УДК 004.457+004.031.2+004.382.2
DOI: 10.15827/2311-6749.18.3.8

СПОСОБЫ И СРЕДСТВА ДИНАМИЧЕСКОЙ РЕКОНФИГУРАЦИИ СЕТЕЙ СУПЕРКОМПЬЮТЕРА ПРИ ПРЕДСТАВЛЕНИИ ПОЛЬЗОВАТЕЛЬСКИХ ЗАДАНИЙ В ВИДЕ КОНТЕЙНЕРОВ

А.В. Баранов, к.т.н., доцент, antbar@mail.ru
(Межведомственный суперкомпьютерный центр Российской академии наук – филиал ФГУ «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук», Ленинский просп., 32а, г. Москва, 119334, Россия);
А.С. Шитик, salexs95@yandex.ru
(Московский физико-технический институт (государственный университет), Керченская ул., 1а, корп. 1, г. Москва, 117303, Россия)

Одним из ключевых методов повышения эффективности использования вычислительных ресурсов является применение технологии контейнерной виртуализации. В отношении организации высокопроизводительных вычислений контейнерная виртуализация позволяет с минимальными накладными расходами решить проблему бинарной переносимости пользовательских заданий между различными суперкомпьютерными установками. Однако для возможности обработки представленных в виде контейнеров пользовательских заданий в системах коллективного пользования необходим механизм динамической реконфигурации сетей суперкомпьютера, осуществляемой перед каждым запуском задания. Статья посвящена поиску и выбору способов и средств, позволяющих осуществить динамическую реконфигурацию при запуске заданий в контейнерах Docker в суперкомпьютерных системах коллективного пользования.

Ключевые слова: *контейнерная виртуализация, высокопроизводительные вычисления, Docker, etcd, calico, СУППЗ.*

В настоящее время применение технологий виртуализации для организации высокопроизводительных вычислений является одним из активно развивающихся направлений научных исследований и технических разработок. Наиболее перспективными в этом плане выглядят технологии контейнерной виртуализации, позволяющие радикально снизить накладные расходы на виртуализацию [1].

Современная суперкомпьютерная система представляет собой, как правило, высокопроизводительный вычислительный кластер, состоящий из множества многопроцессорных (многоядерных) *вычислительных модулей* (ВМ), объединенных высокоскоростной сетью передачи данных. Суперкомпьютеры используются в режиме коллективного доступа, при котором для производства вычислений пользователь должен оформить т.н. задание, представляющее собой информационный объект, в состав которого входят:

– параллельная программа, решающая прикладную задачу;

– требования к ресурсам: сколько ВМ, с какими характеристиками, на какое время требуется для выполнения параллельной программы;

– входные данные параллельной программы.

Параллельная программа состоит из нескольких взаимодействующих посредством высокоскоростной сети процессов, которые могут одновременно выполняться на нескольких ВМ суперкомпьютера. Отметим сильную зависимость параллельной программы от стека системного и инструментального ПО, установленного на суперкомпьютере, что часто приводит к невозможности переноса без перетрансляции программы между разными суперкомпьютерными системами – т.н. бинарной несовместимости. Преодоление бинарной несовместимости суперкомпьютерных приложений возможен за счет объединения технологий контейнерной виртуализации и высокопроизводительных параллельных вычислений. Пользователь сможет самостоятельно выбрать программную платформу параллельных вычислений и использовать ее для своих вычислительных заданий, предварительно подготовив определенного формата образ (контейнер), включающий необходимый стек системного, инструментального и прикладного ПО. В дальнейшем созданный контейнер может быть запущен на любой суперкомпьютерной системе как обычное пользовательское задание.

Управление пользовательскими заданиями в современных суперкомпьютерных системах осуществляется с помощью специального программного обеспечения – *систем управления заданиями* (СУЗ) [2], в качестве которых могут выступать такие известные продукты, как SLURM, PBS, Moab и др. Основной функционал СУЗ состоит в приеме входного потока пользовательских заданий, ведении очереди заданий, выделении вычислительных модулей суперкомпьютера для прошедшего через очередь задания, в запуске и контроле выполнения задания, освобождении выделенных ресурсов по завершении задания. Важ-

ным аспектом функционирования любой СУЗ является обеспечение надежной изоляции пользовательских заданий. Изоляция заданий подразумевает конфигурирование выделенных заданию ВМ таким образом, чтобы доступ к этим ВМ мог получить только пользователь – владелец задания. Кроме того, должно быть исключено влияние выполняющихся процессов одного задания на выполнение процессов другого задания.

Предполагается, что при запуске задания из образа (контейнера) экземпляр этого контейнера будет развернут на каждом выделенном системой управления заданиями ВМ, после чего внутри развернутых контейнеров будут запущены процессы параллельной программы пользовательского задания. Для обеспечения возможности информационных обменов между параллельными процессами задания необходима такая конфигурация подсети развернутых контейнеров, в которой выполняющиеся параллельные процессы «видят» друг друга и могут взаимодействовать. При этом необходимый уровень изоляции заданий определяется невозможностью проникновения в сконфигурированную подсеть контейнеров иных, не относящихся к заданию процессов.

ВМ суперкомпьютера выделяются СУЗ динамически из числа свободных на момент запуска задания, и заранее предсказать состав выделяемых модулей невозможно. В случае представления задания в виде контейнера или образа виртуальной машины это порождает задачу динамической конфигурации сетей суперкомпьютера для такого задания [3] с автоматической организацией необходимого уровня изоляции заданий. Настоящая работа посвящена поиску решений этой задачи для случая контейнерной виртуализации.

Средства контейнерной виртуализации

Существуют два основных подхода к виртуализации как к созданию независимых изолированных вычислительных пространств на одном физическом ресурсе: виртуальные машины и контейнерная виртуализация. В первом случае на одном физическом компьютере создается множество виртуальных машин, в каждой из которых используется собственная, т.н. «гостевая», *операционная система* (ОС) с ее собственным ядром. Управление виртуальными машинами и ресурсами физического компьютера осуществляется специальным программным средством, именуемым гипервизором. По этой причине такой тип виртуализации часто называют гипервизорной виртуализацией.

Контейнерная виртуализация, или виртуализация на уровне ОС [4] – это метод, при котором ядро ОС поддерживает несколько изолированных экземпляров пользовательского пространства, что позволяет запускать изолированные и безопасные виртуальные машины на одном физическом компьютере. Контейнерная виртуализация ограничивает тип используемой гостевой ОС операционной системой физического компьютера, но накладные расходы на контейнерную виртуализацию существенно ниже по сравнению с гипервизорной [1], что очень важно для суперкомпьютерных расчетов.

Прообраз средств контейнерной виртуализации можно увидеть в классической Unix-команде chroot, предоставляющей простейшую изоляцию файловой системы путем смены корневого каталога пользователя. По-настоящему о контейнерной виртуализации заговорили после появления в Linux механизмов пространства имен и контрольных групп, одним из самых известных проектов, использующих эти возможности, стал проект контейнеров Linux под названием LXC (Linux Containers). Существенный толчок в развитии технологии контейнерной виртуализации был дан в 2013 году появлением ПО Docker, компания-разработчик которого Docker Inc предоставила стандартный API, упростивший создание и использование контейнеров. В настоящее время Docker является средством контейнерной виртуализации, получившим наибольшее распространение, имеющим свободно распространяемую версию, наиболее активно развивающимся и имеющим наибольшее число готовых решений для организации сетевого взаимодействия контейнеров [4].

Docker позволяет «упаковать» приложение со всем его окружением и зависимостями в контейнер, переносимый между любыми Linux-системами, ядро которых обладает поддержкой механизма контрольных групп. Изначально Docker использовал возможности реализации технологии контейнерной виртуализации LXC, но с 2015 года применяет собственную библиотеку libcontainer, абстрагирующую возможности ядра Linux по виртуализации.

ПО Docker может быть использовано для представления суперкомпьютерных пользовательских заданий в виде контейнеров [5, 6]. В указанных работах развертывание на суперкомпьютере пользовательского задания в виде набора контейнеров существенно опирается на возможности и средства распространенной СУЗ SLURM, что ограничивает применение предложенного способа. Например, в *Межведомственном суперкомпьютерном центре РАН* (МСЦ РАН) [7] и Институте прикладной математики им. М.В. Келдыша РАН [8] активно применяется отечественная *Система управления прохождением параллельных заданий* (СУППЗ) [8, 9]. С точки зрения организации вычислений главным отличием СУППЗ от других СУЗ, прежде всего от SLURM, является отсутствие постоянно работающих активных програм-

мных компонентов, установленных на ВМ, – все компоненты СУППЗ устанавливаются на управляющую ЭВМ.

Сетевая инфраструктура современной суперкомпьютерной системы

Де-факто в современных суперкомпьютерных системах используются сети двух типов: транспортные (управляющие) и высокоскоростные коммуникационные сети.

В качестве транспортных и управляющих используются Ethernet-сети. Транспортные сети, как правило, обеспечивают доступ с ВМ суперкомпьютера к системе хранения данных. Управляющие служат для целей управления и мониторинга, СУЗ использует управляющую сеть при выделении и конфигурации ресурсов для очередного задания, для запуска и контроля задания, для освобождения ресурсов после завершения задания.

Высокоскоростные коммуникационные сети предназначены для обмена информацией между процессами параллельной программы, выполняющимися на разных выделенных заданию ВМ. Стандартом последних лет в области высокопроизводительных вычислений стало использование в качестве высокоскоростных сетей Infiniband. С 2016 года началось внедрение сетей Intel Omni-Path [11] – реализации Infiniband от компании Intel.

Типовая организация сетевой инфраструктуры суперкомпьютера представлена на рисунке 1.

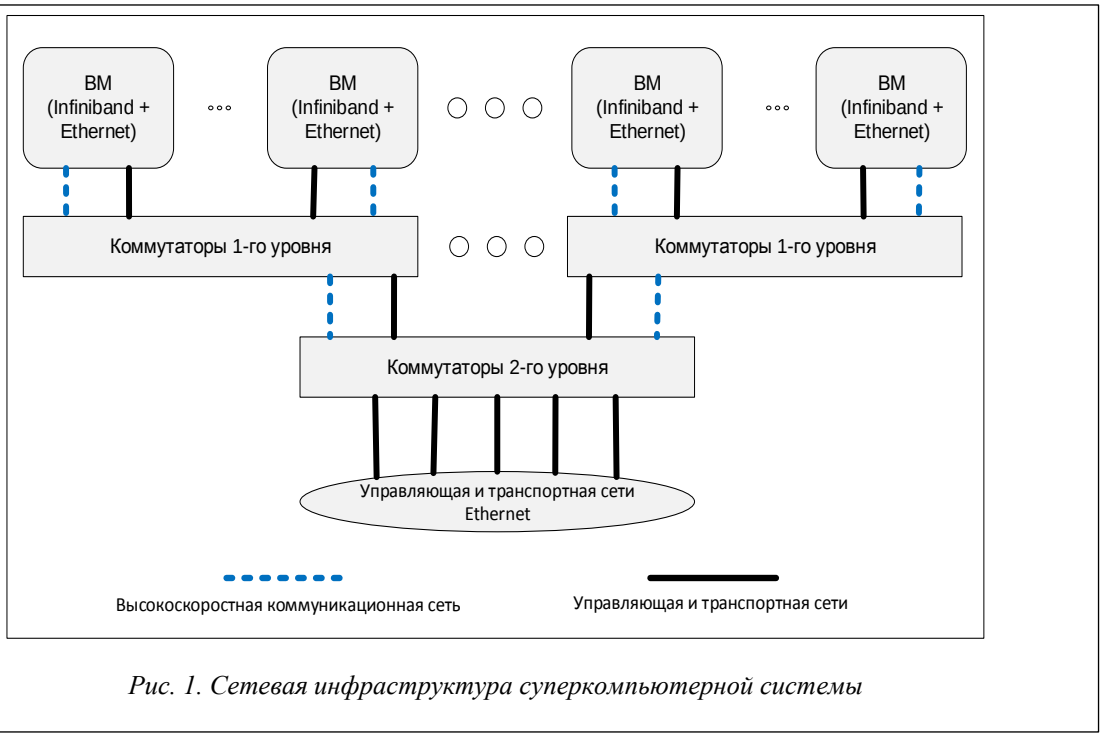

Рис. 1. Сетевая инфраструктура суперкомпьютерной системы

По сути организация транспортных и управляющих сетей суперкомпьютера ничем не отличается от организации обычных локальных сетей, основанных на стеке протоколов TCP/IP. Главным недостатком этого стека являются высокая латентность, обуславливающая существенные потери производительности при параллельных вычислениях [1]. По этой причине высокоскоростная сеть суперкомпьютера (Infiniband, Omni-Path) используется без применения стека TCP/IP.

Представление пользовательского задания в виде контейнера подразумевает следующий порядок запуска задания на выполнение. Подготовленный образ задания (контейнер) рассылается на все выделенные системой управления заданиями ВМ суперкомпьютера, по одному экземпляру образа на каждый выделенный ВМ. Далее на каждом ВМ выполняется привязка (конфигурирование) транспортной (управляющей) и высокоскоростной сетей к контейнеру, содержащему образ задания. Конфигурирование должно быть выполнено таким образом, чтобы размещенные на выделенных ВМ контейнеры задания образовали отдельную изолированную виртуальную сеть как для транспортной (управляющей), так и для высокоскоростной составляющих. При этом необходимо организовать сетевое взаимодействие заранее неизвестного состава ВМ посредством управляющей и высокоскоростных сетей для обеспечения возможности сетевого взаимодействия контейнеров в рамках одного вычислительного задания. После конфигурирования сетей происходит запуск процессов задания внутри контейнеров.

Сетевые средства Docker

Сеть Docker построена на модели CNM (Container Network Model). Модель разработана компанией Docker и описывает основные компоненты и порядок их взаимодействия, необходимые для организации сетевого взаимодействия с контейнерами. Реализацией CNM является библиотека libnetwork, которая позволяет пользователю создать свой собственный сетевой драйвер, если существующие не удовлетворяют его требованиям.

Контейнеры имеют доступ к разным типам сетей и могут подключаться к нескольким сетям одновременно. Docker имеет четыре встроенных драйвера для сетей:

bridge – связь устанавливается через мостовой интерфейс на физическом ВМ; у контейнеров, которые используют одинаковую сеть, есть своя собственная подсеть, и они могут передавать данные друг другу по умолчанию;

host – предоставляет контейнеру доступ к собственному пространству ВМ; контейнер будет «видеть» и использовать тот же интерфейс, что и ВМ;

macvlan – дает контейнерам прямой доступ к интерфейсу ВМ;

overlay – позволяет строить сети на нескольких компьютерах, использующих Docker, объединенных сетью.

Напомним, что сетевое взаимодействие контейнеров в рамках одного задания должно осуществляться как по управляющим и транспортным сетям (Ethernet), так и по высокоскоростным сетям, используемым в ходе вычислений (Infiniband, Omni-Path). Сетевые драйверы Docker нацелены на использование стека TCP/IP. В случае использования устройств высокоскоростной сети, настроенной для работы со стеком TCP/IP, можно организовать высокопроизводительные вычисления, используя стандартные средства Docker и не прибегая к дополнительным способам и средствам. Однако применение надстроек, обеспечивающих работу со стеком TCP/IP, над устройствами высокопроизводительной сети сильно повышает латентность, снижает производительность и в целом приводит к замедлению вычислений [1]. Если представлять пользовательские задания в виде контейнеров Docker, необходимо использовать высокопроизводительные сетевые устройства в «чистом» виде, без использования TCP/IP.

Docker позволяет добавлять в контейнер различные устройства, такие как графические карты, сетевые устройства и т.п. В рамках работы представляет интерес добавление поддержки устройств высокопроизводительной коммутируемой сети Infiniband. Для этого необходимо осуществить добавление двух устройств в момент запуска контейнера:

– *device /dev/Infiniband/rdma_cm;*

– *device /dev/Infiniband/uverbs0.*

При запуске контейнера с указанными параметрами внутри контейнера можно использовать Infiniband-устройство (при установленном в контейнере стеке ПО, необходимого для работы с технологией Infiniband).

В случае запуска нескольких контейнеров, использующих одни и те же устройства, на одном компьютере разделение времени доступа к устройствам будет обеспечиваться механизмами ядра управляющей ОС. В таком случае накладные расходы сводятся к минимуму и аналогичны задержкам, получаемым при запуске двух вычислительных заданий на одном ВМ [12].

Для динамической реконфигурации управляющей и транспортных сетей суперкомпьютера в рамках контейнеров Docker доступны следующие способы.

1. Использование Ethernet-сети физического ВМ.

Docker позволяет определить соответствие порта управляющего компьютера с открытым портом контейнера. При организации сети подобным образом удаётся получить доступ к контейнеру, используя IP-адрес ВМ и открытый на ВМ порт. Однако возможным становится получение доступа к контейнеру посредством любого ВМ, имеющего сетевой доступ к управляющему компьютеру такого контейнера. Для режима коллективного пользования необходимо обеспечить большую изоляцию вычислительных заданий, следовательно, рассматриваемый вариант неприменим.

2. Сети Docker типа overlay.

В версии Docker 1.12 [13] появилась новая опция, получившая название Swarm mode (режим роя). «Режим роя» позволяет объединить в кластер фоновые серверные процессы Docker, расположенные физически на разных узлах сети. При работе в таком режиме фоновый процесс Docker считает все ВМ единым контейнерным пространством. С «режимом роя» в Docker появился новый сетевой драйвер overlay. Контейнеры в данном случае называются службами. «Режим роя» позволяет производить запуск определенного количества контейнеров для одной службы, а также автоматически производит балансировку нагрузки между физическими компьютерами, выполняя равномерное распределение контейнеров по ВМ.

Для объединения контейнеров службы в единую подсеть необходимо создать сеть Docker с драйвером overlay, используя команду создания сети – *docker network create driver overlay my_net* (здесь my_net – произвольное имя сети).

Однако при таком подходе Docker не позволяет присоединять отдельные контейнеры к созданным подобным способом overlay-сетям. Использование служб Docker в рамках настоящей работы невозможно из-за отсутствия у них возможности предотвращения повышения пользователем своих привилегий внутри контейнера, поскольку команда создания службы docker service create не имеет соответствующего параметра запуска.

3. Использование готовых драйверов, созданных сообществом Docker. Оценке их применимости в рамках настоящей работы посвящен следующий раздел статьи.

Динамическое реконфигурирование сетей суперкомпьютера с помощью сторонних сетевых драйверов Docker

Docker является программным средством с открытым исходным кодом, разработчики предоставили API Docker для всех желающих, что обуславливает существование значительного количества разнообразных дополнений (плагинов).

Проблема объединения контейнеров, расположенных на различных физических ВМ, привела к созданию различных сетевых дополнений для Docker, одним из которых является Project Calico [14]. Этот драйвер позволяет объединять контейнеры в единую сеть без использования средств оркестрации контейнеров и «режима роя» Docker. Project Calico является проектом с открытым исходным кодом, отличается безопасностью и легкой масштабируемостью, изначально предназначался для организации сетей в облачных средах.

Альтернативой Project Calico может служить средство настройки сетей Flannel [15], разработанное для системы автоматизации развертывания, масштабирования и управления контейнеризированными приложениями Kubernetes [16]. Однако Kubernetes является хорошим решением в качестве основы для управления ресурсами в облачных платформах [17], требует полного контроля над вычислительными модулями и несовместим с суперкомпьютерными СУЗ. Кроме этого, в работе [18] показано, что Project Calico обладает наилучшей производительностью по сравнению с Flannel и сетями Docker типа overlay.

Для обеспечения возможности работы с сетями, использующими драйвер Project Calico, необходимо:

– настроить распределенное хранилище данных etcd [19];

– сконфигурировать фоновый серверный процесс Docker;

– запустить службу сетей calico;

– создать определенный диапазон IP-адресов, необходимый для создания подсетей.

Применение указанного подхода позволяет решить задачу по объединению заранее неизвестного количества контейнеров, расположенных на заранее неизвестных ВМ суперкомпьютера, в единую подсеть.

Архитектура сети с применением драйвера calico представлена на рисунке 2. Для работы драйвера calico требуется высоконадежное распределенное хранилище конфигурации etcd, настроенное на всех ВМ, где будет производиться запуск контейнеров, подключаемых к сети с использованием драйвера calico. Etcd представляет собой программный пакет, разработанный компанией CoreOS и предназначенный для хранения данных в формате «ключ-значение». Calico использует etcd для хранения информации о созданных диапазонах IP-адресов подсетей calico, а также для хранения соответствий имен контейнеров их IP-адресам и сетям.

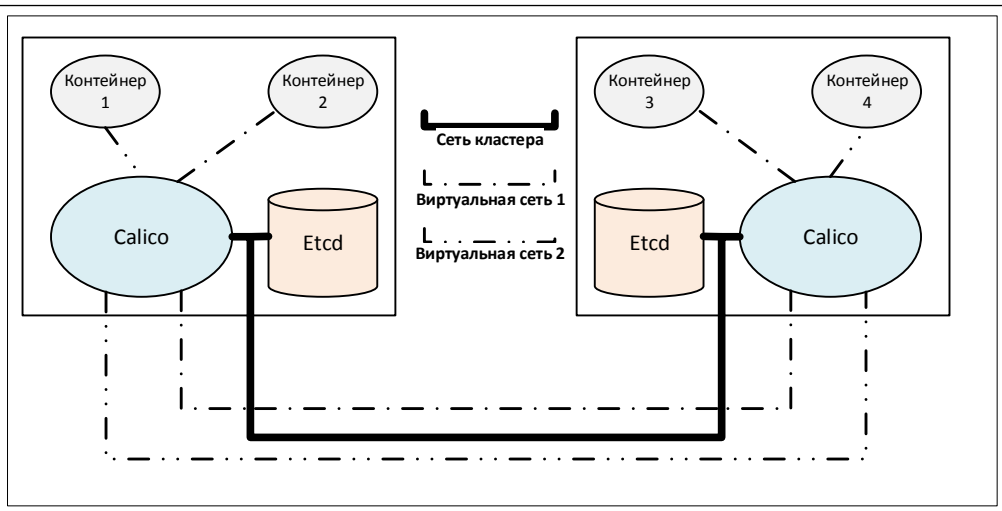

Рис. 2. Архитектура сети с применением драйвера calico

После установки на ВМ программных средств (Docker, etcd, calico), необходимых для создания сети с драйвером calico, для динамической конфигурации сети необходимо выполнение следующих действий:

– для рассматриваемого задания выделить множество ВМ;

– на первом из выделенных заданию ВМ создать Docker-сеть, используя драйвер calico (такая сеть становится доступной на всех ВМ, входящих в кластер etcd);

– для каждого ВМ, выделенного заданию, проверить доступность созданной сети и присоединить к ней запускаемый в рамках задания контейнер;

– с первого из выделенных заданию ВМ проверить доступность в созданной сети всех выделенных ВМ: если все ВМ доступны, завершить конфигурацию сети; если хотя бы один ВМ недоступен, прервать создание сети и аварийно завершить задание.

После создания сети адресация осуществляется между контейнерами по адресам типа «имя_контейнера.имя_сети». Контейнеры, присоединенные к одной сети, не имеют доступа к контейнерам другой сети, даже если их IP-адреса находятся в одном диапазоне [12]. Доступ к Docker-сети, использующей драйвер calico, осуществим только из контейнера, который был к этой сети присоединен, что автоматически определяет необходимый уровень изоляции представленного в виде набора контейнеров задания.

Макет суперкомпьютерной системы с представлением пользовательских заданий в виде контейнеров Docker

Для экспериментальной проверки рассмотренных в статье подходов авторами был подготовлен макет суперкомпьютерной системы, построенный на базе сегмента установленного в МСЦ РАН суперкомпьютера МВС-100К. Сегмент включал три вычислительных модуля HP ProLiant BL2x220c G5. Схема реализованного макета представлена на рисунке 3. Один из ВМ сегмента выступил в роли управляющей ЭВМ, остальные представляли ВМ решающего поля суперкомпьютера.

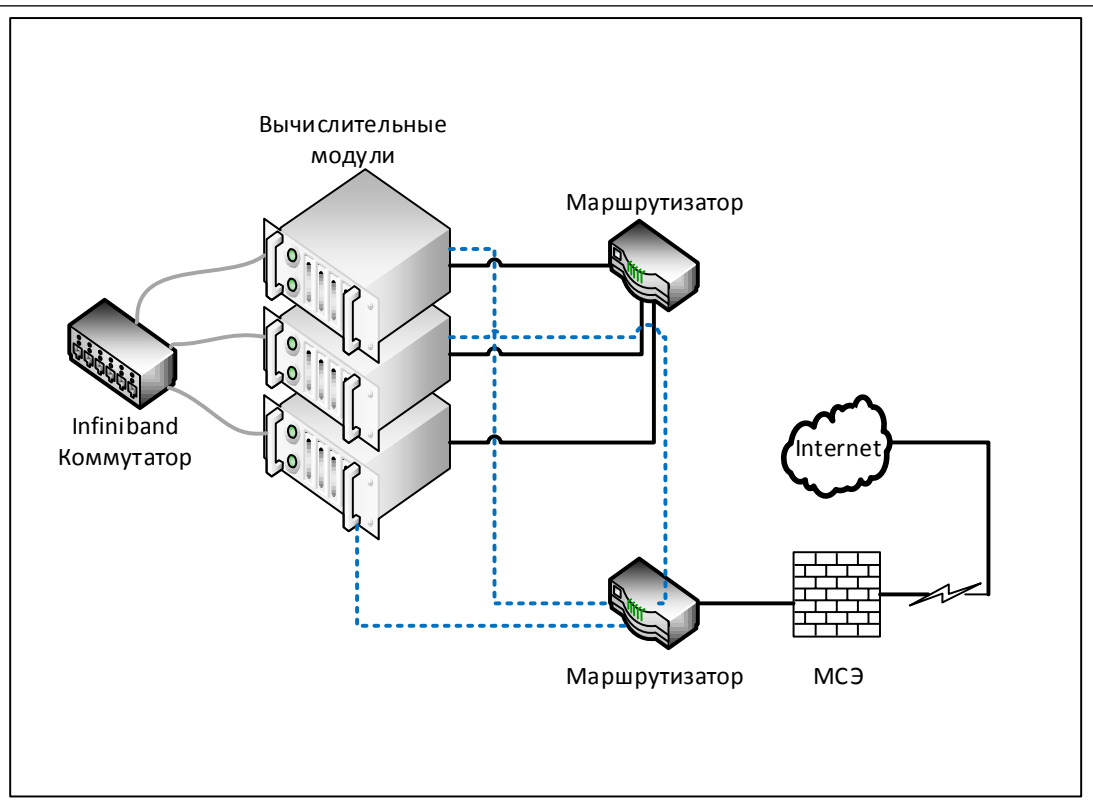

Рис. 3. Макет суперкомпьютерной системы

Каждый ВМ HP ProLiant BL2x220c G5 содержит в своем составе:

– два интерфейса Gigabit Ethernet;

– адаптер Infiniband Mellanox MT25204;

– два процессора Intel Xeon E5450 (4 ядра, работающих на частоте 3 ГГц);

– оперативную память 8 ГБ;
– встроенный жесткий диск 120 ГБ.

На каждый из ВМ было установлено следующее системное и инструментальное ПО:
– ОС CentOS 7.4;
– etcd (версия 3.2.15);
– docker community edition (версия 18.03);
– calicoctl (версия 0.22);
– модуль языка python paramiko (версия 2.4.1).

В качестве СУЗ на макете была использована СУППЗ, для которой авторами были разработаны два программных модуля автоматизации процедур запуска/остановки параллельных вычислительных заданий, представленных в виде Docker-контейнеров. Последовательность действий, выполняемых системой командных файлов СУППЗ при запуске вычислительного задания в виде контейнера, представлена на рисунке 4.

Часть действий выполняется на управляющей ЭВМ макета в специальном командном сценарии пролога СУППЗ, часть – на ВМ в специальном сценарии подготовки модуля. Командные файлы указанных сценариев доступны для модификации системному администратору СУППЗ, то есть процедура внесения в них изменений является стандартной и описана в документации СУППЗ.

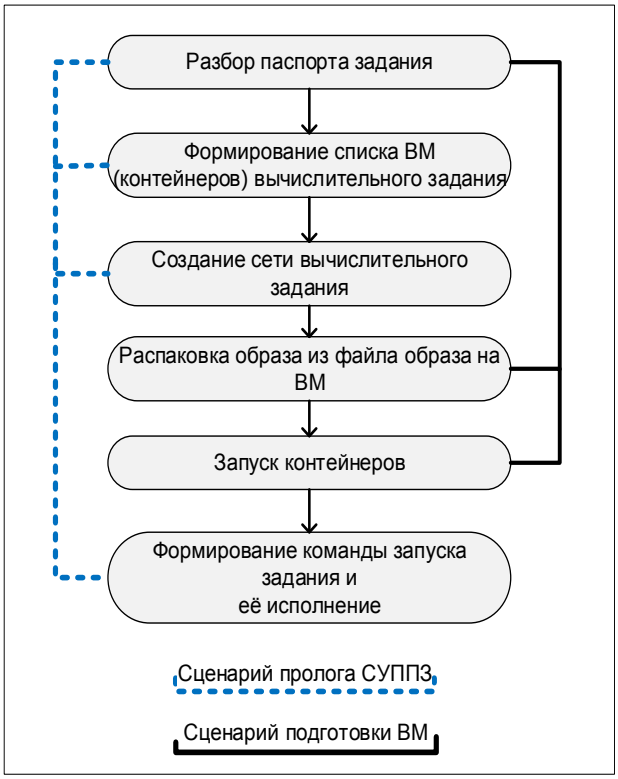

Рис. 4. Последовательность запуска вычислительного задания, представленного в виде Docker-контейнера

Сначала происходит считывание параметров из паспорта задания, содержащего информацию о запускаемом задании. Одним из параметров паспорта является имя файла образа контейнера вычислительного задания.

Далее на первом из списка выделенных заданию ВМ выполняется команда создания сети Docker, использующей драйвер calico. Название сети соответствует имени задания, которое является уникальным в СУППЗ, чем обеспечивается изоляция вычислительного задания.

Для запуска контейнеров вычислительного задания необходимо извлечь образ из файла. Извлечение происходит на каждом из выделенных ВМ.

Следующим шагом является формирование команд запуска контейнера, имя которого должно быть уникальным, по данному имени осуществляется сетевое взаимодействие. После формирования команды происходит запуск контейнера вычислительного задания с именем вида nodeHostname-taskID на каждом из выделенных ВМ.

Для запуска параллельной программы необходимо создать файл, содержащий имена контейнеров вычислительного задания. Сформированный файл, в котором указаны строки вида имя_контейнера.имя_сети_задания, помещается в домашний каталог пользователя, запустившего вычислительное задание.

Идентификаторы владельца файла изменяются на соответствующие пользователю для предоставления возможности чтения/изменения этого файла в контейнере от имени пользователя.

Одним из параметров паспорта задания является задаваемый пользователем шаблон команды запуска вычислительного задания, к которому добавляется сгенерированный на предыдущем шаге список сетевых имен контейнеров. Сформированная из шаблона команда исполняется в контейнере на первом из выделенных заданию ВМ. Как правило, эта команда является командой запуска параллельной программы в среде коммуникационной библиотеки MPI.

Далее вычислительное задание пользователя находится в стадии выполнения.

По истечении времени, выделенного СУППЗ на выполнение задания, происходит очистка выделенных ВМ от процессов задания. Последовательность действий по очистке модулей показана на рисунке 5.

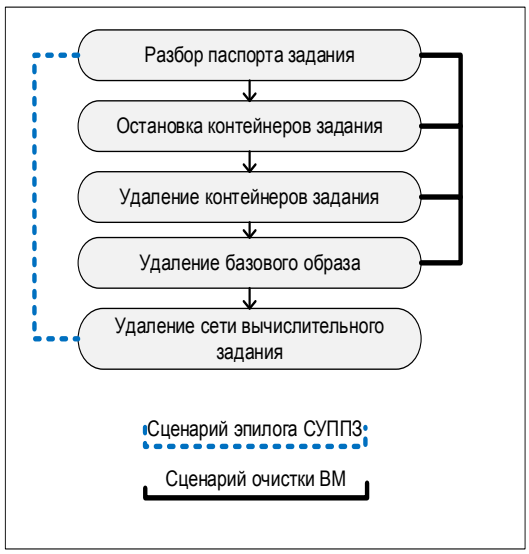

Рис. 5. Последовательность освобождения выделенных ресурсов после завершения вычислительного задания, представленного в виде Docker-контейнера

Часть действий выполняется на управляющей ЭВМ макета в специальном командном сценарии эпилога СУППЗ, часть – на ВМ в специальном сценарии очистки модуля.

Командные файлы указанных сценариев доступны для модификации системному администратору СУППЗ, процедура внесения в них изменений, как и для сценариев пролога и подготовки модулей, является стандартной и описана в документации СУППЗ.

На каждом из выделенных заданию ВМ выполняются остановка запущенного контейнера вычислительного задания, его удаление и извлечение имени образа, на базе которого был запущен контейнер. Для полной очистки ВМ от процессов пользовательского задания удаляется образ пользовательского контейнера, а после завершения и удаления запущенных контейнеров сеть Docker, созданная для завершившегося задания.

Заключение

Для решения задачи динамической реконфигурации сетей суперкомпьютера при запуске представленных в виде Docker-контейнеров пользовательских заданий авторами были исследованы ряд способов и средств. В ходе исследований выяснено, что стандартные сетевые средства Docker не удовлетворяют требованиям обеспечения необходимого уровня изоляции заданий. В результате был определен способ реконфигурации сетей, основанный на применении распределенного хранилища данных etcd и специального драйвера calico. Способ позволяет осуществить динамическую реконфигурацию сетей суперкомпьютера практически для любой системы управления заданиями. Авторы произвели практическую реализацию предложенного способа для отечественной СУППЗ, эксплуатируемой в качестве СУЗ на суперкомпьютерных системах МСЦ РАН и ИПМ им. М.В. Келдыша РАН. Результатом стал программно-аппаратный макет суперкомпьютерной системы, обеспечивающий обработку представленных в виде Docker-контейнеров заданий с динамической реконфигурацией сетей. Ближайшей перспективой работы является внедрение рассмотренного в статье способа на суперкомпьютерах МСЦ РАН, входящих в состав оборудования центра коллективного пользования вычислительными ресурсами МСЦ РАН.

Статья подготовлена в рамках выполнения государственного задания ФГУ ФНЦ НИИСИ РАН.

Литература

1. Баранов А.В., Николаев Д.С. Использование контейнерной виртуализации в организации высокопроизводительных вычислений // Программные системы: теория и приложения. 2016. Т. 7. № 1 (28). С. 117–134. DOI: 10.25209/2079-3316-2016-7-1-117-134.

2. Баранов А.В., Киселёв А.В., Старичков В.В., Ионин Р.П., Ляховец Д.С. Сравнение систем пакетной обработки с точки зрения организации промышленного счета // Научный сервис в сети Интернет: поиск новых решений: тр. Междунар. суперкомп. конф. (17–22 сентября 2012 г., Новороссийск). М.: Изд-во МГУ, 2012. С. 506–508. URL: http://agora.guru.ru/abrau2012/pdf/506.pdf (дата обращения: 12.04.2018).

3. Шабанов Б.М., Овсянников А.П., Баранов А.В., Аладышев О.С., Киселёв Е.А., Жуков Я.О. Методы управления параллельными заданиями суперкомпьютера, требующими развертывания отдельных программных платформ и виртуализации сетей // Суперкомпьютерные дни в России: тр. Междунар. конф. (25–26 сентября 2017 г., Москва). М.: Изд-во МГУ, 2017. С. 616– 627. URL: https://elibrary.ru/download/elibrary_30632214_33009164.pdf (дата обращения: 05.07.2018).

4. Подорожный И.В., Светличный А.Н., Подлеснов А.В. Введение в контейнеры, виртуальные машины и docker // Молодой ученый. 2016. № 19. С. 49–53. URL: https://moluch.ru/archive/123/33873/ (дата обращения: 02.07.2018).

5. Щапов В.А., Денисов А.В., Латыпов С.Р. Применение контейнерной виртуализации Docker для запуска задач на суперкомпьютере // Суперкомпьютерные дни в России: тр. Междунар. конф. (26–27 сентября 2016 г., Москва). М.: Изд-во МГУ, 2016. С. 505–511. URL: https://elibrary.ru/download/elibrary_27206414_31832253.pdf (дата обращения: 05.07.2018).

6. Щапов В.А., Латыпов С.Р. Способы запуска задач на суперкомпьютере в изолированных окружениях с применением технологии контейнерной виртуализации Docker // Науч.-технич. вестн. Поволжья. 2017. № 5. С. 172–177.

7. МСЦ РАН – Вычислительные системы. URL: http://www.jscc.ru/scomputers.html (дата обращения: 04.07.2018).

8. Вычислительные ресурсы ИПМ им. М.В. Келдыша РАН. URL: http://www.kiam.ru/MVS/resourses/ (дата обращения: 04.07.2018).

9. СУППЗ. URL: http://suppz.jscc.ru/ (дата обращения: 03.07.2018).

10. Система управления прохождением параллельных заданий (СУППЗ). Руководство программиста (пользователя). URL: http://www.jscc.ru/informat/СУППЗ_Руководство_пользователя.pdf (дата обращения: 03.07.2018).

11. Intel® Omni-Path Architecture (Intel® OPA) Driving Exascale Computing and HPC with Intel. URL: https://www.intel.ru/content/www/ru/ru/high-performance-computing-fabrics/omni-path-driving-exascale-computing.html (дата обращения: 02.07.2018).

12. Николаев Д.С., Корнеев В.В. Использование механизмов контейнерной виртуализации в высокопроизводительных вычислительных комплексах с системой планирования заданий Slurm // Программная инженерия. 2017. Т. 8. № 4. С. 157–160.

13. Docker CE release notes. URL: https://docs.docker.com/release-notes/docker-ce/ (дата обращения: 02.07.2018).

14. Project Calico – Secure Networking for the Cloud Native Era. URL: https://www.projectcalico.org (дата обращения: 02.07.2018).

15. GitHub – coreos/flannel: flannel is a network fabric for containers, designed for Kubernetes. URL: https://github.com/coreos/flannel (дата обращения: 04.07.2018).

16. Production-Grade Container Orchestration – Kubernetes. URL: https://kubernetes.io (дата обращения: 05.07.2018).

17. Coullon H., Pertin D., Perez C. Production Deployment Tools for IaaSes: an Overall Model and Survey. 2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud), 2017, pp. 183–190. DOI: 10.1109/FiCloud.2017.51.

18. Zeng H., Wang BS., Deng WP., Zhang, WQ. Measurement and Evaluation for docker container networking. Proc. 2017 Intern. Conf. on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2017, pp. 105–108. DOI: 10.1109/CyberC.2017.78.

19. Using etcd. URL: https://coreos.com/etcd/ (дата обращения: 02.07.2018).