A. V. Baranov, D. S. Nikolaev, The use of container virtualization in the organization of high-performance computing, *Program Systems: Theory and Applications*, 2016, Volume 7, Issue 1, 117–134

ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ №1(28), 2016, с. 117–134

УДК 004.457+004.031.2+004.382.2

А. В. Баранов, Д. С. Николаев

Использование контейнерной виртуализации в организации высокопроизводительных вычислений

Аннотация. Статья посвящена вопросу применимости контейнерной виртуализации при организации высокопроизводительных вычислений с точки зрения изоляции пользовательских заданий.

В статье приводятся результаты экспериментов по оценке накладных расходов на организацию контейнерной виртуализации, степени взаимного влияния заданий и обеспечения защиты от несанкционированного доступа.

Ключевые слова и фразы: высокопроизводительные вычисления, контейнерная виртуализация, LXC, Docker, накладные расходы на виртуализацию.

Введение

В настоящей работе под системой высокопроизводительных вычислений будем понимать вычислительную установку типовой кластерной архитектуры, состоящую из нескольких вычислительных модулей, объединенных одной или несколькими высокоскоростными сетями. Вычислительный модуль такой системы представляет собой самостоятельный компьютер, оснащённый, как правило, несколькими центральными процессорами (двумя или более), собственными оперативной и дисковой памятями. Очень часто вычислительные модули имеют в своем составе сопроцессоры — ускорители на базе графических процессоров, ПЛИС или многоядерных мультитредовых решений (Intel Xeon Phi). Важно, что каждый вычислительный модуль управляется отдельным экземпляром операционной системы (как правило, Linux).

Управление вычислительными ресурсами и пользовательскими заданиями в современных системах высокопроизводительных вычислений осуществляет специальное программное обеспечение — система пакетной обработки (СПО). СПО обеспечивает коллективный доступ

пользователей к супер-ЭВМ, принимает входной поток различных заданий от разных пользователей, планирует очереди заданий, выделяет необходимые для выполнения задания вычислительные ресурсы и освобождает их после завершения задания.

Опыт эксплуатации современных СПО позволяет выделить два негативных момента. Во-первых, СПО ориентированы на ведение разных очередей для разных типов так называемых стандартных заданий, которые выполняются в развёрнутой на установке программной среде и не требуют внесения изменений в процесс конфигурации вычислительных модулей. В этом случае пользователи попадают в зависимость от установленного на кластере ПО. Задания должны создаваться с учетом набора ПО, вспомогательных библиотек и их версий на конкретном кластере, в связи с чем возникают сложности с переносом заданий на другой кластер, с воспроизводимостью полученных результатов вычислений, администрированием самого кластера. В последние годы в суперкомпьютерных центрах всё чаще появляются нестандартные задания, предъявляющие особые требования к ресурсам. Например, подобным заданиям могут потребоваться специфическое окружение, особые программные пакеты и лицензии, то есть своя программная среда. Для СПО возникает новая задача — обеспечение возможности автоматического развёртывания различных программных платформ для разных нестандартных заданий на одной вычислительной установке.

Во-вторых, с ростом числа ядер в современных процессорах увеличивается фрагментация решающего поля вычислительной установки. Связано это с тем, что, как правило, при выделении ресурсов для пользовательского задания СПО придерживаются принципа неделимости вычислительного модуля системы, т.е. модуль выделяется заданию либо целиком, либо не выделяется вовсе. Не допускается распределение на один модуль двух разных заданий, тем более, разных пользователей. Указанный принцип применяется с целью исключения взаимного влияния пользовательских заданий или, другими словами, обеспечения изоляции заданий. Отметим два важных аспекта изоляции заданий.

(1) Защита от преднамеренного или непреднамеренного несанкционированного доступа, подразумевающая:
   - защиту пользовательских данных;
   - невозможность монопольного захвата вычислительных ресурсов — процессоров, памяти, портов коммуникационной среды.

(2) Исключение конкуренции за вычислительные ресурсы между процессами заданий, в результате которой снижается производительность вычислений.

Принцип неделимости вычислительного модуля обеспечивает простую и надёжную изоляцию заданий, но платой за это служит снижение утилизации ресурсов за счёт неизбежной фрагментации. Например, если заданию требуется для выполнения 20 процессорных ядер, то в системе с 8-ядерными вычислительными модулями для задания будет выделено 3 модуля или 24 процессора. Очевидно, 4 (около 16%) процессора во время выполнения задания будут простаивать. Как уже было отмечено, подобная фрагментация с увеличением числа ядер на один процессор будет возрастать.

Одним из подходов, позволяющих преодолеть рассмотренные негативные моменты, является использование средств виртуализации, которые позволяют запускать на одном вычислительном узле полностью изолированные друг от друга задания пользователей. Однако, несмотря на всю привлекательность средств виртуализации, они пока не нашли широкого применения в области высокопроизводительных вычислений. Это связано с тем, что виртуализация привносит опредёленные накладные расходы, такие как падение производительности, дополнительное потребление памяти, время на развёртывание виртуальных машин и запуск задания в них, невозможность обеспечить прямой доступ к аппаратным ускорителям и ряд других ограничений. Например, в работе [1] показано, что накладные расходы на виртуализацию при выполнении высокопроизводительных заданий составляют 5–10%, при этом время развёртывания виртуальных машин на вычислительных модулях — от 10 минут.

В настоящей работе рассматривается альтернативный подход — использование сравнительно недавно появившегося в ядре Linux механизма контейнерной виртуализации LinuX Containers (LXC).

1. Контейнерная виртуализация LXC

Средства виртуализации по типу реализации подразделяются на программные, аппаратные и виртуализации на уровне операционной системы. Первые два типа подразумевают наличие слоя гипервизора между заданием пользователя и операционной системой вычислительного узла. Виртуализация на уровне операционной системы позволяет запускать изолированные и безопасные виртуальные

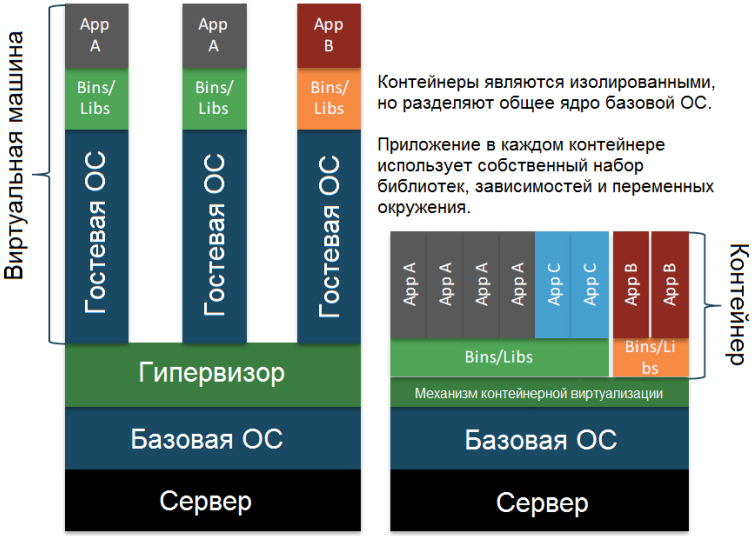


Рис. 1. Различие между виртуализацией с использованием гипервизора и контейнерной виртуализацией

машины на одном физическом узле, используя ядро базовой операционной системы.

Виртуализация на уровне операционной системы — метод виртуализации, при котором ядро операционной системы поддерживает несколько изолированных экземпляров пространства пользователя. Эти экземпляры (часто называемые контейнерами или зонами) с точки зрения пользователя полностью идентичны реальному вычислительному модулю. Ядро обеспечивает полную изолированность контейнеров, поэтому программы из разных контейнеров не могут воздействовать друг на друга. Различие между виртуализацией с использованием гипервизора и контейнерной виртуализацией показано на рис. 1.

LXC [2] — система виртуализации на уровне операционной системы для запуска нескольких изолированных экземпляров ОС Linux на одном компьютере. LXC не использует виртуальные машины, а создает виртуальное окружение с собственным пространством процессов и сетевым стеком. Все экземпляры LXC используют один экземпляр ядра ОС. LXC основана на технологии ядра Linux под названием

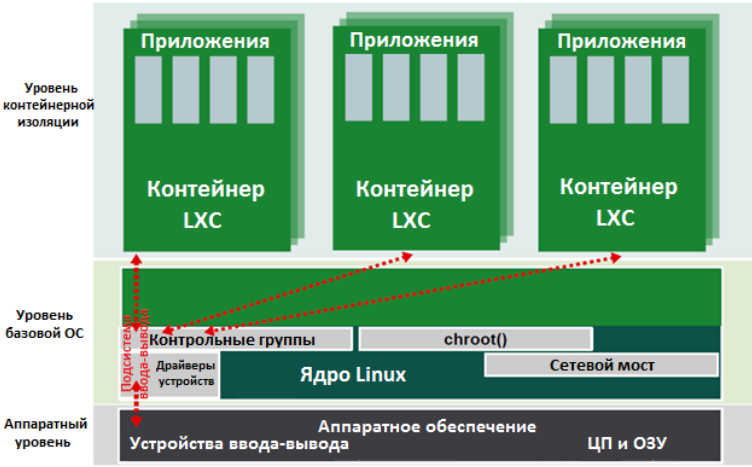


Рис. 2. Технология контейнерной виртуализации LXC

«контрольные группы» (cgroups, добавлено в версии ядра 2.6.29) с использованием механизма изоляции «пространство имён» (namespaces). Часто технология контейнерной виртуализации (см. рис. 2) рассматривается как улучшенная реализация механизма «песочницы» chroot.

Для автоматизации развёртывания и управления приложениями в среде контейнерной виртуализации используется специальное программное обеспечение Docker [3]. Docker позволяет «упаковать» приложение со всем его окружением и зависимостями в контейнер, который может быть перенесён на любую Linux-систему с поддержкой механизма контрольных групп в ядре, а также предоставляет среду по управлению контейнерами.

В состав программного средства Docker входят сервер контейнеров, клиентские средства, позволяющие с помощью командного интерфейса управлять образами и контейнерами, а также API управления контейнерами. Сервер контейнеров обеспечивает полную изоляцию запускаемых на вычислительном модуле контейнеров на уровне файловой системы (у каждого контейнера собственная корневая файловая система), на уровне процессов (процессы имеют доступ только к собственной файловой системе контейнера, а ресурсы разделены средствами LXC), на уровне сети (каждый контейнер имеет доступ только к привязанному к нему сетевому пространству имён и соответствующим виртуальным сетевым интерфейсам). Набор клиентских

средств позволяет запускать процессы в новых контейнерах, останавливать и запускать контейнеры, приостанавливать и возобновлять процессы в контейнерах. Отдельный набор команд позволяет осуществлять мониторинг запущенных процессов.

Новые образы можно создавать из специального сценарного файла, предусмотрена возможность записать все сделанные в контейнере изменения в новый образ. Все команды могут работать как с docker-сервером локальной системы, так и с любым доступным по сети сервером docker.

С помощью ПО Docker Registry имеется возможность создания репозитория для хранения готовых преднастроенных образов контейнеров. Таким образом, пользователь может использовать контейнер из доверенного репозитория с требуемым для выполнения задания набором библиотек и лицензий.

2. Условия проведения эксперимента

2.1. Состав экспериментальных стендов

Авторами был проведён ряд экспериментов, основной целью которых являлась оценка, насколько контейнеры исключают влияние одного задания на другое и обеспечивают изоляцию заданий. Эксперименты проводились на двух испытательных стендах, на одном из которых в качестве коммуникационной сети (т.е. сети информационных обменов задания) использовался Ethernet, на другом — Infiniband (рис. 3). Оба стенда имели одинаковую структуру: два идентичных вычислительных модуля и узел доступа.

В стенде с сетью Ethernet в каждом вычислительном модуле установлен 4-ядерный процессор Intel Core i7-2600 3.40GHz, 8 ГБ оперативной памяти, два сетевых адаптера Gigabit Ethernet. Для корректного планирования загрузки процессора технология Intel Hyper-Threading была отключена.

На стенде с сетью Infiniband в каждом вычислительном модуле установлены по два 4-ядерных процессора Intel Xeon X5472 3.00GHz, 32 ГБ оперативной памяти, два сетевых адаптера Gigabit Ethernet и адаптер высокоскоростной сети InfiniBand Mellanox MT25204.

На обоих стендах на вычислительных модулях установлена ОС CentOS 7.1 с версией ядра Linux 3.10.0-229.14.1.el7.x86_64, компилятор gcc версии 4.8.3, библиотека OpenMPI версии 1.6.4, программное

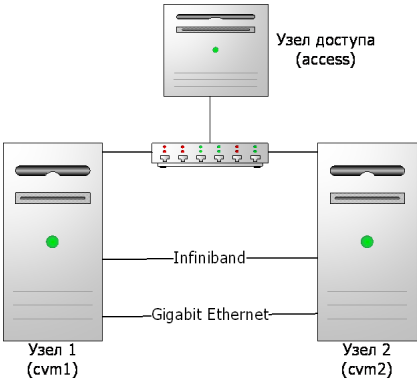


Рис. 3. Схема экспериментальных стендов с коммуникационными сетями Ethernet и Infiniband

обеспечение для автоматизации развёртывания и управления приложениями в среде виртуализации на уровне операционной системы Docker версии 1.7.1.

При проведении вычислительных экспериментов в контейнерах использовались те же версии ПО, что и на основных вычислительных модулях. На каждом вычислительном модуле создавалось по два контейнера, в контейнеры монтировалась общая папка, расположенная на узле доступа.

В стенде на базе сети Ethernet контейнерам назначалось по два ядра центрального процессора, и выделялся 1 ГБ оперативной памяти. Для взаимной видимости контейнеров с разных вычислительных модулей были внесены изменения в сетевой стек, создаваемый ПО Docker по умолчанию. На каждом вычислительном модуле был создан программный сетевой мост, при этом IP-адреса созданных мостов принадлежали одной подсети и были взаимно видимы. Контейнеры создавались без предварительно настроенного сетевого интерфейса. После создания контейнера создавался виртуальный сетевой интерфейс, который подключался к программному сетевому мосту вычислительного модуля и назначался контейнеру. IP-адрес виртуального сетевого интерфейса контейнера назначался из подсети, входящей в подсеть сетевого моста. Такая конфигурация сетевого стека позволяет контейнерам, расположенным на разных вычислительных модулях, осуществлять сетевые соединения.

В стенде на базе сети Infiniband контейнеры запускались в привилегированном режиме (ключ `-privileged`), в этом режиме контейнеру доступны все устройства на вычислительном модуле. Каждому контейнеру было выделено 14ГБ оперативной памяти и назначено по два процессорных ядра таким образом, чтобы контейнер использовал ядра на одном физическом сокете процессора. Сеть Infiniband была разделена между процессами на уровне драйвера ядра.

2.2. Тестовые задания

В качестве тестовых заданий использовался стандартный набор тестов NAS Parallel Benchmark [4] версии 3.3 (BT, CG, EP, FT, IS, LU, MG, SP). Особенности названных тестов с точки зрения их требований к вычислительным возможностям модулей следующие.

Ключевым моментом теста BT является эффективность с точки зрения общего потребления простых арифметических операций.

Ключевым моментом теста CG является оценка скорости передачи данных при отсутствии какой-либо регулярности.

Ключевыми моментами теста EP являются:

- оценка максимальной производительности кластера при операциях с плавающей точкой;
- минимальные межпроцессорные взаимодействия.

Ключевыми моментами теста FT являются:

- большое количество действий, оказывающих большую нагрузку на сеть;
- оценка скорости перемещения массивов данных.

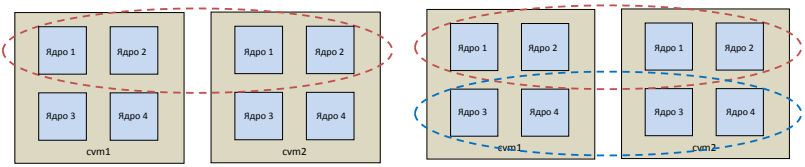
Ключевыми моментами теста IS являются:

- сильное влияние начального распределения чисел в памяти;
- оценка работы с общей памятью.

Ключевым моментом теста LU является критичность ко времени передачи небольших объёмов данных между модулями.

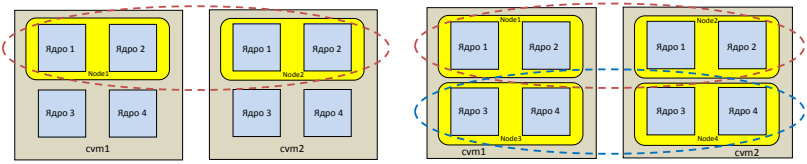
Ключевым моментом теста MG является оценка скорости передачи как длинных, так и коротких данных.

Ключевым моментом теста SP является обеспечение оптимальной загрузки сети.



(a) Схема № 1 —1 задание (b) Схема № 2 —2 задания

Рис. 4. Размещение заданий без контейнеров с использованием по 2 ядра с каждого вычислительного модуля



(a) Схема № 3 — 1 задание (b) Схема № 4 — 2 задания

Рис. 5. Размещение заданий в двухъядерных контейнерах

2.3. Схемы проведения экспериментов для оценки взаимного влияния заданий

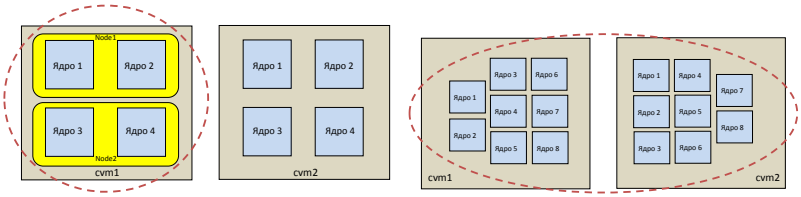
В процессе экспериментов моделировались следующие различные схемы размещения заданий. Схема № 1 (рис. 4(a)) и схема № 2 (рис. 4(b)) используются для определения степени взаимного влияния двух заданий, выполняющихся без контейнеров.

В схеме № 3 (рис. 5(a)) и схеме № 4 (рис. 5(b)) для выполнения заданий используются двухъядерные контейнеры node1-node4, схемы позволяют определить степень взаимного влияния двух заданий, выполняющихся в контейнерах, а также в сравнении со схемами №№ 1-2 оценить долю накладных расходов, привносимых контейнерами.

Схема № 5 (рис. 6(a)) позволяет оценить сетевые потери в сравнении со схемой № 3.

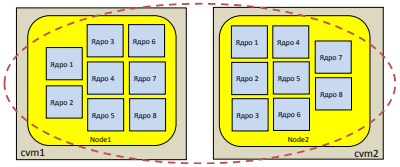
2.4. Схемы проведения экспериментов для оценки накладных расходов на контейнерную виртуализацию

Сравнение схем № 6 (рис. 6(b)) и № 7 (рис. 6(c)) позволяет оценить степень накладных расходов, вносимых контейнерами в организацию



(a) Схема № 5 — в контейнере на одном модуле

(b) Схема № 6 — без контейнеров на двух модулях



(c) Схема № 6 — в контейнерах на двух модулях

Рис. 6. Разделение заданий без дробления вычислительных модулей

вычислений. Эксперименты проводились только на стенде с коммуникационной сетью Infiniband, при этом задействовались все имеющиеся на вычислительных модулях процессорные ядра.

Накладные расходы на контейнерную виртуализацию будут определяться разностью во времени выполнения тестовых заданий по схеме № 6 и по схеме № 7.

Кроме накладных расходов времени выполнения заданий, необходимо оценить время развёртывания подготовленных образов контейнеров на вычислительных модулях. Здесь имеет значение порядок работы с контейнерами, включающий три этапа.

(1) Подготовка (создание) образа контейнера. На данном этапе пользователь или администратор создает контейнер и наполняет его необходимым содержимым. Время подготовки контейнера не входит в накладные расходы и не измеряется.
(2) Загрузка образа контейнера на вычислительный модуль. Если контейнер создан администратором и применяется для группы (или всех) пользователей, то этап загрузки осуществляется один раз, после чего загруженный контейнер используется многими заданиями. Если контейнер создан пользователем, контейнер

будет загружаться всякий раз, когда пользователь запускает своё задание. Время загрузки образа контейнера очень важно и должно обязательно измеряться и учитываться.

(3) Запуск контейнера на вычислительном модуле. Запуск производится всякий раз при старте использующего контейнер задания.

(4) Удаление загруженного контейнера после окончания выполнения задания. Удаление производится только, если контейнер был создан и загружен пользователем. В случае, если результаты работы задания остаются в контейнере, перед удалением его необходимо предварительно сохранить.

Загрузка образа контейнера производилась из общей папки, расположенной на узле доступа, по сети Gigabit Ethernet. Для уменьшения влияния NFS-кэширования, загрузка образа на вычислительный узел производилась сразу после размещения файла в общей папке. Для оценки времён загрузки и запуска контейнера использовался заранее созданный образ объёмом 1,2 ГБ.

3. Результаты экспериментов

3.1. Оценка взаимного влияния заданий

Результаты выполнения тестов в соответствии со схемами №№ 1-5 на экспериментальном стенде с использованием коммуникационной сети Ethernet представлены на рис. 7.

По результатам тестов видно, что контейнерная виртуализация оказывает значительное влияние на время выполнения заданий, подразумевающих интенсивный сетевой обмен между узлами. Отчасти это можно объяснить использованием стандартных сетевых адаптеров Gigabit Ethernet. Наилучший результат был достигнут при 5-й схеме размещения задания, т.е. задание выполнялось в двух контейнерах, размещенных на одном вычислительном узле при отсутствии сетевого обмена между узлами. Кроме этого, со схемой № 3 был проведён дополнительный эксперимент, в котором используемый сетевой интерфейс напрямую передавался контейнеру, минуя программный сетевой мост. Результаты этого эксперимента полностью совпали с приведёнными на рис. 7, из чего можно сделать вывод, что для схемы № 3 и сети Ethernet программный сетевой мост не вносит значимых задержек при сетевом обмене между контейнерами на разных вычислительных узлах.

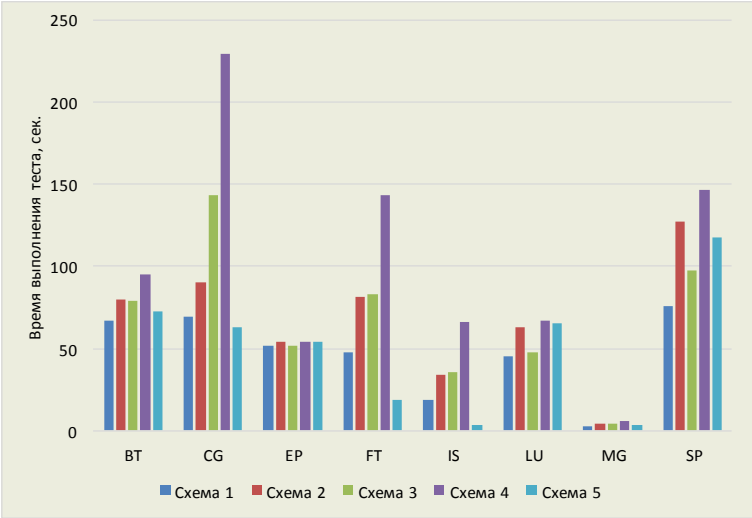


Рис. 7. Результаты выполнения тестов NPB 3.3 на экспериментальном стенде с коммуникационной сетью Ethernet

Результаты выполнения тестов в соответствии со схемами №№ 1-5 на экспериментальном стенде с использованием коммуникационной сети Infiniband представлены на рис. 8.

Как видно из рис. 8, при использовании Infiniband как взаимное влияние заданий (сравнение схемы № 3 и схемы № 4), так и накладные расходы от использования контейнеров (сравнение схемы № 1 и схемы № 3) меньше по сравнению с сетью Ethernet. Авторы относят высокие накладные расходы Ethernet на счёт протокола TCP/IP, использованного при организации сети информационных обменов на стенде с Ethernet.

Результаты экспериментов показывают, что, как в случае коммуникационной сети Ethernet, так и в случае коммуникационной сети Infiniband, присутствует существенное взаимное влияние заданий. Использование контейнеров не устраняет это влияние, следовательно, о возможности разделения вычислительного модуля между заданиями при их планировании в СПО говорить пока рано.

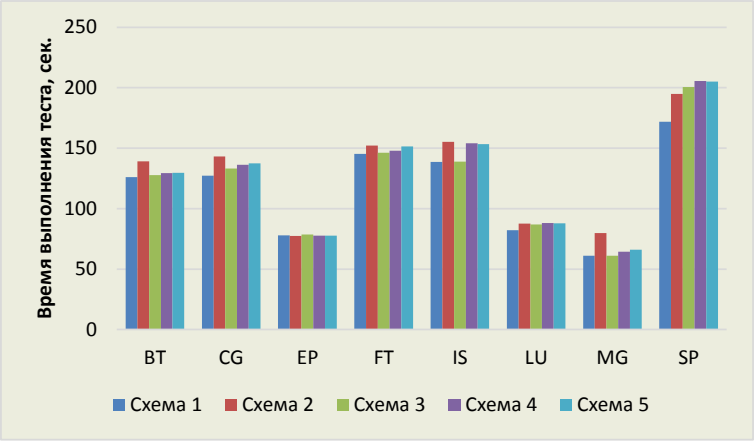


Рис. 8. Результаты выполнения тестов NPB 3.3 на экспериментальном стенде с коммуникационной сетью Infiniband

3.2. Оценка накладных расходов на контейнерную виртуализацию

Результаты выполнения тестов в соответствии со схемами №№ 6-7 на экспериментальном стенде с использованием коммуникационной сети Infiniband представлены на рис. 9. Максимальный объём оперативной памяти потребовался для выполнения теста IS и составил порядка 13 ГБ — это менее половины объёма оперативной памяти вычислительного узла (32 ГБ). Видно, что влияние контейнеров не превышает погрешности измерений, что говорит о возможности использования контейнеров для развёртывания пользовательских программных платформ в качестве нестандартных заданий.

Время загрузки на вычислительный модуль образа контейнера объёмом 1,2 ГБ составило 115 секунд, время запуска контейнера — 2 секунды. Время удаления корректно остановленного контейнера (внутри контейнера процессы задания завершены, и выполнена команда exit) составило 3 секунды. Время сохранения на общем сетевом ресурсе и удаления корректно остановленного контейнера составило 145 секунд.

Сравнивая полученные результаты с результатами работы [1], можно утверждать, что контейнерная виртуализация привносит значительно меньшие накладные расходы по сравнению с виртуализацией

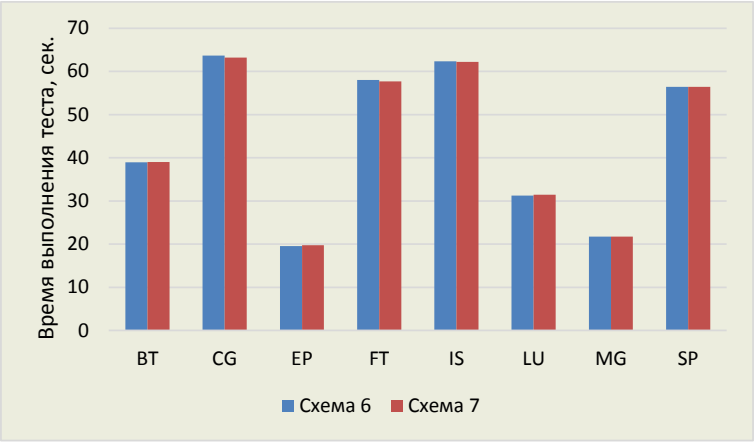


Рис. 9. Результаты выполнения тестов NPB 3.3 на экспериментальном стенде с коммуникационной сетью Iinfiniband с использованием всех процессорных ядер

на базе гипервизоров.

3.3. Оценка возможностей контейнеров по защите от несанкционированного доступа при выполнении заданий

Выполнение задания внутри контейнера инициируется отдельной командой, которая специфицируется при запуске контейнера. Команда инициализации задания может быть запущена как от имени пользователя, так и от имени суперпользователя. Если контейнер подготовлен администратором и включен в репозиторий доверенных контейнеров, то при условии запуска команды инициализации задания от имени пользователя практически исключается возможность запуска пользователем внутри контейнера процесса с правами суперпользователя (привилегированного процесса). Если контейнер был подготовлен самим пользователем, такой возможности исключать нельзя.

Проведённые авторами эксперименты показали следующие результаты.

(1) Для запуска контейнера от имени пользователя, последний обязательно должен быть включен в группу docker. Администратор должен внимательно отслеживать права группы docker, чтобы

исключить преднамеренный или непреднамеренный доступ пользователя к не принадлежащим ему данным.

(2) При запуске контейнера указывается папка, которая будет смонтирована в качестве домашнего каталога пользователя, причем монтируемая папка может располагаться на сетевом диске. Поскольку при запуске контейнера используется системный вызов chroot, для процессов контейнера гарантируется невозможность выйти из домашнего каталога пользователя. Если контейнер подготовлен пользователем, и последнему удалось запустить привилегированный процесс внутри контейнера, то обращение к смонтированной папке будет осуществляться с правами суперпользователя. Для предотвращения несанкционированного доступа сетевая файловая система должна запрещать доступ к домашним каталогам пользователей от имени суперпользователя.

(3) Не было обнаружено способа, с помощью которого пользователь изнутри контейнера смог бы преодолеть ресурсные ограничения (по процессорному времени и памяти), заданные для контейнера.

(4) Авторами моделировалась ситуация «зависания» процессов с максимальным потреблением процессорного времени и ситуация бесконечного размножения процессов в контейнере. В обеих ситуациях завершение контейнера гарантированно приводило к завершению всех процессов внутри него и освобождению занятых ресурсов.

Таким образом, контейнеры способны обеспечить достаточную защиту от несанкционированного доступа при условии соблюдения администратором следующих рекомендаций:

- запуск контейнера от имени пользователя с отслеживанием прав группы docker;
- монтирование домашних каталогов пользователей с запретом доступа суперпользователя к сетевой файловой системе.

Заключение

Результаты проведённых авторами экспериментов позволяют сделать следующие выводы.

(1) Контейнерная виртуализация может быть использована при организации высокопроизводительных вычислений как средство развёртывания специализированных программных платформ, в т.ч.

сформированных пользователем. Накладные расходы на контейнерную виртуализацию при развёртывании программной платформы существенно меньше, чем на виртуализацию с помощью гипервизора.

(2) Накладные расходы на контейнерную виртуализацию существенно возрастают, если сеть информационных обменов использует стек протоколов TCP/IP.

(3) На сегодняшнем этапе развития контейнеры не способны исключить взаимное влияние заданий, разделяющих один и тот же вычислительный модуль, применение контейнеров не решает проблемы фрагментации вычислительных ресурсов при коллективном доступе.

Благодарности. *Авторы искренне благодарны Виктору Владимировичу Корнееву и Олегу Сергеевичу Аладышеву за предложенные идеи по выбору и развитию направления исследований.*

Список литературы

[1] О. С. Аладышев, А. В. Баранов, Р. П. Ионин, Е. А. Киселёв, В. А. Орлов. «Сравнительный анализ вариантов развертывания программных платформ для высокопроизводительных вычислений», *Вестник УГАТУ*, **18**:3 (2014), с. 295–300, URL: http://journal.ugatu.ac.ru/index.php/vestnik/article/view/1101 ↑ 119,129

[2] М. Хэлсли. *LXC: Контейнерные утилиты Linux*, IBM developerWorks, 14.07.2009, URL: https://www.ibm.com/developerworks/ru/library/l-lxc-containers/l-lxc-containers-pdf.pdf ↑ 120

[3] *Docker user guide*, URL: https://docs.docker.com/userguide/ ↑ 121

[4] В. П. Гергель. «NAS Parallel Benchmarks», *Технологии построения и использования кластерных систем*, Учебный курс Национального открытого университета «ИНТУИТ» (дата обновления 18.10.2009), URL: http://www.intuit.ru/studies/courses/542/398/lecture/9173?page=3#sect9 ↑ 124

Рекомендовал к публикации *Программный комитет*
Четвёртого национального суперкомпьютерного форума ***НСКФ-2015***

Об авторах:

Антон Викторович Баранов

Ведущий научный сотрудник МСЦ РАН, к.т.н., доцент. Области научных интересов: организация высокопроизводительных вычислений, планирование заданий и управление вычислительными ресурсами в суперкомпьютерах, технологии виртуализации и облачных вычислений

e-mail: antbar@mail.ru

Дмитрий Сергеевич Николаев

Научный сотрудник МСЦ РАН. Области научных интересов: технологии виртуализации вычислительных ресурсов и их применение в высокопроизводительных вычислениях

e-mail: dmitry.s.nikolaev@gmail.com

Пример ссылки на эту публикацию:

А. В. Баранов, Д. С. Николаев. «Использование контейнерной виртуализации в организации высокопроизводительных вычислений», *Программные системы: теория и приложения*, 2016, **7**:1(28), с. 117–134.

URL: http://psta.psiras.ru/read/psta2016_1_117-134.pdf

Anton Baranov, Dmitrii Nikolaev. *The use of container virtualization in the organization of high-performance computing.*

Abstract. The article focuses on the applicability of operating-system-level virtualization (Linux containers) in the organization of high-performance computing in terms of the isolation of user tasks. The results of experiments evaluate the overhead cost of operating-system-level virtualization, the degree of mutual influence of user tasks and protection from unauthorized access. (*In Russian*).

Key words and phrases: high-performance computing, operating-system-level virtualization, Linux containers, LXC, Docker, overhead cost of virtualization.

References

[1] O. S. Aladyshev, A. V. Baranov, R. P. Ionin, Ye. A. Kiselyov, V. A. Orlov. "Comparative analysis of variants of deployment of program platforms for high performance computing", *Vestnik UGATU*, **18**:3 (2014), pp. 295–300 (in Russian), URL: http://journal.ugatu.ac.ru/index.php/vestnik/article/view/1101

[2] M. Helsley. *LXC: Linux container tools*, IBM developerWorks, 03.02.2009, URL: https://www.ibm.com/developerworks/library/l-lxc-containers/l-lxc-containers-pdf.pdf

[3] *Docker user guide*, URL: https://docs.docker.com/userguide/

[4] V. P. Gergel'. "NAS Parallel Benchmarks", *Tekhnologii postroyeniya i ispol'zovaniya klasternykh sistem*, Uchebnyy kurs Natsional'nogo otkrytogo universiteta "INTUIT" (data obnovleniya 18.10.2009) (in Russian), URL: http://www.intuit.ru/studies/courses/542/398/lecture/9173?page=3#sect9

Sample citation of this publication:

Anton Baranov, Dmitrii Nikolaev. "The use of container virtualization in the organization of high-performance computing", *Program systems: theory and applications*, 2016, **7**:1(28), pp. 117–134. (*In Russian*).

URL: http://psta.psiras.ru/read/psta2016_1_117-134.pdf