Задачи оптимизации размещения контейнеров MPI-приложений на вычислительных кластерах[1]

[1] *Д.А. Грушин <grushin@ispras.ru>*
[1,2] *Н.Н. Кузюрин <nnkuz@ispras.ru>*
[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Московский физико-технический институт, 141700, Московская область, г. Долгопрудный, Институтский пер., 9*

Аннотация. MPI — это хорошо зарекомендовавшая себя технология, которая широко используется в высокопроизводительной вычислительной среде. Однако настройка кластера MPI может быть сложной задачей. Контейнеры — это новый подход к виртуализации и простой упаковке приложений, который становится популярным инструментом для высокопроизводительных задач (HPC). Именно этот подход рассматривается в данной статье. Упаковка MPI-приложения в виде контейнера решает проблему конфликтных зависимостей, упрощает конфигурацию и управление запущенными приложениями. Для управления ресурсами кластера может использоваться обычная система очередей (например, SLURM) или системы управления контейнерами (Docker Swarm, Kubernetes, Mesos и др.). Контейнеры также дают больше возможностей для гибкого управления запущенными приложениями (остановка, повторный запуск, пауза, в некоторых случаях миграция между узлами), что позволяет получить преимущество при оптимизации размещения задач по узлам кластера по сравнению с классической схемой работы планировщика. В статье рассматриваются различные способы оптимизации размещения контейнеров при работе с HPC-приложениями. Предлагается вариант запуска MPI приложений в системе Fanlight, упрощающий работу пользователей. Рассматривается связанная с данным способом запуска задача оптимизации.

Ключевые слова: контейнеры, docker, оптимизация

DOI: 10.15514/ISPRAS-2017-29(6)-14

Для цитирования: Грушин Д.А., Кузюрин Н.Н. Задачи оптимизации размещения контейнеров MPI-приложений на вычислительных кластерах. Труды ИСП РАН, том 29, вып. 6, 2017 г., стр. 229-244. DOI: 10.15514/ISPRAS-2017-29(6)-14

[1] Работа выполнена в рамках гранта РФФИ 17-07-01006 А

1. Введение

Высокопроизводительные вычисления (HPC) – это важная область в информатике, которая предполагает решение больших задач, требующих высокой вычислительной мощности. Эти задачи слишком велики для одного компьютера, поэтому для HPC используют группу машин, которые работают вместе. Существуют различные модели параллельного программирования, разработанные для эффективной координации кластера компьютеров. Одной из популярных моделей является передача сообщений для оперативной связи и совместных вычислений.

В настоящее время MPI является «де-факто» стандартом передачи сообщений в HPC [1]. MPI имеет устойчивость к аппаратным или сетевым ошибкам, обеспечивает гибкий механизм для разработчиков распределенной программы или инструментов поверх нее.

MPI – это библиотека функций, которые можно вызывать из разных языков программирования для обеспечения различных функциональных возможностей для параллельных программ. Каждый процесс MPI хранит свои данные в локальной памяти и обменивается данными с другими процессами MPI, передавая сообщения через сеть. Существует много реализаций, соответствующих стандарту MPI. Стандарт определяет различные аспекты интерфейса передачи сообщений, включая передачу сообщений «точка-точка», коллективные коммуникации и привязки различных языков программирования. Существует множество версий MPI с открытым исходным кодом, таких как Open-MPI и MVAPICH. Существуют также коммерческие реализации MPI, такие как Intel MPI, ScaMPI (бывший HP-MPI) и Voltaire MPI.

Традиционно настройка кластера MPI является сложной задачей, которая требует от системных администраторов значительного времени, т.к. системы HPC обычно обслуживают большое количество пользователей, которым необходимо запускать разные приложения с противоречивыми требованиями и зависимостями.

Контейнеризация в Linux – это технология виртуализации на уровне операционной системы, которая предлагает "легкую" виртуализацию. Приложение, которое работает как контейнер, имеет свою собственную корневую файловую систему, но разделяет ядро с операционной системой хоста.

За последние несколько лет контейнерные технологии были быстро внедрены в IT отрасль, и эта технология почти стала синонимом Docker. Разработчики проекта взяли за основу довольно зрелую концепцию контейнеров (Solaris Zones, IBM LPAR, LXC и др.) и быстро разработали поверх нее удобный для

пользователя продукт с рабочим процессом, который идеально подходит под философию DevOps[2] и современных микросервисных архитектур.

В последние годы проект Docker с открытым исходным кодом завоевывает популярность в применении контейнеров для высокопроизводительных вычислений [2]. Контейнеры Docker упаковывают программное обеспечение в полную файловую систему, в которой содержится все необходимое для запуска: код, среда выполнения, системные инструменты и системные библиотеки.

Разработчики программного обеспечения могут создавать сертифицированные контейнеры для своих приложений, уменьшая накладные расходы для клиента. По умолчанию (при первом запуске) контейнер не имеет состояния. Это гарантирует, что каждый запуск одного и того же образа контейнера будет одинаковым. Измененные файлы или настройки будут очищены после завершения приложения. Это гарантирует, что программа будет работать как ожидалось, независимо от ее окружения. Вместо того, чтобы делить вычислительные ресурсы на разделы для удовлетворения различных требований, кластер может быть установлен с минимальной конфигурацией, а все требования к окружению будут обеспечены контейнерами приложений. Это значительно уменьшает накладные расходы на управление конфигурацией.

Поскольку контейнеры, в отличие от виртуальных машин, не требуют наличия полной ОС, потеря производительности оказывается незначительной. Тесты показывают среднюю потерю для приложений MPI в 1,5-2% [4]. Данный факт, а также неоспоримые плюсы упаковки MPI-приложения в контейнер объясняют растущую популярность применения контейнеров для высокопроизводительных вычислений.

Однако одной из важных проблем вычислительных кластеров является оптимизация загрузки. Общепринятым решением является отправка задачи в очередь планировщика, ожидание освобождения необходимых ресурсов и запуск задачи эксклюзивно на выделенных узлах.

В статье рассматриваются различные способы оптимизации размещения контейнеров при работе с HPC-приложениями. Также рассматривается вариант запуска MPI-приложений в системе Fanlight.

[2] DevOps (акроним от англ. development и operations) — набор практик, нацеленных на активное взаимодействие специалистов по разработке со специалистами по информационно-технологическому обслуживанию и взаимную интеграцию их рабочих процессов друг в друга. Базируется на идее о тесной взаимозависимости разработки и эксплуатации программного обеспечения и нацелен на то, чтобы помогать организациям быстрее создавать и обновлять программные продукты и услуги.

2. Распределение контейнеров по узлам кластера

Как известно, вычислительный кластер состоит из управляющего и вычислительных узлов, связанных высокопроизводительной сетью. Каждый узел содержит несколько процессоров, каждый процессор –. несколько ядер.

Для параллельной программы необходимо выделить заданное количество процессорных ядер в эксклюзивное пользование. В противном случае (две задачи используют одно ядро –. overbooking) производительность сильно снижается, что приводит к непредсказуемому увеличению времени работы задач [7].

Для управления доступом к вычислительным узлам используется планировщик. Планировщик устанавливается на все узлы, на управляющем узле находится очередь. Доступ на узлы в обход планировщика запрещается. При запуске задачи указывается время работы, по окончанию которого задача будет принудительно завершена.

Все поступающие на кластер задачи помещаются в очередь. Если необходимое количество ресурсов для первой в очереди задачи свободно, то ресурсы помечаются как занятые, задача забирается из очереди и запускается. Способ выбора необходимых задаче узлов из свободных определяется алгоритмом оптимизации. При этом оптимизация может проводиться по одному или нескольким критериям. Для вычислительного кластера обычно используют следующие критерии:

- минимизация времени ожидания в очереди,
- максимизация пропускной способности кластера (количество завершенных задач в единицу времени),
- минимизация энергопотребления и времени ожидания в очереди, и др.
- Наиболее распространенным является алгоритм Backfill, когда для заполнения "пустых мест" в расписании используются задачи не с начала очереди [8]. При этом время работы задачи неизвестно, либо задается пользователем. В последнем случае оказывается, что время работы пользователи задают с большой положительной погрешностью, так как система принудительно завершает задачи по истечении выделенного времени [9]. В [10] было предложено не использовать время, задаваемое пользователем, а предсказывать его на основе истории запуска (журнала) задач.
- Менеджер ресурсов (планировщик) вычислительного кластера использует shell-сценарий, который является описанием задания, и отвечает за настройку среды выполнения и передачу необходимой информации процессу `mpirun`. Для удаленного выполнения используется SSH. При запуске, процессу передается список адресов зарезервированных для данной задачи узлов кластера, который используется библиотекой MPI для запуска остальных процессов

программы. Данная схема запуска является стандартной для MPI-приложений и плотно интегрирована со всеми известными планировщиками.

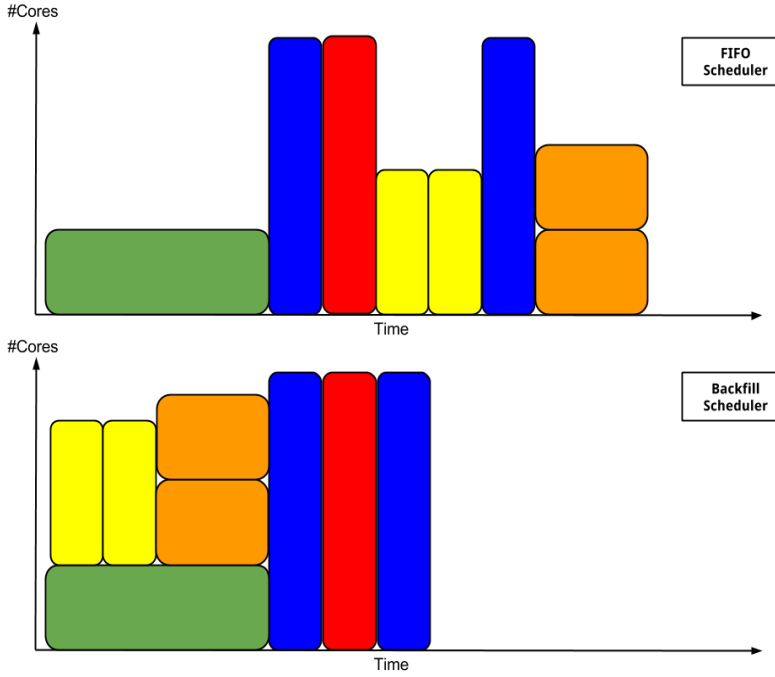

Рис. 1. Backfill
Fig. 1. Backfill

3. HPC-Контейнеры

Жизненным циклом контейнера управляет специальное ПО –. система управления контейнерами, в задачи которой входит:

- построение образа контейнера,
- хранение и обработка образов контейнеров,
- создание контейнера из образа,
- запуск, контроль выполнения, остановка, удаление контейнера и др.

Наиболее известными на сегодняшний день системами управления контейнерами являются: Docker, Kubernetes, Mesos, LXC, и др. Для формирования единого подхода в управлении контейнерами была создана организация OCI (Open Container Initiative) [11].

Управляющее ПО должно быть установлено на каждый из узлов кластера. Для запуска контейнера необходимо, чтобы образ контейнера находился на каждом из задействованных узлов кластера. Это означает, что образ необходимо скопировать на узлы заранее, или непосредственно перед запуском задачи. Средний размер образа параллельной программы может достигать нескольких гигабайт, что в определенных условиях вносит существенную задержку при запуске. Предпочтительной стратегией оптимизации в данном случае является регистрация и отслеживание образов и управление кешированными данными на узлах кластера [12].

При использовании HPC-контейнеров возможны несколько вариантов.

- Наиболее распространенной является схема запуска через существующий планировщик. В данном случае сохраняется совместимость –. кластер одновременно можно использовать для обычных приложений [13]. Разработчики некоторых планировщиков уже начали внедрять поддержку контейнеров [16].
- Управление распределением задач средствами системы управления контейнерами (Kubernetes, Docker, Mesos).
- Смешанный режим. Кластер используется как для высокопроизводительных вычислений, так и для микросервисов. В данном режиме кластер находится под управлением двух планировщиков: системы управления контейнерами и HPC-планировщика. Администратор кластера решает, какое количество ресурсов будет предоставлено под высокопроизводительные вычисления [18].

4. Оптимизация размещения HPC-контейнеров

В финансовом или инженерном моделировании HPC-задание может состоять из десятков тысяч коротких задач, требующих планирования с малой задержкой и высокой пропускной способностью для завершения моделирования в течение приемлемого периода времени. Задача вычислительной гидродинамики (CFD) может выполняться параллельно на многих сотнях или даже тысячах узлов, используя библиотеку передачи сообщений для синхронизации состояния. Для размещения и запуска таких заданий (а также проверки, приостановки, возобновления) требуются специализированные функции планирования и управления.

Другие HPC-задачи могут требовать специализированные ресурсы, такие как графические процессоры или доступ к ограниченным лицензиям на программное обеспечение. Организации могут проводить политику в отношении того, какие типы ресурсов могут быть использованы для обеспечения адекватного финансирования проектов и соблюдения сроков.

HPC-планировщики развивались и внедряли поддержку таких видов рабочей нагрузки. Поэтому, если говорить об универсальном решении для запуска

контейнеризированных программ на кластере, то запуск через HPC-планировщик остается единственным вариантом. Однако, универсальное решение требуется не всегда. Организовать и эффективно использовать кластер для определенного вида HPC-задач с использованием контейнеров можно и с помощью систем управления контейнерами.

В контейнерных распределенных вычислительных системах решается другой вид задач. Распределенная программная платформа (framework [3]) предоставляет абстракцию разработчикам, что позволяет им использовать большой объем ресурсов, рассматривая их как единый пул. Платформа предоставляет средства для распределения и обработки данных и задач на узлах, а также решает проблемы отказоустойчивости, такие как перезапуск задачи, сохранение состояния и др. Известными системами являются Spark[19], Storm[20], Tez[21] и др.

Существует три основные категории планировщиков для распределенных программных платформ:

- монолитные,
- двухуровневые,
- с разделяемым состоянием.

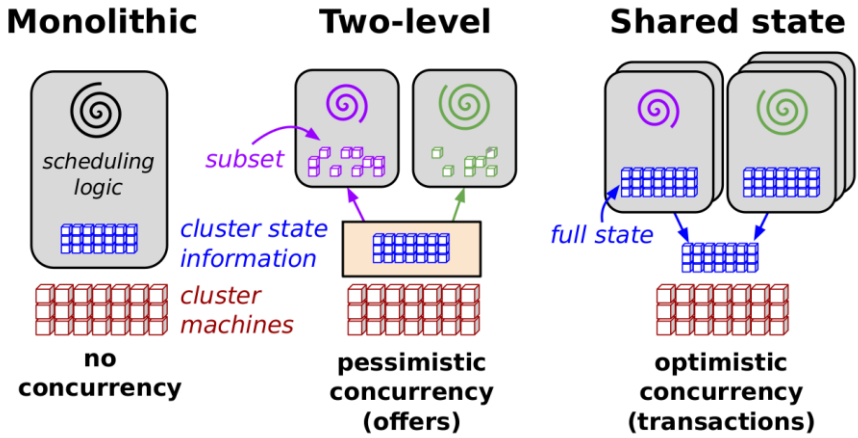

Рис. 2. Виды планировщиков
Fig. 2. Schedulers

Первый тип имеет глобальный пул ресурсов и должен реализовывать множество возможных политик распределения задач для каждой структуры

[3] Остов, каркас, структура — программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

кластера (предполагается, что кластер используется одновременно различными вычислительными платформами). По мере увеличения количества специализированных структур и приложений планировщик может стать узким местом.

Двухуровневые планировщики, такие как Mesos, выделяют пул ресурсов в структуры, которые затем реализуют свои собственные специализированные политики распределения. Каждая структура имеет частичное представление глобального пула ресурсов.

Планировщики с разделяемым состоянием предоставляют весь пул вычислительных ресурсов для набора специализированных планировщиков. За счет этого достигается высокий уровень консолидации ресурсов и появляется большая гибкость с точки зрения различий в политике распределения, но при этом необходимо тщательно управлять изменениями в глобальном пуле ресурсов для смягчения противоречивых решений о распределении.

С ростом популярности больших данных (big data) количество машин во многих кластерах увеличилось с сотен до десяти тысяч и более, а поток обрабатываемых задач характеризуется высокой интенсивностью и малым размером отдельной задачи. При этом планировщик должен обеспечивать высокую загрузку кластера, доступность, локальность данных. За последние годы было разработано множество планировщиков, некоторые используются в крупных компаниях: Microsoft Apollo [22], Google Borg [23], Facebook Tupperware [24], Twitter Aurora [25], Alibaba Fuxi [26], Omega [27], Sparrow [28], Hawk [29], Tarcil [30].

5. Разработка в ИСП РАН

В ИСП РАН разработана и развивается программная система Fanlight, позволяющая использовать разнородные вычислительные ресурсы в виде модели "Desktop as a Service" [31]. Работая с системой через Web-браузер пользователь получает доступ к графическому рабочему столу и возможность запускать на нем приложения, используя вычислительные ресурсы облака. Рабочий стол и приложения размещаются в отдельных контейнерах.

В ИСП РАН Fanlight используется пользователями в рамках программы Unihub для задач вычислительной гидродинамики (CFD).

Система поддерживает несколько видов приложений, запускаемых в отдельных контейнерах.

- **Графические Linux-приложения**. Приложение запускается внутри контейнера, окно приложения отображается в контейнере с запущенным Xvnc сервером, клиент HTML5 (noVnc) подсоединяется к Xvnc серверу через прокси.
- **Консольные**. Приложение запускается в контейнере, доступ к нему можно получить через HTML5 терминал.

- **Web-приложения**. Приложение доступно через браузер. Соединение происходит через прокси.

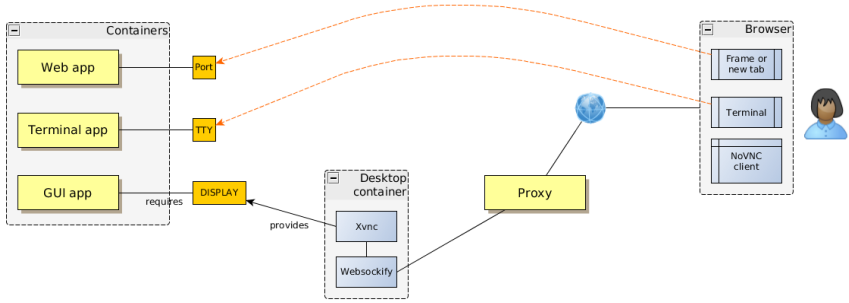

Рис. 3. Приложения Fanlight
Fig. 3. Fanlight applications

В основном, система используется для работы с графическими приложениями, такими как Salome, Paraview, Blender, Tnavigator и др. В системе реализована возможность работы с графическими ускорителями. OpenGL графика работает через VirtualGL, который перенаправляет команды 3D-рендеринга из OpenGL приложения в аппаратный 3D-ускоритель и отдает сжатые изображения VNC серверу.

Система Fanlight состоит из нескольких микросервисов.

- Controller
  - Взаимодействует с Docker API.
  - Отвечает за обработку REST API.
  - Управляет авторизацией и хранением данных.
- Proxy. Обеспечивает соединение между noVNC клиентом в браузере и VNC сервером в контейнере.
- Frontend. Реализует WEB-интерфейс.
- Admin Dashboard. WEB-интерфейс для администратора.

Для развертывания системы на вычислительных ресурсах необходимо установить на каждом узле Docker, далее процесс развертывания происходит в автоматическом режиме с помощью системы Ansible и/или Docker Compose. Все приложения добавляются в систему в виде Docker образов и не требуют дополнительной настройки системного окружения.

Домашние директории пользователей монтируются на все узлы средствами сетевой файловой системы. Все приложения и рабочие столы пользователя имеют доступ к его домашней директории.

Система Fanlight может работать как на одном, так и на нескольких серверах. Если в качестве ресурсов используется вычислительный кластер с сетью Infiniband, то система может запускать контейнеризированные MPI-приложения.

6. Запуск контейнера с MPI-задачей в Fanlight и стратегии размещения контейнеров

При работе с MPI-приложением на вычислительном кластере пользователь использует систему очередей планировщика. Подготовленный выполняемый скрипт специальной командой ставится в очередь (пакетное задание). Когда запрашиваемые задачей ресурсы освобождаются, планировщик запускает скрипт, передавая ему необходимое системное окружение. При большом числе пользователей задача может ожидать своей очереди несколько часов, и если в скрипте имеется ошибка, то все это время будет потрачено впустую. Такой сценарий случается достаточно часто. Для борьбы с этой проблемой обычно администраторы настраивают тестовую очередь, в которой задача принудительно завершается сразу после запуска. Это позволяет убедиться, что скрипт работает без ошибок, и программа начинает расчет.

Fanlight предлагает другой способ запуска. Поскольку контейнер может быть временно приостановлен, то ресурсы для задачи выделяются сразу, задача запускается и затем приостанавливается до освобождения узлов. Это позволяет решить проблему с ошибкой запуска и упростить работу для пользователей, особенно начинающих.

При распределении задач таким способом могут применятся различные стратегии. Наиболее простыми, имеющимися по умолчанию в системах управления контейнерами, например, Docker Swarm, являются:

- **random –**. выбирается случайный узел;
- **spread –**. выбирается наименее загруженный узел;
- **binpack –**. размещается как можно больше контейнеров на одном узле.

При большом количестве задач данные стратегии работают недостаточно эффективно.

HPC-планировщик со стратегией Backfill запускает задачу в момент освобождения необходимых ресурсов [8]. Это означает, что планировщик может выбирать, когда и где запустить задачу. В предлагаемом нами варианте запуска задач в системе Fanlight задача запускается сразу, а затем ждет своей очереди в "спящем" состоянии. В данном случае мы можем выбрать момент времени для пробуждения задачи, однако не можем переместить задачу на другие узлы. Это ограничение может в определенных ситуациях приводить к образованию пустых мест –. окон в расписании. Однако, проведенные нами наблюдения показали, что в большинстве случаев этого не происходит.

Тем не менее, данная особенность требует более детального исследования, выходящего за рамки данной статьи. Стратегия распределения должна учитывать вероятность возникновения таких окон и минимизировать ее, определенным образом размещая задачи.

7. Заключение

В статье были рассмотрены различные способы оптимизации размещения контейнеров при работе с HPC-приложениями. Также был показан вариант запуска MPI-приложений в системе Fanlight, разработанной в ИСП РАН [31].

В [32] мы предложили собирать данные о поведении пользователей и характере работы различных приложений с целью прогнозирования создаваемой контейнерами нагрузки. В данной работе мы предлагаем использовать полученную информацию о поведении пользователей и приложений для более точного предсказания времени работы задачи.

Предварительный анализ показывает, что чем точнее определяется время работы каждой задачи, тем меньше вероятность сдвига (раньше или позже запланированного) запуска следующей задачи и возникновения незаполненных окон в расписании. Таким образом, точность прогнозирования повышает эффективность использования ресурсов.

Дальнейшие исследования будут направлены на более детальный анализ факторов, влияющих на точность прогнозирования времени работы задач, а также на изучение факторов, приводящих к возникновению окон в расписании при распределении задач в системе Fanlight.

С использованием имеющихся вычислительных ресурсов ИСП РАН построена система из нескольких сотен контейнеров под управлением систем Docker Swarm, Kubernetes, Mesos. Исследовано поведение каждой из перечисленных систем в следующих сценариях работы: обновление приложения, развертывание приложения, добавление и удаление серверов. Разработан автоматический тест для оценки скорости работы систем в перечисленных сценариях.

Список литературы

[1]. Forum M.P. MPI: A message-passing interface standard. Knoxville, TN, USA: University of Tennessee, 1994.

[2]. Nguyen N., Bein D. Distributed mpi cluster with docker swarm mode. 2017 ieee 7th annual computing and communication workshop and conference (ccwc). 2017. Pp. 1–7.

[3]. Azab A. Enabling docker containers for high-performance and many-task computing. 2017 ieee international conference on cloud engineering (ic2e). 2017. Pp. 279–285.

[4]. Ermakov A., Vasyukov A. Testing docker performance for HPC applications. CoRR. 2017. Vol. abs/1704.05592.

[5]. Felter W. et al. An updated performance comparison of virtual machines and linux containers. 2014.

[6]. Di Tommaso P. et al. The impact of docker containers on the performance of genomic pipelines. PeerJ. 2015. Vol. 3. P. e1273.
[7]. Herbein S. et al. Resource management for running hpc applications in container clouds. High performance computing: 31st international conference, isc high performance 2016, frankfurt, germany, june 19-23, 2016, proceedings / ed. Kunkel J.M., Balaji P., Dongarra J. Cham: Springer International Publishing, 2016. Pp. 261–278.
[8]. Baraglia R. et al. Backfilling strategies for scheduling streams of jobs on computational farms. Making Grids Work. Springer, 2008. Pp. 103–115.
[9]. Mu'alem A.W., Feitelson D.G. Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. IEEE Transactions on Parallel and Distributed Systems. 2001. Vol. 12, № 6. Pp. 529–543.
[10]. Nissimov A., Feitelson D.G. Probabilistic backfilling. Job scheduling strategies for parallel processing: 13th international workshop, jsspp 2007, seattle, wa, usa, june 17, 2007. revised papers. ed. Frachtenberg E., Schwiegelshohn U. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. Pp. 102–115.
[11]. https://www.opencontainers.org.
[12]. Harter T. et al. Slacker: Fast distribution with lazy docker containers. 14th USENIX conference on file and storage technologies, FAST 2016, santa clara, ca, usa, february 22-25, 2016. 2016. Pp. 181–195.
[13]. Higgins J., Holmes V., Venters C. Orchestrating docker containers in the hpc environment. High performance computing: 30th international conference, isc high performance 2015, Frankfurt, Germany, July 12-16, 2015, proceedings / ed. Kunkel J.M., Ludwig T. Cham: Springer International Publishing, 2015. Pp. 506–513.
[14]. Benedicic L. et al. Portable, high-performance containers for hpc. arXiv preprint arXiv:1704.03383. 2017.
[15]. Kurtzer G.M., Sochat V., Bauer M.W. Singularity: Scientific containers for mobility of compute. PLOS ONE. Public Library of Science, 2017. Vol. 12, № 5. Pp. 1–20.
[16]. http://www.univa.com/resources/files/gridengine_container_edition.pdf.
[17]. https://developer.ibm.com/storage/products/ibm-spectrum-lsf/.
[18]. https://navops.io/command.html.
[19]. https://spark.apache.org.
[20]. https://storm.apache.org.
[21]. https://tez.apache.org.
[22]. Boutin E. et al. Apollo: Scalable and coordinated scheduling for cloud-scale computing. Proceedings of the 11th usenix conference on operating systems design and implementation. Berkeley, CA, USA: USENIX Association, 2014. Pp. 285–300.
[23]. Verma A. et al. Large-scale cluster management at google with borg. Proceedings of the tenth european conference on computer systems. New York, NY, USA: ACM, 2015. Pp. 18:1–18:17.
[24]. http://www.slideshare.net/dotCloud/tupperware-containerized-deployment-at-facebook.
[25]. http://aurora.incubator.apache.org/.
[26]. Zhang Z. et al. Fuxi: A fault-tolerant resource management and job scheduling system at internet scale. Proc. VLDB Endow. VLDB Endowment, 2014. Vol. 7, № 13. Pp. 1393–1404.
[27]. Schwarzkopf M. et al. Omega: Flexible, scalable schedulers for large compute clusters. Proceedings of the 8th acm european conference on computer systems. New York, NY, USA: ACM, 2013. Pp. 351–364.

[28]. Ousterhout K. et al. Sparrow: Distributed, low latency scheduling. Proceedings of the twenty-fourth acm symposium on operating systems principles. New York, NY, USA: ACM, 2013. Pp. 69–84.
[29]. Delgado P. et al. Hawk: Hybrid datacenter scheduling. Proceedings of the 2015 usenix conference on usenix annual technical conference. Berkeley, CA, USA: USENIX Association, 2015. Pp. 499–510.
[30]. Delimitrou C., Sanchez D., Kozyrakis C. Tarcil: Reconciling scheduling speed and quality in large shared clusters. Proceedings of the sixth acm symposium on cloud computing. New York, NY, USA: ACM, 2015. Pp. 97–110.
[31]. http://fanlight.ispras.ru.
[32]. Д.А. Грушин, Н.Н. Кузюрин. Балансировка нагрузки в системе Unihub на основе предсказания поведения пользователей. Труды ИСП РАН, том 27, вып. 5, 2015 г., стр. 23–34. DOI: 10.15514/ISPRAS-2015-27(5)-2

Optimization problems running MPI-based HPC applications

[1]*D.A. Grushin <grushin@ispras.ru>*
[1,2]*N.N.Kuzjurin <nnkuz@ispras.ru>*
[1]*Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia*
[2] *Moscow Institute of Physics and Technology, Dolgoprudnyj, Institutskij alley, Moscow region, 141700, Russia*

Abstract. MPI is a well-proven technology that is widely used in a high-performance computing environment. However, configuring an MPI cluster can be a difficult task. Containers are a new approach to virtualization and simple application packaging, which is becoming a popular tool for high-performance tasks (HPC). This approach is considered in this article. Packaging an MPI application as a container solves the problem of conflicting dependencies, simplifies the configuration and management of running applications. A typical queue system (for example, SLURM) or a container management system (Docker Swarm, Kubernetes, Mesos, etc.) can be used to manage cluster resources. Containers also provide more options for flexible management of running applications (stop, restart, pause, in some cases, migration between nodes), which allows you to gain an advantage optimizing the allocation of tasks to cluster nodes in comparison with the classic scheduler. The article discusses various ways to optimize the placement of containers when working with HPC-applications. A variant of launching MPI applications in Fanlight system is proposed, which simplifies the work of users. The optimization problem associated with this method is considered also.

Keywords: docker, containers, scheduling

DOI: 10.15514/ISPRAS-2017-29(6)-14

For citation: Grushin D.A., Kuzjurin N.N. Optimization problems running MPI-based HPC applications. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 6, 2017. pp. 229-244 (in Russian). DOI: 10.15514/ISPRAS-2017-29(6)-14

References

[1]. Forum M.P. MPI: A message-passing interface standard. Knoxville, TN, USA: University of Tennessee, 1994.
[2]. Nguyen N., Bein D. Distributed mpi cluster with docker swarm mode. 2017 ieee 7th annual computing and communication workshop and conference (ccwc). 2017. Pp. 1–7.
[3]. Azab A. Enabling docker containers for high-performance and many-task computing. 2017 ieee international conference on cloud engineering (ic2e). 2017. Pp. 279–285.
[4]. Ermakov A., Vasyukov A. Testing docker performance for HPC applications. CoRR. 2017. Vol. abs/1704.05592.
[5]. Felter W. et al. An updated performance comparison of virtual machines and linux containers. 2014.
[6]. Di Tommaso P. et al. The impact of docker containers on the performance of genomic pipelines. PeerJ. 2015. Vol. 3. P. e1273.
[7]. Herbein S. et al. Resource management for running hpc applications in container clouds. High performance computing: 31st international conference, isc high performance 2016, frankfurt, germany, june 19-23, 2016, proceedings / ed. Kunkel J.M., Balaji P., Dongarra J. Cham: Springer International Publishing, 2016. Pp. 261–278.
[8]. Baraglia R. et al. Backfilling strategies for scheduling streams of jobs on computational farms. Making Grids Work. Springer, 2008. Pp. 103–115.
[9]. Mu'alem A.W., Feitelson D.G. Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. IEEE Transactions on Parallel and Distributed Systems. 2001. Vol. 12, № 6. Pp. 529–543.
[10]. Nissimov A., Feitelson D.G. Probabilistic backfilling. Job scheduling strategies for parallel processing: 13th international workshop, jsspp 2007, seattle, wa, usa, june 17, 2007. revised papers. ed. Frachtenberg E., Schwiegelshohn U. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. Pp. 102–115.
[11]. https://www.opencontainers.org.
[12]. Harter T. et al. Slacker: Fast distribution with lazy docker containers. 14th USENIX conference on file and storage technologies, FAST 2016, santa clara, ca, usa, february 22-25, 2016. 2016. Pp. 181–195.
[13]. Higgins J., Holmes V., Venters C. Orchestrating docker containers in the hpc environment. High performance computing: 30th international conference, isc high performance 2015, Frankfurt, Germany, July 12-16, 2015, proceedings / ed. Kunkel J.M., Ludwig T. Cham: Springer International Publishing, 2015. Pp. 506–513.
[14]. Benedicic L. et al. Portable, high-performance containers for hpc. arXiv preprint arXiv:1704.03383. 2017.
[15]. Kurtzer G.M., Sochat V., Bauer M.W. Singularity: Scientific containers for mobility of compute. PLOS ONE. Public Library of Science, 2017. Vol. 12, № 5. Pp. 1–20.
[16]. http://www.univa.com/resources/files/gridengine_container_edition.pdf.
[17]. https://developer.ibm.com/storage/products/ibm-spectrum-lsf/.
[18]. https://navops.io/command.html.
[19]. https://spark.apache.org.
[20]. https://storm.apache.org.

[21]. https://tez.apache.org.
[22]. Boutin E. et al. Apollo: Scalable and coordinated scheduling for cloud-scale computing. Proceedings of the 11th usenix conference on operating systems design and implementation. Berkeley, CA, USA: USENIX Association, 2014. Pp. 285–300.
[23]. Verma A. et al. Large-scale cluster management at google with borg. Proceedings of the tenth european conference on computer systems. New York, NY, USA: ACM, 2015. Pp. 18:1–18:17.
[24]. http://www.slideshare.net/dotCloud/tupperware-containerized-deployment-at-facebook.
[25]. http://aurora.incubator.apache.org/.
[26]. Zhang Z. et al. Fuxi: A fault-tolerant resource management and job scheduling system at internet scale. Proc. VLDB Endow. VLDB Endowment, 2014. Vol. 7, № 13. Pp. 1393–1404.
[27]. Schwarzkopf M. et al. Omega: Flexible, scalable schedulers for large compute clusters. Proceedings of the 8th acm european conference on computer systems. New York, NY, USA: ACM, 2013. Pp. 351–364.
[28]. Ousterhout K. et al. Sparrow: Distributed, low latency scheduling. Proceedings of the twenty-fourth acm symposium on operating systems principles. New York, NY, USA: ACM, 2013. Pp. 69–84.
[29]. Delgado P. et al. Hawk: Hybrid datacenter scheduling. Proceedings of the 2015 usenix conference on usenix annual technical conference. Berkeley, CA, USA: USENIX Association, 2015. Pp. 499–510.
[30]. Delimitrou C., Sanchez D., Kozyrakis C. Tarcil: Reconciling scheduling speed and quality in large shared clusters. Proceedings of the sixth acm symposium on cloud computing. New York, NY, USA: ACM, 2015. Pp. 97–110.
[31]. http://fanlight.ispras.ru.
[32]. Grushin D., Kuzyurin N. Load balancing in unihub saas system based on user behavior prediction. Trudy ISP RAN/Proc. ISP RAS, vol. 27, issue 5, 2015, pp. 23–34 (in Russian). DOI: 10.15514/ISPRAS-2015-27(5)-2