друга,так и дробить их на изолированные компоненты-сервисы, что особенно востребовано распределёнными командами разработчиков. Такой подход получил название «сервис-ориентированная архитектура» (Service-Oriented Architecture - SOA) или «микросервисы» [3]. Отличительные черты таких приложений:

- Сервисы являются небольшими, слабо связанными модулями, каждый из которых выполняют свою, часто единственную функцию;
- Каждый сервис — законченный, при этом легко меняющийся модуль;
- Развёртывание и изменения в сервисы должны производиться с использованием средств автоматизации.

Для обеспечения отказоустойчивости изолированных сервисов приложений, а также балансировки нагрузки между ними, возможно развёртывание избыточного количества виртуальных контейнеров, с созданием управляющей сети взаимосвязей между ними для сопровождения и контроля за их жизненным циклом [4]. Избыточное количество изолированных контейнеров и более высокая плотность в пересчёте на узел АО открывает возможность к получению преимуществ в построении устойчивой и работоспособной архитектуры систем. Проведённые исследования [8] указывают на более чем двукратном повышении плотности контейнеров технологий OpenVZ и LXC в сравнении с гипервизорами ПВ - VMware ESXi, KVM, Microsoft Hyper-V, Citrix XenServer.

В качестве платформы КВ для инфраструктуры облака целесообразно использование OpenVZ, так как архитектура LXC рассчитана на быстрое создание сред для тестирования кода разработчиками, а не как долгосрочная среда для исполнения бизнес-приложений. Преимущества OpenVZ в сравнении с другими технологиями КВ:

- стабильность;

- наличие плавной защиты контейнеров от переполнения ОЗУ (виртуальный своп);
- поддержка виртуальных дисков с возможностью развёртывания у каждого контейнера собственной ФС;
- квотирование пространства, занимаемого контейнером;
- миграция контейнеров без выключения и простоя между несколькими хостами;
- альтернативность сетевых настроек: виртуальный мост (bridge), маршрутные таблицы (virtual routing) и проброс физической сетевой карты в контейнер [7].

Главный недостаток OVZ - отсутствие интеграции в ядро Linux и, как следствие, ограничение в полнофункциональном развёртывании линейкой дистрибутивов RHEL/Fedora/CentOS/Scientific Linux, а также необходимость команды разработчиков проекта в систематическом портировании технологий из свежей версии ядра в более старое ядро поддерживаемой версии. OpenVZ позволяет работать с поддержванием приемлемой скорости работы до 16000 контейнерных процессов на один хост КВ, при максимальном протестированном лимите в 32000 процессов [10], что превышает максимальный лимит виртуальных машин на один хост ПВ приблизительно в 16-30 раз [9].

Обратной стороной раздробленных архитектур типа SOA является лавинообразное увеличение (от 2х до 10х раз) количества модулей с необходимостью контроля их жизненного цикла. Классические средства клиент-серверной архитектуры, в которых единый кластер мониторинга централизованно

опрашивает узлы или собирает информацию с агентов (такие как Nagios, Zabbix, Microsoft SCVMM и SCOM, VMware vRealize Operations, Quest Foglight и т.д.), дают слишком сильную нагрузку на сетевую инфраструктуру, а также не справляются с самостоятельной обработкой поступающих данных с множества контейнеров. Вынужденной мерой является указание метрик для отслеживания и фильтрации, как следствие, снижение точности мониторинга. Для примера, современная система клиент-серверного мониторинга рассчитана на количество узлов сбора статистики (single node max objects) не более 15000, при этом требует до 48Gb ОЗУ и 16 ядер ЦПУ [12], что фактически означает необходимость выделения целого сервера под мониторинг одного хоста КВ максимальной плотности.

Отказоустойчивая архитектура управляемых контейнеров

Для построения отказоустойчивой системы требуется новый подход в организации сетевой архитектуры КВ. На рисунке 1 представлена гибридная децентрализованная сетевая топология взаимодействия хостов КВ и контейнеров.

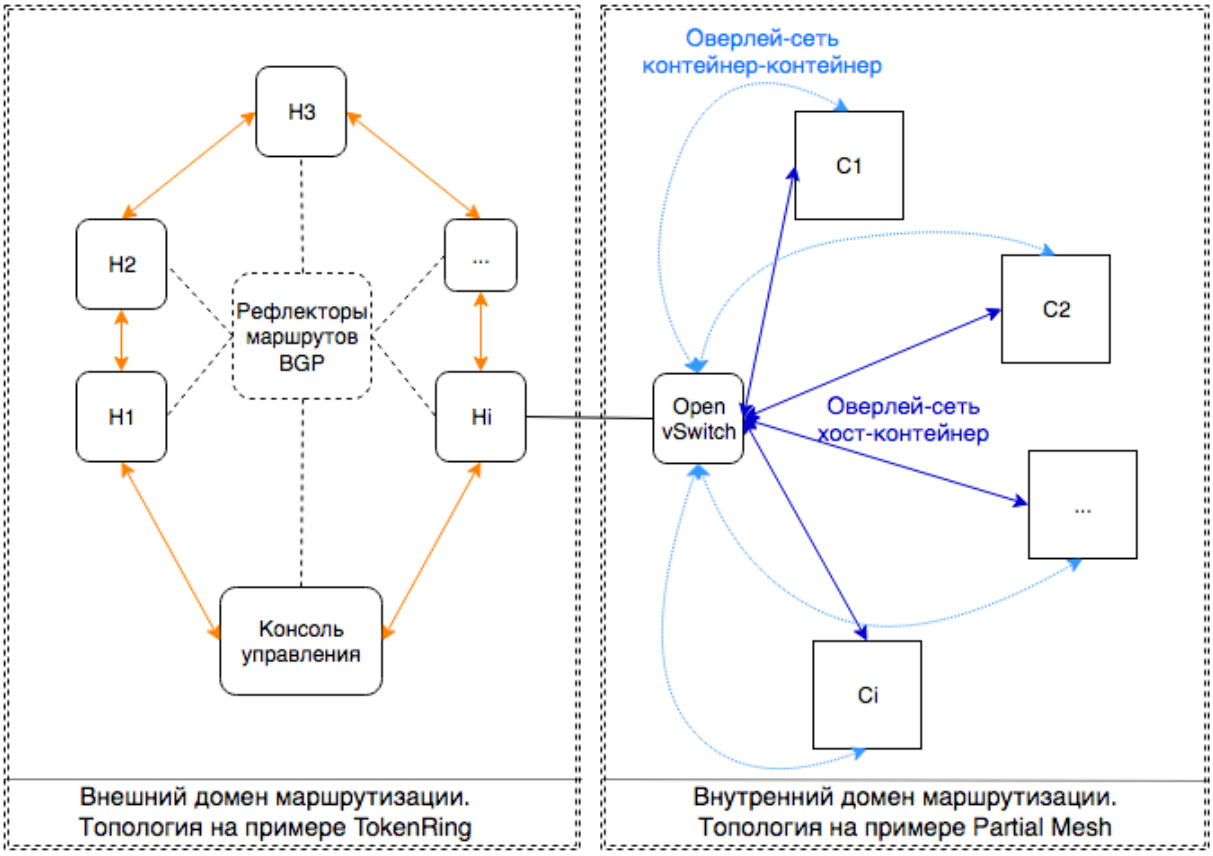

Рис.1 Сетевая топология хостов КВ.

Для описания проблематики взаимодействия контейнеров и модели отказоустойчивых мультисервисных систем под нагрузкой, воспользуемся формализацией конечно-множественных представлений доменной модели.

Обозначим множество физических серверов(хостов), на которых разворачиваются рабочие инструменты КВ -кластера H ={H1,H2,...Hn} при условии, что n >=3 (см. рис.1) Каждый узел имеет два сетевых интерфейса и образует сегмент, похожий на кольцо. В сравнении с классической схемой типа звезда, такой подход имеет ряд преимуществ:

- дублированный доступ к хостам по кольцу
- упрощенная маршрутизация за счет фиксированного числа uplinks (на каждом хосте - 2);

- экономия пассивных и активных элементов ВОЛС;
- удаление и добавление узла в кольце требует минимум действий по коррекции таблиц маршрутизации (4 операции для соседних хостов в двунаправленном кольцевом списке).

На каждом хосте Hi (i=1..n) имеется множество C={C1,C2,....Ck} контейнеров с сервисами S={S1,S2,..Sm}. Количество контейнеров Cj (j=1..k) на каждом сервере ограничено его ресурсами, однако может принимать достаточно большие значения (k ~ 32000), поэтому для отказоустойчивой вычислительной системы необходимо организовать децентрализованный обмен информацией и агрегацию статистики на уровне самих контейнеров, с передачей на хост необходимого минимума релевантных событий, потенциально требующих реакции планировщика хоста.

Для обеспечения устойчивой работы сервисов в среде КВ требуется убедиться в правильности их функционирования. Это могут быть серии специальных тестов, кратких запросов или небольших пакетов уровня TCP/IP. Алгоритм контроля работоспособности контейнерных систем в рамках рассматриваемой доменной модели представлен на рисунке 2.

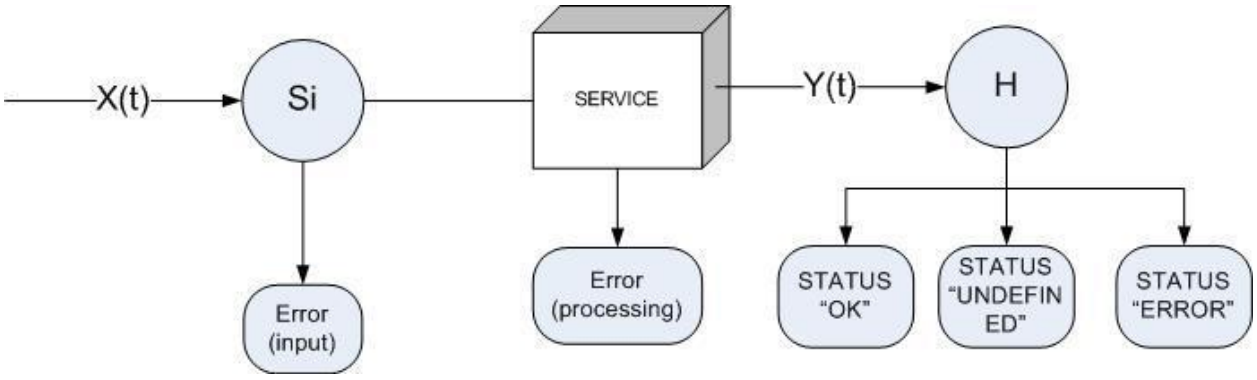

Рис.2 Процесс верификации работы сервисов

Каждый сервис Sl (l=1..m) в фокусе тестирования его состояния может быть представлен как «черный ящик», в который на вход подается некоторый набор исходных данных - X(t). В результате обработки(отклика) формируется "выходной импульс" - Y(t). X(t) представляет из себя входной скрипт (или входной набор данных) для выполнения простейшего плана проверки. Время выполнения такого плана должно быть минимальным $t \rightarrow 0$, чтобы выполнение теста не мешало работе сервиса. Y(t) представляет собой статусный отклик о состоянии сервиса (см. рис 2). Для каждого проверяемого сервиса сохраняется их история и фиксируются все изменения в поведении сервисов. В случае обнаруженных отклонений, планировщик может усложнить входной тест (с целью детализации сбоя в сервисе) или сразу инициировать сигнал тревоги на хост КВ, если статус определяется как "ошибочный"

Таким образом может быть построена итоговая (по всем сервисам Sl с учетом их значимости (веса) для работоспособного состояния ВК) оценка работоспособности контейнера Cj. Сравнив ее с пороговым значением (могут различаться для различных видов бизнес- приложений), планировщик вычислений осуществляет управляющее воздействие на j контейнер. Операция анализа состояния работоспособности контейнера выполняется автоматически (с определенным временным интервалом), а решение о дальнейших действиях с контейнером могут выполняться как в автоматическом режиме, так и в режиме "заморозки" текущего состояния контейнера до момента ручного вмешательства в

этот процесс со стороны администратора. В любом случае, возможный перечень действий ограничивается следующими альтернативами:

Перезапустить проблемные сервисы с целью их перехода в корректные начальные состояния;

1. Осуществить перезапуск проблемного контейнера C_j;
2. Удалить проблемный контейнер C_j и осуществить запуск клонированного контейнера с корректным состоянием работы сервисов.

Данный алгоритм описывает динамическую комплексную проверку работоспособности и может быть реализован без значительных затрат ресурсов центра обработки данных (ЦОД).Однако, для его практической реализации потребуется сетевая топология контейнеров типа "слабосвязанная фабрика" (partial mesh). Первоначальное сетевое соединение устанавливается между хостом КВ и каждым контейнером. По мере необходимости сетевого взаимодействия между контейнерами централизованное управление теряет актуальность и в результате работы алгоритма обнаружения, работающего согласно одному из протоколов внутридоменной маршрутизации с возможностью сохранения состояния: OSPF, IS-IS, IGRP или EIGRP, топология меняется на древовидную. Дальнейшая децентрализация приводит к тому, что выбор внутридоменного протокола маршрутизации не является детерминированным в связи с принадлежностью подмножеств контейнеров разным тенантам, со своими правилами сетевого обмена. В связи с этим предлагается определиться с тем, какую технологию использовать на

транзитном шлюзе (в качестве такового на рисунке 3 используется хост КВ Hi) для разделения трафика приложений и служб мониторинга.

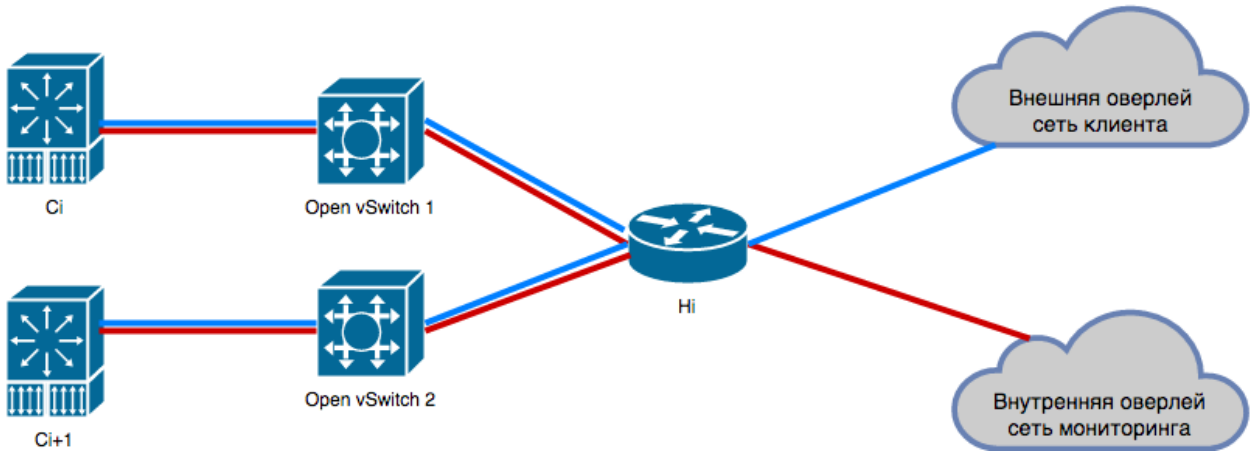

Рис.3 Схема изоляции различных видов трафика.

Существуют три технологии изоляции трафика: Policy Based Routing (PBR), где в выборе маршрута доминирует адрес отправки IP-пакета, Virtual Routing and Forwarding (VRF), где маршрут определяется по виртуальной таблице маршрутизации и Access Lists (ACL), где определяющую роль играют индивидуальные списки доступа. Технология VRF в описанной схеме имеет следующие преимущества над PBR и ACL:

1. Обеспечивает полную изоляцию маршрутных таблиц для трафика от различных подсистем.
2. Высокая степень безопасности и надежности передаваемых данных из-за отсутствия взаимного влияния маршрутов друг на друга и политик маршрутизации одной подсистемы на другую;

3. Статичность PBR- политик для каждого транзитного узла прохождения трафика. В динамической виртуальной среде возможны ошибки и затруднения в их обнаружении и устранении
4. Простота настройки и относительная легкостью в сопровождении.

Внутри VRF можно запусить несколько OSPF или IS-IS процессов, а также один процесс EIGRP, RIP и BGP. При настройке протоколов маршрутизации OSPF и IS-IS используются отдельные независимые процессы под каждый VRF, при настройке EIGRP, RIP и BGP внутри протоколов маршрутизации настраиваются различные семейства адресов (address-family) под разные VRF внутри одного процесса маршрутизации (см. рисунок 4).

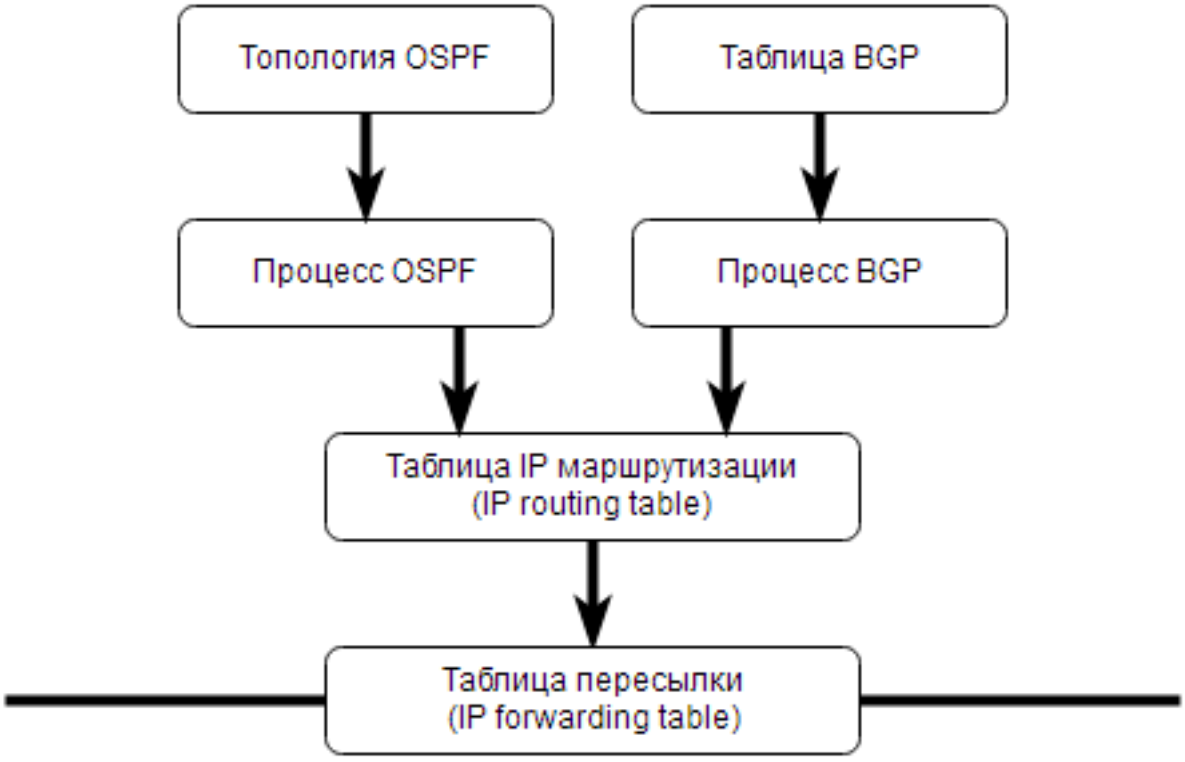

Рис.4 Схема взаимодействия протоколов маршрутизации в VRF.

В описанной архитектуре на шлюзе (модуль маршрутизации), развернутом на хосте КВ, в направлении виртуального коммутатора Open vSwitch настраиваются 2

под-интерфейса, которые помещаются в разные VRF: клиентский и VRF для средств сбора статистики. В клиентском VRF прописан маршрут по умолчанию в шлюз Интернет-сети, а в VRF для средств сбора статистики указывается маршрут умолчания (default route) в направлении сети провайдера. Для того, чтобы избежать необходимости настройки отдельного экземпляра протокола маршрутизации под каждый VRF на всех транзитных шлюзах (нерационально в плане управления настройками и использования ресурсов каждого сервера) предлагается использовать мультипротокольный BGP (Multi-Protocol BGP - MP-BGP). Схема работы приведена на рисунке 5.

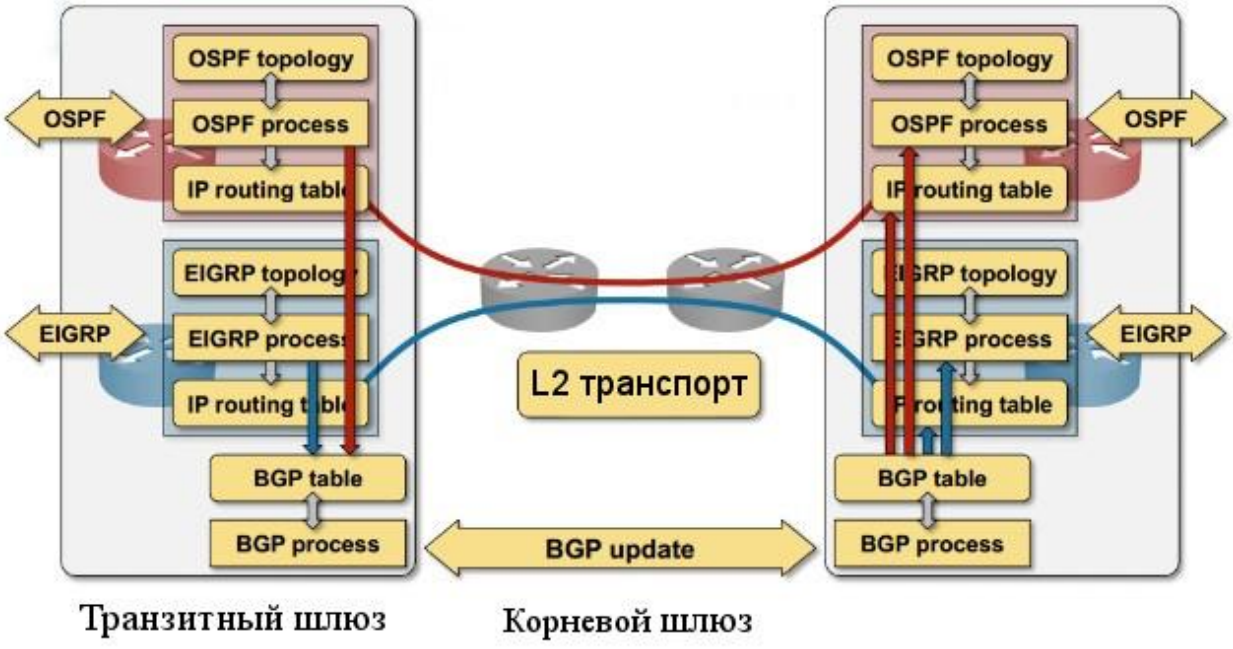

Рис.5 Схема работы MP-BGP.

На пограничных узлах запускается BGP-процесс, в котром настраивается семейство адресов (address-family) по IPv4 под каждый VRF (MP-BGP). Далее происходит перераспределение маршрутов из VRF IGP в MP-BGP и из MP-BGP в

VRF IGP (альтернатива - перераспределением default route). Правила перераспределения из IGP в MP-BGP изменены специальным образом, чтобы исключить появление маршрутных петель (routing loops). Например, при перераспределении из EIGRP в MP-BGP добавляются дополнительные аттрибуты, поэтому при обратном перераспределении на принимающем узле EIGRP метрика сохраняется полностью. То же самое происходит при перераспределении из OSPF в MP-BGP на узле отправления и обратном перераспределении на принимающем узле: внешние OSPF маршруты остаются внешними, внутренние OSPF маршруты остаются внутренними, и OSPF метрика всех маршрутов прозрачно передается в неизменном виде через операцию BGP update на принимающий узел.

При наличии транзитных узлов между двумя площадками хостов КВ возникает необходимость обеспечить транспорт между VRF'ами на двух площадках. В качестве транспорта можно использовать:

- транспорт второго уровня (Layer 2 transport)
- VXLAN, NVGRE, Geneve, STT, IPsec
- MPLS

Представляется наиболее эффективным использование туннелей VXLAN или Geneve, так как реализация оверлей-сетей на основе этих технологий не требует установку дополнительных компонентов на хосты КВ, а является надстройкой к существующей технологии виртуальной коммутации Open vSwitch - Open Virtual Network (OVN) [11].

Построение контроллера взаимодействия и сопровождения

Для увязывания воедино описанных технологий передачи информации, сбора и агрегации статистических данных, а также выполнения управляющих команд, разработан специализированный механизм взаимодействия объектов внутри доменного пула(хостов, контейнеров, сервисов) с возможностью вмешательства в автоматизированный процесс со стороны внешнего наблюдателя. Типовые задачи, возникающие при обеспечении отказоустойчивой работы локализованных в контейнерах сервисов представлены на диаграмме прецедентов рис. 6.

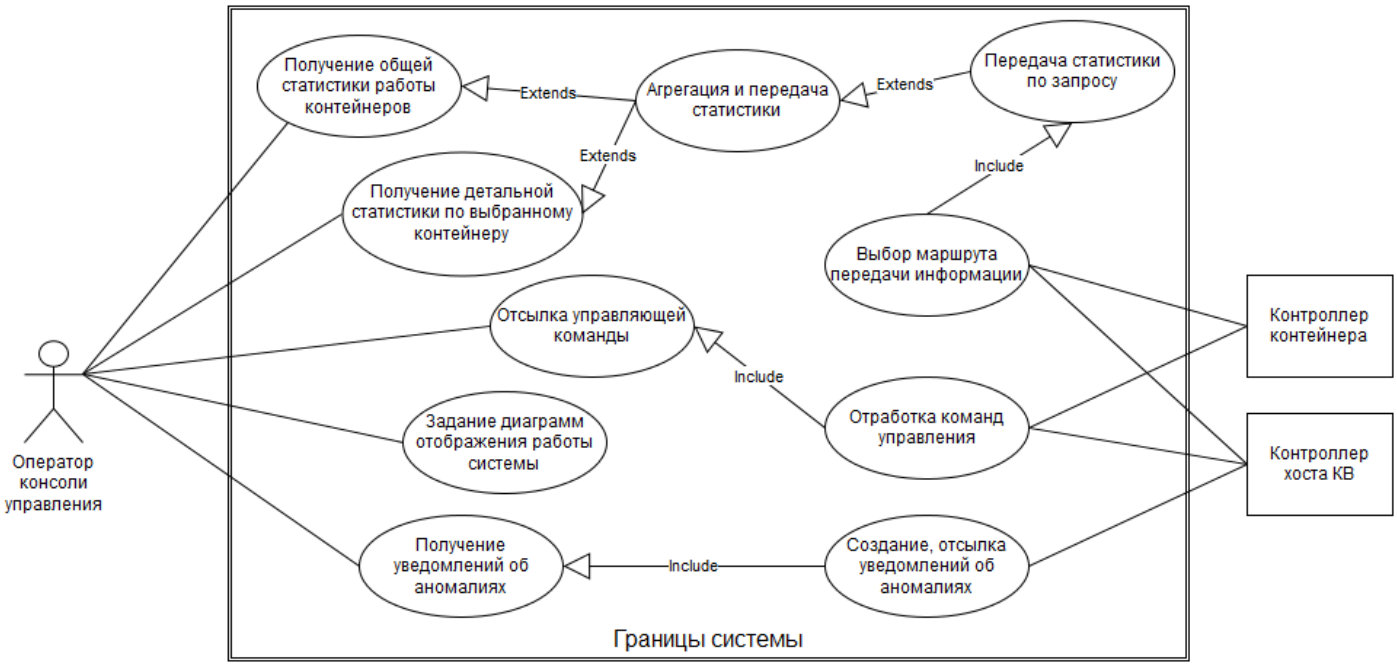

Рис. 6. Диаграмма прецедентов контроллера взаимодействия.

Рассмотрим подробнее структуру объектов контроллера на примере управления сервером(хостом) рис. 7. В качестве основы для передачи информации во внешний домен маршрутизации будем использовать реализацию программного модуля работы с BGP протоколом - ExaBGP [14]. Данный модуль поддерживает BGP до версии 4, обладает удобством взаимодействия посредством обмена

стандартными пакетами данных с JSON форматом, открыт для изменений технически и лицензионно, а также легко встраивается в любой код.

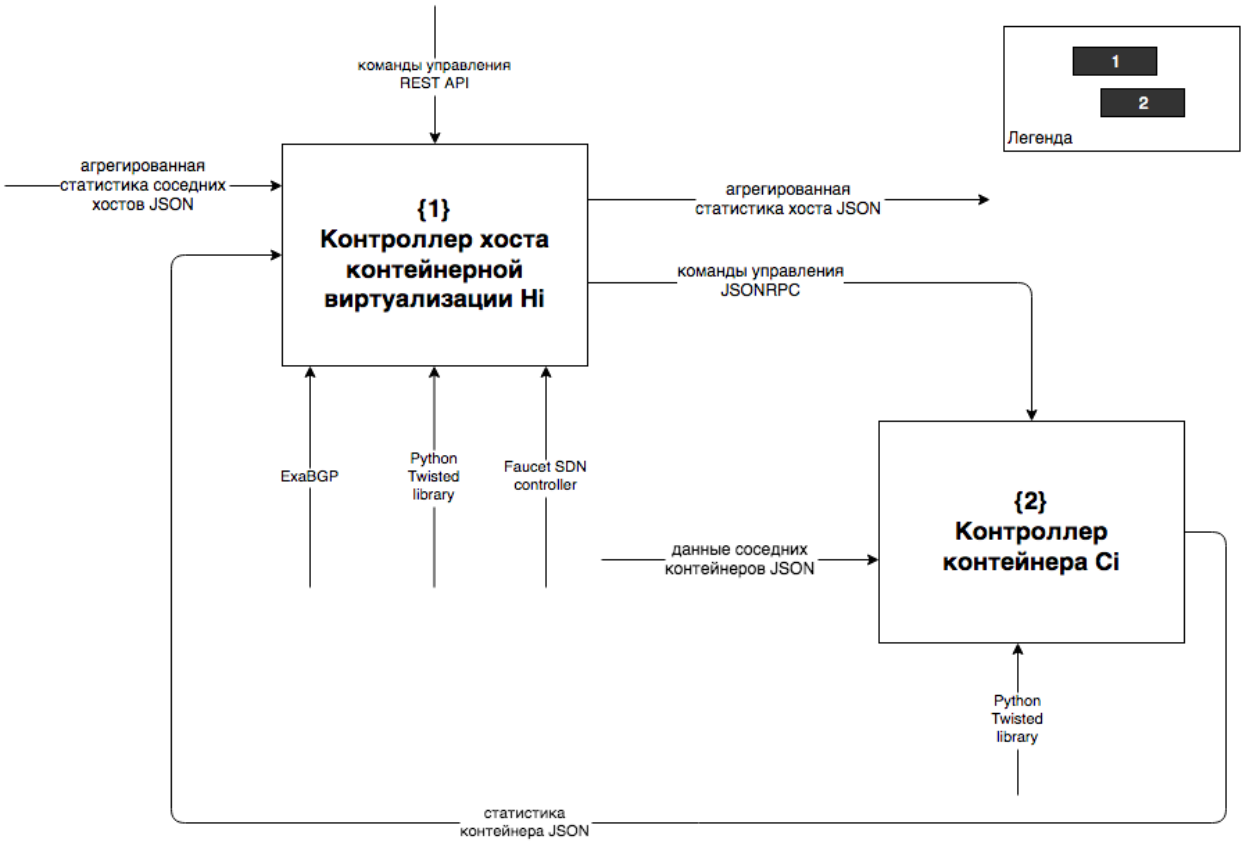

Рис. 7. Диаграмма связей контроллера взаимодействия.

Параллельность и многопотоковость сетевой передачи данных реализуется посредством библиотеки управления распределёнными сетевыми взаимодействиями в рамках языка Python - Twisted, в сочетании с механизмом асинхронной обработки потоков asyncio, встроенного и доработанного для использования в продуктивной среде в последней версии языка Python 3.6 [13]. В качестве основы для задания маршрутов и настройки оверлей-сетей представляется эффективным использования SDN-контроллера Faucet [15], построенного на библиотеке Python Twisted, имеющего тесную интеграцию с ExaBGP, набор функций работы с оверлей-сетями, а также поддержкой всех необходимых протоколов коммутации.

Выводы

В статье была представлена методика моделирования отказоустойчивой среды взаимодействующих серверов, виртуальных контейнеров и сервисов, локализованых в них по признаку функциональной и логической целостности вычислительной системы, и представленная тройкой {H,C,S}. Сравнительный анализ технологий ПВ и КВ, а также сравнение популярных технологий КВ с учётом функционала изоляции сервисов разных тенантов, позволил построить ряд схем, алгоритмов и технологических сборок, улучшающих показатели отказоустойчивости вычислительного процесса при высоких нагрузках на такой сервис или дейстий злоумышленников. Для апробации предложенных решений выбрана технология OpenVZ, наиболее подходящая по требованиям стабильности и эффективная по ресурсам. Разработанная гибридная сетевая топология вида кольцо с неполносвязной фабрикой для связи между сервисами. Проведён анализ и подбор стека протоколов маршрутизации для сбора статистики и передачи команд настройки для обеспечения работоспособности сервисов. Проведено проектирование контроллера сбора статистики и децентрализованной передачи информации в заданной топологии, а также описан механизм для опроса сервисов, накопления и анализа статистических данных.

Работа выполнена при поддержке гранта РФФИ № 15-07-02914.

Библиографический список

1. Рыбалко А.А. Моделирование системы защиты облачных сервисов с использованием механизмов виртуализации // Вестник Московского Авиационного Института. 2010. Т. 16. № 6. С. 143 – 149.

2. IBM Research Report. An Updated Performance Comparison of Virtual Machines and Linux Containers / Wes Felter, Alexandre Ferreira, Ram Rajamony, Juan Rubio. IBM Research Division, Austin Research Laboratory, 11501 Burnet Road, Austin, TX 78758, USA. RC25482 (AUS1407-001) July 21, 2014. Computer Science.

3. James Lewis, Martin Fowler. Microservices - a definition of this new architectural term, available at: https://martinfowler.com/articles/microservices.html

4. Рыбалко А.А. Механизмы сетевого взаимодействия в системе сопровождения инфраструктуры на базе виртуальных контейнеров с приложениями // Материалы XX Международной конференции по вычислительной механике и современным прикладным программным системам (ВМСППС'2017), 25-31 мая 2017, Алушта. - М.: Изд-во МАИ-ПРИНТ, 2017. – С. 159 – 160.

5. Alfred V. Aho, Jeffrey D. Ullman, John E. Hopcroft. Data Structures and Algorithms, ISBN-10: 0201000237, ISBN-13: 978-0201000238. Pearson; 1st edition (January 11, 1983), 427 p.

6. Рыбалко А.А. Формализация средств обеспечения устойчивой работоспособности виртуальных сервисов в центрах обработки данных // Материалы VIII Международной конференции по неравновесным процессам в соплах и струях (NPNJ'2010), 24-31 мая 2010, Алушта. - М.: Изд-во МАИ, 2010. - С. 595 – 597.

7. Feature comparison of different virtualization solutions, available at: https://openvz.org/Comparison

8. Performance Evaluation of Virtualization Technologies for Server Consolidation / Pradeep Padala, Xiaoyun Zhu, Zhikui Wang, Sharad Singhal, Kang G. Shin. HP Laboratories, HPL-2007-59R1, 2007, available at: http://www.hpl.hp.com/techreports/2007/HPL-2007-59R1.pdf

9. VMware documentation guide on vSphere 6.5 maximums, available at: https://www.vmware.com/pdf/vsphere6/r65/vsphere-65-configuration-maximums.pdf

10. OpenzVZ project documentation page on UBC primary parameters, available at: https://openvz.org/UBC_primary_parameters

11. What Is Open Virtual Network (OVN)? How It Works? - Open Virtual Network project documentation page, available at: https://www.sdxcentral.com/sdn/network-virtualization/definitions/what-is-open-virtual-network-ovn-how-it-works/

12. John Dias. VMware corporate blog. Posted March 23, 2017, available at: https://blogs.vmware.com/management/2017/03/vr-ops-6-5-whats-new-improved-scalability.html

13. Elvis Pranskevichus, Yury Selivanov. Python documentation. What's New In Python 3.6., available at: https://docs.python.org/3/whatsnew/3.6.html

14. Thomas Mangin. ExaBGP project documentation, available at: https://github.com/Exa-Networks/exabgp

15. Faucet project documentation. "What is Faucet?" page, available at: https://faucetsdn.github.io/

16. Захаров В.Н. Виртуализация как информационная технология // Системы и средства информатики. 2006. Т. 16. № 3. С. 279 – 298.

17. James Smith, Ravi Nair. The Architecture of Virtual Machines // IEEE Computer Society. 2005. 38 (5), pp. 32 – 38.

18. Gideon Gerzon. Intel® Virtualization Technology, Processor Virtualization Extensions and Intel® Trusted Execution Technology. 2007, available at: https://software.intel.com/sites/default/files/m/0/2/1/b/b/1024-Virtualization.pdf

19. Баранов А.В., Николаев Д.С. Использование контейнерной виртуализации в организации высокопроизводительных вычислений // Программные системы: теория и приложения. 2016. Т. 7. № 1, С. 117 – 134.

20. Виртуализация: технологические подходы. URL: http://ru.pcmag.com/osnovy/3919/help/virtualizatsiia-tekhnologicheskie-podkhody

21. Кондрашин М.А., Арсенов О.Ю., Козлов И.В. Применение технологии виртуализации и облачных вычислений при построении сложных распределенных моделирующих систем // Труды МАИ. 2016. № 89 URL: http://trudymai.ru/published.php?ID=73411

22. Наумов А.В., Сай Кхин Аунг Тинт Об адаптации обучающих систем переподготовки молодых специалистов на предприятиях авиационного комплекса // Труды МАИ. 2011. № 42. URL: http://trudymai.ru/published.php?ID=24321

23. Naumov A.V., Mkhitaryan G.A., Rybalko A.A. Software set of intellectual support and security of LMS MAI CLASS.NET // Вестник ЮУрГУ. Математическое моделирование и программирование. 2016. Т. 9. № 4. С. 129 - 140