РАСПРЕДЕЛЁННЫЕ ВЫЧИСЛЕНИЯ НА ОСНОВЕ DOCKER SWARM И TENSORFLOW ДЛЯ КЛАСТЕРОВ

Очередько О.О., Полежаев П.Н.
Оренбургский государственный университет

В настоящее время стало весьма актуально использование нейронных сетей для решения большого количества разнообразных задач. Их область применения не ограничена, нейронные сети используют для прогнозирования, распознавания образов, для анализа данных и кластеризации, принятия решений и управления, аппроксимации и оптимизации и т.д. Поскольку нейронная сеть для получения результата выполняет значительное количество векторных и матричных операций, наиболее подходящим способом её реализации является использование параллельного программирования на графических процессорах. Также в последние годы достаточно часто используют контейнеры для развертывания приложений, которые можно легко развернуть на распределенных системах, позволяя системе быстро масштабироваться и оставаться работоспособной при отказе отдельных машин или приложения.

Для управления Docker контейнерами необходим инструмент, с помощью которого можно создавать и управлять кластерами Docker на узлах, как единой виртуальной системой. Кластеризация является важной особенностью для контейнерных технологий, так как она создает совместную группу систем, которые могут обеспечить избыточность. Чаще всего в качестве такого инструмента выступает Docker Swarm.

Docker Swarm – это группа машин, на которых запущен Docker, при этом они соединены в единый кластер [1]. Функции управления кластерами и оркестровки (распределение контейнеров, управление кластером, и возможность добавления дополнительных машин), встроенные в Docker Engine, построены с использованием swarmkit. Swarmkit — это отдельный проект, который реализует уровень оркестровки Docker и используется непосредственно в Docker.

Swarm состоит из нескольких машин, которые могут быть физическими или виртуальными. При этом они работают либо как менеджеры (для управления членством и делегирования полномочий), либо как рабочие (которые выполняют службы swarm), либо выполняют обе роли. При создании сервиса можно определить её оптимальное состояние (количество реплик, доступных для сети и ресурсов хранения, порты, предоставляемые сервисом для внешнего мира, и многое другое). Docker поддерживает необходимое состояние. Например, если рабочий узел становится недоступным, Docker распределяет задачи этого узла на других узлах. Задача — это запущенный контейнер, который является частью службы swarm и управляется менеджером.

Одним из ключевых преимуществ сервисов swarm над автономными контейнерами является то, что можно изменить конфигурацию службы, включая сети и тома, к которой она подключена, без необходимости вручную переза-

пуска службы. Docker обновит конфигурацию, остановит задачи обслуживания с устаревшей конфигурацией и создаст новые, соответствующие желаемой конфигурации.

Для развертывания приложений на нескольких компьютерах с использованием графических процессоров NVIDIA используют Nvidia-docker. Nvidia-docker по существу является оберткой обычного Docker, которая прозрачно создает контейнер с необходимыми компонентами для выполнения кода на графическом процессоре [2].

Ещё до не давнего времени Nvidia-Docker не поддерживал режим Swarm. Но в конце декабря 2017 года вышла новая версия Docker 17.12.0-ce, в которой добавлена поддержка режима изоляции службы Swarm [3]. Теперь пользователь может настроить демон Docker так, чтобы графические процессоры были видны в Docker Swarm. Для этого необходимо выполнить следующие шаги:

1. Создать переопределение для конфигурации dockerd, изменив время выполнения по умолчанию и добавив ресурсы графического процессора. Флаги ресурсов можете сгенерировать следующим образом:

nvidia-smi -a | grep UUID | awk '{print "--node-generic-resource gpu="substr($4,0,12)}' | paste -d' ' -s

sudo systemctl edit docker

[Service]
ExecStart=
ExecStart=/usr/bin/dockerd -H fd:// --default-runtime=nvidia < ресурсы, приведенные выше >

2. Раскомментировать swarm-ресурс в /etc/nvidia-container-runtime/config.toml

3. Перезапустить демон Docker, создать swarm и создать новый сервис, запрашивающий графические процессоры, например:

docker service create -t -generic-resource "gpu = 1" ubuntu bash

Однако в некоторых случаях может возникнуть ошибка, если в файле конфигурации указан аргумент «node-generic-resource». Данную ошибку разработчики обещают исправить в ближайшее время. По этой причине в настоящее время рациональнее всего реализовывать распределённые вычисления для нейронных сетей через кластер TensorFlow с распределением части вычислений по графическим ускорителям и процессорам вычислительных узлов.

Кластер TensorFlow представляет собой набор «задач» («tasks»), которые участвуют в распределенном выполнении графа TensorFlow [4]. Каждая задача связана с сервером TensorFlow, который содержит «master» для создания сеан-

сов и «worker» для выполнения операций на видео ускорителях. Кластер также можно разделить на одно или несколько «заданий» («job»), где каждое задание содержит одну или несколько задач.

Для создания нового кластера необходимо во всех задачах запустить сервер TensorFlow, а затем выполнить следующие действия:

а) создать tf.train.ClusterSpec, который описывает все задачи в кластере. Он создаётся одинаковым для всех задач;

б) создать tf.train.Server, передав конструктору tf.train.ClusterSpec и определив локальную задачу с именем задания и индексом задачи.

Объект tf.train.Server содержит набор локальных устройств, набор подключений к другим задачам в tf.train.ClusterSpec и tf.Session, которые могут использовать их для реализации распределенных вычислений. Каждый сервер является членом определенного именованного задания и имеет индекс задачи в этом задании. Сервер может взаимодействовать с любым другим сервером в кластере.

Чтобы поместить операции в конкретный процесс, необходимо использовать функцию tf.device для определения места выполнении операции: на процессоре или на графическом процессоре.

Обычно в системе есть несколько вычислительных устройств. В TensorFlow поддерживаются такие типы устройств как CPU и GPU. В случае если операция будет назначена на CPU, а в системе имеются устройства обоих типов, то GPU будет присвоен приоритет [5].

Чтобы определить каким устройствам назначены операции, необходимо создать сеанс с параметром конфигурации log_device_placement, установленным в True.

Если необходимо, чтобы определенная операция выполнялась на некотором определённом устройстве, а не на том, которое было автоматически выбрано, необходимо указать это устройство в tf.device (например, tf.device('/cpu:0')). Таким образом, все операции в этом контексте будут иметь одинаковое назначенное устройство.

По умолчанию TensorFlow отображает память всех графических процессоров (с учетом CUDA_VISIBLE_DEVICES). Это делается для ещё более эффективного использования относительно ценных ресурсов памяти GPU на ускорителях за счет сокращения фрагментации памяти.

В некоторых случаях желательно, чтобы процесс выделял только подмножество доступной памяти или только увеличивал использование памяти. TensorFlow предоставляет два параметра конфигурации сеанса для его управления:

а) первый — это параметр allow_growth, который выделят такое количество памяти GPU, сколько необходимо для времени выполнения размещения: он начинает выделять достаточно малое количество памяти, и по мере того, как сеансы запускаются, расширяется область памяти GPU, необходимая процессу TensorFlow;

б) второй метод – это параметр per_process_gpu_memory_fraction, который определяет долю общего объема памяти, которую должен выделять каждый видимый графический процессор. Например, можно указать, чтобы TensorFlow выделял 50% общей памяти для каждого GPU, если прописать config.gpu_options.per_process_gpu_memory_fraction = 0.5.

Если нужно запустить TensorFlow на нескольких графических процессорах, то вы можете построить свою модель в режиме multi-tower. Для этого сначала создаётся массив со всеми необходимыми устройствами, а затем в цикле для каждого устройства задаются операции в tf.device().

Для того чтобы отследить производительность и другие характеристики графических процессоров в режиме реального времени необходимо воспользоваться командой «watch nvidia-smi», запускаемой в консоли. Она дает информацию о температуре графических процессоров, текущей используемой памяти, напряжении GPU, о том, какие задачи используют различные графические процессоры и др.

Таким образом, из-за некоторых ограничений Nvidia-Docker в Docker Swarm, рациональнее реализовывать распределённые вычисления для нейронных сетей через кластер TensorFlow с распределением части вычислений по графическим ускорителям и процессорам вычислительных узлов. Однако в скором времени, после исправления всех ошибок при использовании графических процессоров в Docker Swarm станет возможным создавать и быстро разворачивать кластеры на Docker машинах.

Исследование выполнено при финансовой поддержке Правительства Оренбургской области и РФФИ (проекты №17-47-560046, №16-29-09639 и №18-07-01446), Президента Российской Федерации в рамках стипендии для молодых ученых и аспирантов (СП-2179.2015.5).

Список литературы

1. JSwarm mode overview [Электронный ресурс] / Docker Documentation // URL: https://docs.docker.com/engine/swarm/ (дата обращения: 9.11.2017).

2. Очередько О.О., Полежаев П.Н. Сравнительный анализ обучения нейронной сети с использованием виртуальных машин и контейнеров NVidia-Docker // Современная техника и технологии: проблемы, состояние и перспективы: Материалы VII Всероссийской научно-практической конференции с международным участием 27-28 октября 2017 г. / Под ред. к.т.н., доцента С.А. Гончарова; к.ф-м.н., доцента Е.А. Дудник / Рубцовский индустриальный институт. – Рубцовск, 2017. – С. 65-72.

3. Docker CE release notes [Электронный ресурс] / Docker Documentation // URL: https://docs.docker.com/release-notes/docker-ce/ (дата обращения: 9.01.2018).

4. Distributed TensorFlow [Электронный ресурс] / TensorFlow // URL: https://www.tensorflow.org/deploy/distributed (дата обращения: 10.11.2017).

Using GPUs / TensorFlow [Электронный ресурс] // URL: https://www.tensorflow.org/tutorials/using_gpu (дата обращения: 10.