КОНТЕЙНЕРИ ВЪВ ВИСОКОПРОИЗВОДИТЕЛНИ ИЗЧИСЛИТЕЛНИ КЛЪСТЕРИ - КАЗУСИ

Стоян Пизов[1], Ана Пройкова[1,2]

[1]HPC Laboratory in the Laboratory Complex at the Sofia Tech Park, hpc-lab@sofiatech.bg
[1,2] Centre of Excellence in ICT "UNITe" at Sofia University, anap@phys.uni-sofia.bg

***Резюме**: Docker контейнерите не са първият избор в областта на високопроизводителните изчисления (HPC) поради необходимостта от ескалация на привилегиите при изпълнението на контейнерите, както и поради трудната интеграция със пакетни системи. Сингулярността е по-добра алтернатива на контейнерите Docker в HPC, разработени от лабораторията в Бъркли (САЩ). Ефективността на контейнеризираните приложения и възможността за настройване и конфигуриране на паралелни приложения с множество зависимости от външни (трети страни) софтуерни пакети се оценява за следните случаи: използва се само един изчислителен възел (32 ядра); множество софтуерни пакети в един контейнер за класификация на рентгенови изображения на гръден кош на пациенти с SARS-CoV-2 https://github.com/lindawangg/COVID-Net, използвайки платформата за машинно обучение с отворен код Tensorflow и библиотека за компютърно зрение OpenCV.*

***Ключови думи**: HPC, machine learning, computer vision, containers in HPC clusters*

1. Контейнери в изчислителни клъстери

Контейнерната (containers) архитектура като част от виртуализацията на ниво операционна система (ОС) има своята история, която започва с AIX Workload Partitions и Solaris Containers, но в последните няколко години започна да придобива особено голяма популярност с една от своите реализации Docker, поради широкото използване на софтуерната архитектура на разпределени микро-сервизи. Едно от основните предимства на контейнерната архитектура е високото ниво на абстракция при създаването на контейнерите, което свежда до минимум необходимостта да се познава операционната система, върху която контейнерите ще се реализират.

Съвременните Docker контейнери са напълно преносими върху различни платформи, а създаването им е изключително просто като се дефинира с помощта на структурирани текстови файлове. Поради широката си популярност, контейнерната архитектура има богат избор от публични хранилища с готови изображения (images) в контейнери, което позволява бързо и без особено усилие потребителите да инсталират, конфигурират и изпълнят контейнери.

За удобство предлагаме дефинициите на използваните термини:

Image: Изображението (заготовката) на контейнера е олекотен пакет (често tar архив), който включва целият необходим с софтуер като изпълними файлове, системни библиотеки и конфигурационни файлове, необходими за приложението да се изпълнява самостоятелно.

Dockerfile: Текстов файл, който включва инструкции за генериране на изображението (заготовката) на контейнера.

Container : Контейнерът е стартирано изображение. Възможно е потребителят да изпълнява едновременно множество изображения.

Документацията за използване на контейнерите docker е достъпна на https://docs.docker.com/get-started/overview/.

Въпреки широката популярност в разработката на съвременни софтуерни приложения за бизнеса, Docker контейнерите не са първи избор в областта на високопроизводителните изчисления (ВПИ, high performance computing HPC) поради необходимостта от ескалация на привилегиите при изпълнението на контейнерите както и трудната интеграция със системите за пакетна обработка на задачите (batch systems).

Съществува обаче алтернатива на Docker контейнерите в областта на HPC, която се разработва активно от лабораторията Бъркли (Lawrence Berkeley National Laboratory). Singularity е контейнерна архитектура, която се изпълнява в потребителското пространство (user space). Предимства на Singularity са, че не изисква ескалация на привилегиите, съвместим е с контейнерите на Docker и не изисква сервизен демон за разлика от Docker. Това са и причините Singularity да бъде избран като софтуерен продукт за контейнеризация, чиято ефективност бе оценена на изчислителният клъстер Nestum. Singularity версия 3.4.0 беше инсталирана и настроена като софтуерен модул. Изчислителният клъстер Nestum е хомогенен CPU базиран изчислителен клъстер. Повече информация за клъстера, инсталирания софтуер и възможностите за използването му е достъпна на http://hpc-lab.sofiatech.bg/.

2. Сравнение на ефективността на изпълнение на MPI приложение, компилирано в контейнер Singularity върху един изчислителен възел

Когато стрес-тестът се изпълнява само на един възел се изключва комуникацията между отделните възли, което дава възможност да се оцени по-точно влиянието на контейнеризацията. За сравнение на паралелната ефективност беше избрана версия на стрес теста LINPACK с отворен код HPL, https://www.netlib.org/benchmark/hpl/ , който измерва броя математични операции на числа с двойна точност (double precision) за единица време (flops). Ефективността на LINPACK стрес-теста се дефинира като отношението на

очакваната (теоретичната) паралелна ефективност Rmax и измерената пикова стойност Rpeak :

ε = Rpeak / Rmax

Очакваната теоретична производителност Rmax на един възел може да бъде пресметната с помощта на формулата

Rmax = Number of nodes × Number of cores per node ×
AVX2 base frequency × Number of DP operation per cycle

Броят на ядрата в един изчислителен възел в клъстера Nestum е 32, базовата честота е 1.9, а броят на операциите с двойна точност в цикъл е 16.

Rmax= 1 × 32 × 1.9 x 16 = 972.8 Gflops

Софтуерните пакети, използвани за компилиране на HPL стрес-теста:

- Базова операционна система: Ubuntu 16.04 LTS
- Компилатори: Intel Parallel Studio XE 2017
- Библиотеки за математични пресмятания: BLAS и LAPACK
- Паралелна среда OpenMPI 4.0.3
- LINPACK стрес тест HPL 2.3

За определяне оптималните параметри на изпълнение на HPL стрес-теста беше използван онлайн генератор на HPL.dat https://www.advancedclustering.com/act_kb/tune-hpl-dat-file/

2.1 Резултати за един изчислителен възел

Таблица 1: Сравнение на ефективността на изпълнението в контейнер Singularity с естественото (нативно) изпълнение

Начин на изпълнение	Пикова производителност R_{peak}	Паралелна ефективност ε
Приложение в контейнер	851.2 GFlops	87.5%
Нативно приложение	847.7 GFlops 87%	87,00%
Prog. Code	Програмен код	10 pt Font: Consolas

Получените резултати демонстрират, че не се наблюдава разлика в паралалелната ефективност между приложения, компилирани от изходен код и такива, които са компилирани и изпълнени в контейнер.

В изследвания случай, можем да приемем, че изпълнението в контейнер на малки паралелни задачи е добро и може да се препоръча.

3. Създаване на комплексно приложение изискващо множество софтуерни пакети - Tensorflow и OpenCV

Често в практиката се налага да се инсталират софтуерни пакети със сложни зависимости, което е трудоемка задача която отнема време и увеличава броя на софтуерните пакети, които трябва да бъдат поддържани от системните администратори на клъстера. Също така, потребителите могат да имат изисквания към версии на софтуера, който е несъвместим с базовата операционна система на клъстера. Контейнеризацията предоставя решение на тези проблеми като автоматизира до голяма степен инсталацията, компилирането и конфигурирането на софтуерните пакети.

Като един примерен случай, който демонстрира инсталацията на множество софтуерни пакети в един контейнер беше избран проекта за класификация на рентгенови изображения на гръдният кош на пациенти със SARS-CoV-2 https://github.com/lindawangg/COVID-Net с помощта на платформата за машинно обучение с отворен код Tensorflow и библиотеката за компютърно зрение OpenCV. Проектът е реализиран на програмният език Python3 и има следните софтуерни зависимости:

- Tensorflow 1.13 or 1.15
- OpenCV 4.2.0
- Python 3.6
- Numpy
- Scikit-Learn
- Matplotlib
- PyDicom
- Pandas
- Jupyter

Беше използван следният Singularity файл за създаване на контейнер, който включва изброените пакети:

```
#Singularity
Bootstrap: docker
From: ubuntu:18.04
%help
```

```
        COVID-Net container image
%files
        Anaconda3-2020.02-Linux-x86_64.sh /downloads/
%post
        apt update
        apt install -y libgl1-mesa-dev \ vim
        bash /downloads/Anaconda3-2020.02-Linux-x86_64.sh \
                -b \
                -p /usr/local \
                -f
        conda create -y -n tf \
                && . /usr/local/etc/profile.d/conda.sh \
                && conda activate tf \
                && conda install -y -c anaconda tensorflow=1.15.0 \
                && conda install -y -c conda-forge opencv \
                && conda install -y -c conda-forge scipy \
                && conda install -y -c conda-forge matplotlib \
                && conda install -y -c conda-forge notebook \
                && conda install -y -c conda-forge jupyter \
                && conda install -y -c conda-forge pydicom \
                && conda install -y -c conda-forge pillow \
                && conda install -y -c conda-forge scikit-learn \
                && conda install -y -c anaconda pandas
        rm -rf /download
        apt clean
        echo ". /usr/local/etc/profile.d/conda.sh" >>
$SINGULARITY_ENVIRONMENT
%runscript
.       /usr/local/etc/profile.d/conda.sh
        conda activate tf
        exec /bin/bash "$@"
```

За генерирането на изображението бяха използвани следните команди:

```
module load singularity
wget https://repo.anaconda.com/archive/Anaconda3-2020.02-Linux-x86_64.sh
singularity build --fakeroot covid-net.simg Singularity
```

Следвайки инструкциите, невронната мрежа беше тренирана с помощта на следната серия от команди веднъж когато множество от снимки на пациенти беше изтеглено:

```
module load singularity
git clone
cd covid-net
singularity run covid-net.simg -c "cd COVID-Net/data \
&& python train_tf.py \
--weightspath models/COVIDNetv2 \
--metaname model.meta_train \
--ckptname model-2069 \
--trainfile train_split_v2.txt \
--testfile test_split_v2.txt"
```

3.1 Оценката (assemement) на снимките на пациентите.

Нека предположим, че разполагаме с рентгенова снимка на пациент **x-ray.png** тя може да бъде оценена с помощта на следните команди:

```
DATA_DIR=”$PWD/COVID-Net/data”
cp x-ray.png $DATA_DIR/test_cases/
singularity run covid-net.simg -c \
    "cd ${DATA_DIR} \
    && python inference.py --weightspath models/COVIDNetv2 \
            --metaname model.meta_eval \
            --ckptname model-2069 \
            --imagepath test_cases/x-ray.png"
```

3.2 Резултат

Успешна демонстрация на възможността за създаване на контейнери, включващи множество софтуерни пакети без нужда от инсталация на допълнителен софтуер.

Изпълнението на командите има специфичен синтаксис, който изисква допълнително разучаване, като например потребителят трябва да се запознае с командите на singularity: exec и shell, които са лесно достъпни от https://singularity.lbl.gov/docs-run

Заключение

От направените тестове за производителност може да се каже, че въпреки началният си етап на развитие, контейнеризацията на HPC приложенията има голям потенциал за приложение в областта на HPC. Предимствата са липса на деградация в производителността, платформена независимост на контейнерните приложения, опростена системна поддръжка на софтуерните пакети на изчислителния клъстер.

Благодарности

Авторите благодарят за използване на HPC клъстера NESTUM по договор със СНИРД-СТП, .№ Д-140-2019/03.12.2019г. Изразяват благодарност за частична финансова подкрепа от Националната научна програма Информационните и комуникационни технологии в науката, образованието и сигурността, финансирана от Министерството на образованието и науката.

Литература

1. https://docs.docker.com/get-started/overview/.
2. https://www.netlib.org/benchmark/hpl/
3. https://singularity.lbl.gov/docs-run
4. http://hpc-lab.sofiatech.bg/

CONTAINERS IN HIGH PERFORMANCE COMPUTERS – CASE STUDIES

Abstract: Docker containers are not the first choice in the field of high performance computing (HPC) due to the need for escalation of the privileges in the implementation of the containers as well as the difficult one integration with batch systems. Singularity is a better alternative to Docker containers in HPC developed by the Berkeley Laboratory (Lawrence Berkeley National Laboratory). The performance of containerized applications and the ability to set up and configure parallel applications with multiple dependencies on external (third party) software packages is assessed for the following cases: one compute node is only used (32 cores); multiple software packages in one container for classification of chest X - ray images of patients with SARS-CoV-2 https://github.com/lindawangg/COVID-Net using the platform for Tensorflow open source machine learning and computer vision library OpenCV.