СРАВНИТЕЛЬНЫЙ АНАЛИЗ СУЩЕСТВУЮЩИХ ТЕХНОЛОГИЙ КОНТЕЙНЕРИЗАЦИИ

Адрова Л.С., Полежаев П.Н.
ФГБОУ ВО «Оренбургский государственный университет», г. Оренбург

Задача планирования SaaS- и PaaS-сервисов является схожей с проблемой планирования вычислительных задач в высокопроизводительных (High Performance Computing, HPC) системах. Для этой цели в настоящее время используются различные системы управления, включая решения для кластерных и грид-систем. Наиболее известными системами управления кластерами являются – Torque, PBS Pro, LSF, Load Leveler, Cleo, грид-системами – SGE, Condor, Work Queue. В таких системах для организации потока управления задачами (job workflow – графа зависимостей подзадач по данным) используются специальные менеджеры задач, например, Makeflow, Condor DAG-Man, Galaxy, Kepler, Pegasus, Swift, Taverna.

Существенным недостатком существующих систем управления вычислениями в облачных и высокопроизводительных системах является неэффективное использование вычислительных ресурсов серверов. В первом случае сервисы размещаются внутри виртуальных машин, во втором подзадачи – поверх физических серверов или виртуальных машин (HPC поверх облака). При этом нередки случаи, когда вычислительные ресурсы остаются недогруженными [1]. Логичным решением для повышения загруженности вычислительных ресурсов является использование контейнеров.

Основным достоинством технологии контейнеризации в сравнении с использованием виртуализации является снижение непроизводительных накладных расходов. Для организации работы полноценных виртуальных машин необходима полная эмуляция гипервизором (например, KVM, Xen, Hyper-V) аппаратного обеспечения компьютера, каждая виртуальная машина включает в себя полную копию гостевой ОС. В случае же использования технологии контейнеров (например, Linux-VServer, OpenVZ, LXC, Docker) имеется единственное общее ядро ОС компьютера, поверх которого работают легковесные контейнеры с процессами запущенных программ, нет необходимости эмулировать аппаратное обеспечение – используются физические средства сервера.

Существующие экспериментальные исследования по использованию контейнеров и виртуальных машин для HPC [2] показывают, что издержки виртуализации являются существенными. Контейнеры имеют вычислительную производительность, скорость доступа к оперативной и дисковой памяти практически эквивалентные показателям физического сервера. Использование виртуальных машин снижает вычислительную производительность на 4-5%, приводит к замедлению на 30% операций при работе с оперативной памятью (трансляция адресов), на 30-50% при доступе к файлам на жестких дисках, на 40-60% снижается пропускная способность сети.

С другой стороны, анализ степени изолированности контейнеров и виртуальных машин, выполненный в работе [2], показывает, что виртуальные машины обеспечивают более честное распределение физических ресурсов, чем контейнеры, а также более высокий уровень безопасности и надежности (гипервизоры виртуальных машин имеют меньший размер кодовой базы и лучше протестированы).

Другим важнейшим достоинством контейнеров является удобство развертывания приложений и сервисов с учетом всех зависимых компонент, файлов и библиотек [3]. Каждый контейнер содержит все необходимое окружение для работы сервиса, использование сети и подключаемых разделяемых томов позволяет обеспечить коммуникации между различными сервисами.

Контейнеры в сравнении с виртуальными машинами запускаются значительно быстрее, т.к. нет необходимости загружать ОС, достаточно только запустить необходимые приложения [4, 5].

Linux-VServer базируется на собственной реализации механизмов изоляции процессов, которые реализуются путем модификации ядра Linux [6]. Пространство процессов (PID space) является глобальным с сокрытием всех процессов, которые являются внешними по отношению к контейнеру. Основное достоинство – возможность масштабирования на случай значительного количества контейнеров на одном узле, недостатки – сложность реализации механизмов сохранения и восстановления состояния, живой миграции из-за невозможности получения процессами тех же PID после перезапуска. Сеть в Linux-VServer не изолируется, в частности, таблицы маршрутизации, таблицы IP-адресов являются общими для всех контейнеров – это таблицы физического сервера. Изоляция сетевого трафика реализуется инкапсуляцией в пакеты контейнера специальной метки с последующей фильтрацией для контейнера всех сетевых пакетов, которые ее не имеют. Такой подход не позволяет контейнеру самостоятельно задавать свои сетевые настройки, а также создавать сокеты, привязанные к конкретным IP-адресам физического сервера. Для ограничения использования контейнером системных ресурсов применяются технологии rlimit и cgroups, позволяющие группам процессов контейнеров задать лимиты и приоритеты использования ресурсов ввода-вывода, оперативной памяти и процессора. Для планирования процессов контейнера используется алгоритм Token Bucket Filter.

Система OpenVZ для изоляции программ внутри контейнеров использует пространства имен ядра Linux (kernel namespaces) [7]. Каждому контейнеру прозрачным образом предоставляется окружение, которое им воспринимается как монопольное использование системы. Данное окружение включает в себя идентификаторы процессов (process IDs), средства межпроцессного взаимодействия (IPC), сеть и файловую систему. Оно позволяет осуществлять сохранение/восстановление контейнера, его живую миграцию. Также в OpenVZ каждый контейнер обладает своим собственным сетевым стеком (network

namespace), позволяющим настраивать его различные режимы подключения к сети, параметры адресации, маршрутизации, межсетевого экранирования.

Система LXC поддерживает: пространства имен ядра Linux, для изоляции сети – network namespaces, cgroups. Для планирования ввода-вывода, как и в OpenVZ, используется алгоритм Completely Fair Queuing.

В настоящее время наиболее популярным инструментом для управления контейнерами является Docker, основанный на системе LXC и расширяющий ее возможности. Основным достоинством Docker в сравнении с LXC является значительное упрощение создания, настройки и эксплуатации контейнеров. Кроме того, Docker завоевал огромную популярность и стал частью многих других открытых и коммерческих проектов, включая CoreOS, Orchard, Memcached as a Service, OpenShift и др. В связи с вышеупомянутыми достоинствами планируется использовать Docker для размещения контейнеров с сервисами и другими программными модулями в рамках данной НИР.

С целью централизованного управления контейнерами Docker на нескольких серверах/виртуальных машинах могут быть использованы инструменты Docker Swarm или Google Kubernetes.

Также существуют инструменты, которые объединяют в себе механизмы развертывания и управления виртуальными машинами и контейнерами – OpenHost и Proxmox VE [8], позволяющие создавать кластеры развертывания.

Для запуска вычислительных задач и облачных приложений существуют различные варианты использования контейнеров [3]. Первым вариантом является размещение каждой подзадачи/сервиса в отдельном контейнере, запускаемом в начале выполнения вычислений и останавливаемом по завершению. Основные недостатки – временные издержки на запуск и остановку каждого контейнера под отдельное задание или запрос к сервису, необходимость создавать и размещать множество уникальных образов для контейнеров. Другим вариантом является размещение всей задачи или облачного сервиса внутри одного контейнера. Недостатки – отсутствие возможности масштабирования, размещения на различных узлах, разделение только общего программного окружения. Еще одним вариантом является размещение каждой подзадачи/сервиса в отдельном контейнере с максимальным его повторным использованием для обработки различных входных данных или запросов, такой подход поддерживает изоляцию, внешнее масштабирование (при увеличении загрузки запуск нескольких контейнеров на основе общего образа). Улучшением последнего подхода, позволяющего еще больше сократить количество запусков и остановок контейнеров, является размещение внутри одного контейнера нескольких подзадач/сервисов с общим программным окружением и уровнем доверия. К числу достоинств данного подхода можно отнести сокращение количества создаваемых образов и возможность реализации внутреннего масштабирования (изменения количества запущенных экземпляров подзадач/сервисов внутри одного контейнера).

Результаты сравнительного анализа существующих систем контейнеризации сведены в таблицу 1.

Анализ данной таблицы показывает, что наиболее адекватными системами контейнеризации являются LXC и Docker. LXC сложно настраивать, Docker же базируется на LXC и максимально упрощает создание и настройку контейнеров, поэтому он и был выбран для использования совместно с инструментами виртуализации для размещения контейнеров SaaS и PaaS.

Таблица 1 – Сравнение существующих систем контейнеризации

Критерии \ Система управления контейнерами	Linux-VServer	OpenVZ	LXC	Docker
Изоляция процессов, IPC	+ (собственная модификация ядра Linux)	+ (kernel namespaces)	+ (kernel namespaces)	+ (kernel namespaces)
Изоляция файловой системы	+	+	+	+
Изоляция сети	+/- (использования сети физического сервера)	+	+ (network namespaces)	+
Планирование процессора	+ (Token Bucket Filter)	+ (Fair Scheduler)	+	+
Ограничение использования системных ресурсов	+ (rlimit, cgroups)	+ (User Beancounters, Disk Quota и VCPU Affinity)	+ (cgroups)	+ (cgroups)
Возможность сохранения /восстановления	-	+	+	+
Возможность живой миграции	-	+	+	+

Работа выполнена при поддержке РФФИ (проект №15-07-06071), Президента Российской Федерации, стипендия для молодых ученых и аспирантов (СП-2179.2015.5).

Список литературы

1 Peterson L. et al. Experiences building planetlab //Proceedings of the 7th symposium on Operating systems design and implementation. – USENIX Association, 2006. – С. 351-366.

2 Xavier M. G. et al. Performance evaluation of container-based virtualization for high performance computing environments //Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on. – IEEE, 2013. – С. 233-240.

3 Zheng C., Thain D. Integrating Containers into Workflows: A Case Study Using Makeflow, Work Queue, and Docker. – 2015.

4 Huber N. et al. Evaluating and Modeling Virtualization Performance Overhead for Cloud Environments //CLOSER. – 2011. – С. 563-573.

5 Rosenblum M., Garfinkel T. Virtual machine monitors: Current technology and future trends //Computer. – 2005. – Т. 38. – №. 5. – С. 39-47.

6 Soltesz S. et al. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors //ACM SIGOPS Operating Systems Review. – ACM, 2007. – Т. 41. – №. 3. – С. 275-287.

7 Rosen R. Resource management: Linux kernel Namespaces and cgroups //Haifux, May. – 2013. – URL: http://www.cs.ucsb.edu/~rich/class/cs290-cloud/papers/lxc-namespace.pdf

8 *Kovári A., Dukan P. KVM & OpenVZ virtualization based IaaS open source cloud virtualization platforms: OpenNode, Proxmox VE //Intelligent Systems and Informatics (SISY), 2012 IEEE 10th Jubilee International Symposium on. – IEEE, 2012. – С. 335-339.*