Methods of Cloud Computing

Winter Term 2018/2019

Practical Assignment No. 2

Due: 20.12.2018 23:59

The primary goal of this assignment is to gain insight into performance characteristics of virtualization by benchmarking different virtualization and isolation techniques and comparing and evaluating the benchmark results. The secondary goal is to gain an understanding of isolation features provided by modern Linux kernels and how they can be employed to create your own container solution.

1. Prepare Virtual Resources

Install QEMU (and the KVM kernel module), Docker, and [optional] the Rumpkernel Unikernel tool-chain (rumprun) on a Linux host. Run all experiments on the same underlying host. It can be a physical machine you own (e.g. your Laptop) or a virtual machine on one of the IaaS clouds.

Study the QEMU documentation on how to create disk images. Create or obtain a disk image with a current x86_64 GNU/Linux distribution installed. You need to be able to access network resources running in the QEMU guest from your host system.

Document your steps and issued command line commands in a commented file **listing-setup.txt**.

Hints:

- We suggest using a minimal installation of Debian stable, but you are free to use other distributions.
- You can download an existing QEMU image, e.g. from Google Cloud, as long as you document the source and steps you took.
- You can use the command-line tool `script` to record all inputs and outputs typed in your terminal, including control sequences. Clean up the files before submitting your listings (remove redundant commands and so on).

Outputs:

- File **listing-setup.txt**
  - Containing all commands and inputs you used for installing the tools and preparing your VM image, including comments explaining what the commands do
- File **environment.txt**
  - Containing details on your test systems (guest and hosts):
    e.g. Software versions, cpu features as of `/proc/cpuinfo`, …

2. Prepare Performance Benchmarks

Write benchmarking scripts for the main system resources CPU, memory and disk.

- The scripts take no parameters (except Nginx) and each script must run for about 10 seconds while benchmarking the respective system resource.
- In case you obtain multiple measurements during that time, use the median of those values as the final output.
- Except for the disk and nginx benchmarks, your scripts are not allowed to use external tools. All files required by your benchmarks must be submitted together with the scripts.
- Afterwards the script must output one single floating-point value (e.g. 14.533)
- The resulting value depends on the benchmarked resource and used tool.

CPU Benchmark
LINPACK is a popular CPU benchmark in the supercomputing community. Derive the implementation of your benchmark script for CPU compute power from the `linpack.sh` script provided on the lecture website.

Memory Benchmark
Create a benchmarking script for memory operations based on the `memsweep.sh` script provided on the lecture website.

Disk Benchmark
Benchmark random (Operations/s) write access to the (virtualized) disk. Use and install the `fio` tool in your VMs and container (no `dd` benchmark required).

Fork Benchmark
Write a parallel computation of an integer range sum. Your program **forksum.c** receives a start and end value of an integer range and prints the sum of all numbers in that range on stdout. Unless start and end are equal your program should `fork` twice, with each child computing a subrange and the parent process reading the results from a `pipe`. You can use the `dup` system call for redirecting stdout and stdin to/from your pipe. An example can be found in the slides for this assignment.

Nginx Benchmark
Use the Nginx webserver to serve one large (500 MiB or larger) static file via HTTP. Measure the time required to access the file with (at least) two parallel requests from your host. The benchmark script receives as input the IP of the container/VM that executes the target Nginx server. It is the only benchmark script that is started outside the target container/VM.

Outputs:
- Scripts:
  - `measure-cpu.sh`
  - `measure-mem.sh`
  - `measure-disk-random.sh`
  - `measure-fork.sh`
  - `measure-nginx.sh`
- All files required by your scripts

3. Execute Performance Benchmarks

Execute the prepared benchmarks on the following platforms:

- Native on your host
- Virtualization with dynamic binary translation (qemu)
- Virtualization with hardware support (qemu-kvm)
- In a Docker container
- *[optional]* Virtualization without guest OS (rumpkernel, qemu)
  (You do not need to run the fork benchmark on rumpkernel.)

Each benchmark must be executed 48 times. Try to flush your system caches between measurement. This results in:

- (4 platforms * 5 benchmarks *[+ 4 rump-benchmarks]*) * 48 runs
  = 960 *[1152]* total measurements

The result of the invocations must be 20 [24] CSV (comma separated values) files (one per benchmark) that contain the collected values and the Unix timestamp of when each measurement was done.

Example of such a CSV file:

```
time,value
1542717365,677.4445
1542717366,746.3566
1542717367,144.6778
1542717368,143.5634
1542717369,435.7880
...
```

After taking all benchmarks, look at your results (maybe plot them) and answer the questions below.

Outputs:

- 20 [24] CSV-files for the five platforms, covering the 5 benchmarks, excluding fork on rump:
  - `[native|qemu|kvm|docker/`*`rump`*`]-[cpu|mem|disk-random|fork|nginx].csv`

- Five Docker files `[cpu|mem|disk-random|fork|nginx].Dockerfile`
- Text file `answers.txt`
  - Including all the answers for the questions below. Please use max. 200 words per question

CPU benchmark questions:

1. Look at your LINPACK measurements. Are they consistent with your expectations? If not, what could be the reason?

Memory benchmark questions:

1. Look at your memsweep measurements. Are they consistent with your expectations? If not, what could be the reason?

Disk benchmark questions:

1. Look at your disk write measurements. Are they consistent with your expectations? If not, what could be the reason?
2. Which disk format did you use for qemu? How do you expect this benchmark to behave differently on other disk formats?

Fork benchmark questions:

1. Look at your fork sum measurements. Are they consistent with your expectations? If not, what could be the reason?
2. [not optional] Why did we exclude this benchmark from the Rump Unikernel? How can you adapt the experiment for this platform?

Nginx benchmark questions:

1. Look at your nginx measurements. Are they consistent with your expectations? If not, what could be the reason?
2. How do your measurements relate to the disk benchmark findings?

4. My Container Tool

Develop your own minimal container solution, orchestrated with a tool named `myct`. You are free to develop this tool using Shell, Python or C++ (provide Makefile). Place all source files into a folder named `myct`.

Your implementation should:

- Setup a root file system (e.g. download a tarball, or using `debootstrap`)
- Allow existing host directories to be mapped read-only into the container
- Limit filesystem access (e.g. using `chroot`)
- Create a minimal kernel namespace with `unshare` (e.g. prevent signals to be send to other processes outside of the container)
- Let new processes in the container join existing namespaces, identified via `/proc/<pid>/ns/<kind>` (e.g. using `nsenter`)
- Limit resource usage (memory, cpu, …) with control groups (e.g. directly via `/sys/fs/cgroup/<controller>/…/<controller.key>`, or `libcgroup`)

Provide at least the following command line interface:

Creates a container in the given directory (downloads and extracts root file system)
```
$ myct init <container-path>
```

Mounts a host directory read-only into the container at given destination
```
$ myct map <container-path> <host-path> <target-path>
```

Runs the file exectuable in container with passed arguments
```
$ myct run <container-path> [options] <executable> [args...]
```
with options being:
```
--namespace <kind>=<pid>
--limit <controller.key>=<value>
```

Use `script` to record one session, where you

- create a new container
- run a self-written program, exceeding a specified cgroup limit

Place your test program and the resulting `typescript` file in a subfolder named `test`. Make sure to not include any sensitive data or passwords.

5. Submission Deliverables

Use the OpenSubmit system running at https://www.dcl.hpi.uni-potsdam.de/submit/ to submit your solutions. Every group member must register on the OpenSubmit platform, but only one member should submit the final solution. **Select your co-authors when creating a new submission.** Your solution will be validated automatically, you can resubmit your solution multiple times until it passes the tests.

Expected contents of your solution zip file:

- `listing-setup.txt`
  - **1 commented listing file, 20 points *[+5 for rump]***
- `environment.txt`
  - **1 info file, 5 points *[+1 for rump]***
- `answers.txt`
  - **8 points**
- `[cpu|mem|disk-random|fork|nginx].Dockerfile`
  - **5 Docker files, 2 points each**
- `measure-[cpu|mem|disk-random|fork|nginx].sh`
  - **5 script files, 4 points each**
- `[native|qemu|kvm|docker/`*`rump`*`]-[cpu|mem|disk-random|fork|nginx].csv`
  - **20 CSV files, 0.5 point each *[+ 4 for rump]***
- `Any additional files required by your benchmark scripts`
- `myct/*  (C++-Sources+Makefile or Python/Shell-Script(s))`
  - **code for init, 2 points**
  - **code for map, 3 points**
  - **code for run (chroot), 3 points**
  - **code for run (new namespace), 4 points**
  - **code for run (join namespace), 4 points**
  - **code for run (resource control), 4 points**
- `myct/test/typescript`
  - **1 session recoding, 2 points**
- `myct/test/*`
  - **1 limit test program, 3 points**
- `Any additional files required by your myct limit test program`

Total points: 100 + [10 for rump]