FragVisor Artifact Evaluation Guide

Jan 2023

A Artifact Appendix

A.1 Abstract

The artifact contains the source code of FragVisor's host and guest Linux kernels, FragVisor hypervisor based on kvmtool (lkvm), and a plethora of helper scripts. The system has been deployed on baremetal hardware on a cluster of servers connected via RDMA.

A.2 Description & Requirements

Fragvisor consists of three software modules. The distributed hypervisor built on top of kvmtool (lkvm) [1], a KVM-only, type-2, paravirtualized hypervisor. The host Linux kernel, heavily modified to support FragVisor. The guest Linux kernel, which is a vanilla kernel, but an additional version that enables certain optimisations to improve performance is also made available. The project also comes with a plethora of helper scripts. Scripts automates experiments, including initramfs images creation.

A.2.1 kvmtool Hypervisor

The source code for the modified kvmtool can be found here `https://github.com/systems-nuts/fragvisor-kvmtool`. Many files have been modified to support the distributed VM, mostly using the thread migration API to replicate the VM object across all the hosts.

The repository also contains initramfs images as well as build and run scripts. Two notable scripts are `msg_pophype4node.sh` which establishes the messaging layer between all the hosts, and `run.sh` which is used to build and start the guest VM.

A.2.2 Host and Guest OS Kernels

They share the same source tree at `https://github.com/systems-nuts/fragvisor-linux`. The kernel is heavily modified. Besides the task migration code and the dis-

[1]https://github.com/kvmtool/kvmtool

tributed shared memory protocol in `kernel/popcorn/`, there is the main FragVisor extensions in `kernel/popcorn/hype_kvm.c` as well as a plethora of other modifications in the virtualisation subsystem in the kernel, namely the *KVM* subsystem, both architecture-independent parts as well as the *x86* architecture-specific portions in `arch/x86/kvm/`.

A.3 How to access

The source tree for the all the components above can be fetched from `https://github.com/systems-nuts/FragVisor/blob/main/README.md`

A.4 Hardware dependencies

FragVisor is deployed on a cluster of computers composed of multiple servers equipped with a Xeon E5-2620 v4 (2.1 GHz, 8/16 cores/threads) and 32 GB of RAM. Servers run Linux 4.4 as the host kernel, and are connected via 56Gbps InfiniBand using Mellanox Connect-X4 and an Infiniband Switch. All nodes are also connected via 1Gbps Ethernet switch.

A.5 Software dependencies

None.

A.6 Benchmarks

The benchmarks used are based on NPB, ApacheBench, LEMP and OpenLambda (more on this later).

A.7 Set-up

For a full guide including how to use IPMI serial console, refer to the README file on the github repository [2].

First we need to compile and install the host kernel on every node of the cluster. For each node in the *echo* cluster (*echo5, echo4, echo1, echo0*), run:

```
git clone --recurse-submodules git@github.com:systems-nuts/
    FragVisor
cd FragVisor/fragvisor-linux
```

Before you start building the kernels, make sure that the macro `CONFIG_POPCORN_ORIGIN_NODE` in `include/linux/popcorn/debug.h` is only defined for the origin node. All the other nodes must **NOT** define it and it should be commented. Then, run:

[2] https://github.com/systems-nuts/FragVisor/blob/main/README.md

```
    make -j17
    sudo make modules_install
    sudo make install
    reboot
```

The next step is to load the messaging layer. from either *echo0* or *echo5*, run:

```
    cd FragVisor/fragvisor-kvmtool
    ./msg_pophype4node.sh
```

This will compile and load the messaging layer kernel modules. After a successful loading, you should observe on the IPMI serial console that RDMA connection was established between all nodes.

A.8 Evaluation workflow

To run experiments, navigate to `fragvisor-kvmtool`. We will be using the `run.sh` script. Every experiment will use a different kvmtool launch command with different arguments and initramfs images.

A.9 Major Claims

- (C1): FragVisor achieves higher throughput for longer requests than the state-of-the-art GiantVM. This is proven by the experiment (E1) described in section 7.2 whose results are illustrated in Figure 12 in the paper.
- (C2): FragVisor performs faster than the state-of-the-art GiantVM in every phase of OpenLambda serverless computing. This is proven by the experiment (E2) described in section 7.2 whose results are illustrated in Figure 13 in the paper.
- (C3): With DSM bypass, FragVisor I/O delegation can achieve very high performance to offset the effects of distribution. This is proven by the experiment (E3) described in section 7.1 whose results are illustrated in Figures 6 and 7 for networking and storage, respectively.

A.10 Experiments

A.10.1 E1. Running the LEMP experiments

For 4 nodes, use the following kvmtool command:

```
    sudo bash -c "./lkvm run -a 1 -b 1 -x 1 -y 1 -w 4 -i $USER/c/
        ramdisk_lemp_4php.gz -k $KERNEL_PATH/arch/x86/boot/bzImage
         -m 20480 -c 4 -p \"root=/dev/ram rw init=/init2 fstype=
        ext4 spectre_v2=off nopti pti=off numa=fake=4 percpu_alloc
        =page -no-kvm-pit-reinjection clocksource=tsc\" --network
        mode=tap,vhost=1,guest_ip=10.4.4.222,host_ip=10.4.4.221,
        guest_mac=00:11:22:33:44:55"
```

Then follow with

```
root@(none):~# Kill nginx and PHP
echo5$ cd FragVisor/experiments/nginx-new/nginx-1.16.1
echo5$ ./auto_configure_make_scp.sh
root@(none):~# nginx -c /usr/local/nginx/conf/nginx.conf
root@(none):~# php-fpm7.2 --fpm-config /etc/php/7.2/fpm/php-
    fpm.conf &
```

In the VM, there are N number of php-fpm, php worker threads, where N is the number of nodes. Each of these threads have to be pinned to a vCPU.

```
root@(none):~# ps aux |grep php
# Pin each of the php workers on a vCPU!
root@(none):~# taskset -p 0x1 312
root@(none):~# taskset -p 0x2 313 (Stop here for 2 Nodes)
root@(none):~# taskset -p 0x4 314 (Stop here for 3 Nodes)
root@(none):~# taskset -p 0x8 315
# Pin nginx master and worker on vCPU0
root@(none):~# ps aux |grep nginx
root@(none):~# taskset -p 0x1 112
root@(none):~# taskset -p 0x1 120
```

The customized LEMP experiment uses a php-benchmark script with variable iterations of each benchmark. There are 4 variations of this which have to be copied to the VM from the host node. Copy the scripts to VM:

```
echo5:$ scp -r FragVisor/experiments/php-script/* root@10
    .4.4.222:/var/www/travel_list/routes/
root@(none):~# cd /var/www/travel_list/routes
echo5:$ cd FragVisor/experiments/run_lemp
echo5:$ ./run_lemp_no_stats.sh
# Check output
echo5:$ cd run_lemp_no_stat/output_dir
echo5:$ time_taken_out
```

A.10.2 E2. Running OpenLambda Experiments

For a 4 nodes configuration, use this kvmtool command:

```
sudo bash -c "./lkvm run -a 1 -b 1 -x 1 -y 1 -w 4 -i $USER/c/
    ramdisk_for_openlambda200625.gz -k $KER NEL_PATH/arch/x86/
    boot/bzImage -m 32768 -c 4 -p \"root=/dev/ram rw fstype=
    ext4 spectre_v2=off nopti p ti=off numa=fake=4
    percpu_alloc=page -no-kvm-pit-reinjection clocksource=tsc
    \" --network mode=tap,vho st=1,guest_ip=10.4.4.222,host_ip
    =10.4.4.221,guest_mac=00:11:22:33:44:55"
```

Then follow with the instructions below to run the experiment:

```
echo5:$ cd FragVisor/experiments/open-lambda/
echo5:$ ./copy4nodes.sh # passwd: popcorn
echo5:$ ssh popcorn@10.4.4.222
root@(none):~# sudo su
root@(none):/home/popcorn/$ ./folders4nodes.sh
root@(none):/home/popcorn/$ cd ~/open-lambda0 && taskset 0x1
    ./ol worker --path=l0 -d
root@(none):~/open-lambda0$ cd ~/open-lambda1 && taskset 0x2
    ./ol worker --path=l1 -d # Stop here for 2 node setup
root@(none):~/open-lambda0$ cd ~/open-lambda2 && taskset 0x4
    ./ol worker --path=l2 -d # Stop here for 3 node setup
root@(none):~/open-lambda0$ cd ~/open-lambda3 && taskset 0x8
    ./ol worker --path=l3 -d
echo5:$ cd FragVisor/experiments/run_openlambda
echo5:$ ./run_openlambda_no_stat_4_node.sh
```

A.10.3 E3. Running Network Delegation Overhead Experiment

For 4 node case choose this in `run.sh`:

```
sudo bash -c "./lkvm run -a 1 -b 1 -x 1 -y 1 -w 4 -i $USER/c/
    ramdisk_NPB_ATC.gz -k $KERNEL_PATH/arch/x86/boot/bzImage -
    m 32768 -c 4 -p \"root=/dev/ram rw init=/init2 fstype=ext4
     spectre_v2=off nopti pti=off numa=fake=4 percpu_alloc=
    page -no-kvm-pit-reinjection clocksource=tsc\" --network
    mode=tap,vhost=1,guest_ip=10.4.4.222,host_ip=10.4.4.221,
    guest_mac=00:11:22:33:44:55"
```

Then, to compile kvmtool and the guest kernel and launch the VM:

```
echo5:~/fragvisor_kvmtool$ ./run_echo.sh 1 1 0
```

The following are the commands for running the network delegation experiment (note that `root@(none)` is the VM):

```
root@(none):~# ps aux | grep nginx
# For the first case pin the nginx worker and master on vCPU 0
root@(none):~# taskset -p 0x1 112 #(master)
root@(none):~# taskset -p 0x1 120 #(worker)

echo:$ cd FragVisor/experiments/popcorn-utils (Run this on
    echo)
echo:$ ./ab_micro_diff_sizes_lan_0x1_all.sh <name_for_the_run>

# Check output in:
echo:$ cd pophype_ab_micro_diff_sizes_lan/
```

```
root@(none):~# ps aux |grep nginx
# For the second case we pin the nginx master and worker on
    vCPU 1
root@(none):~# taskset -p 0x1 112 #(master)
root@(none):~# taskset -p 0x2 120 #(worker)

fox3:$ cd FragVisor/experiments/popcorn-utils
fox3:$./ab_micro_diff_sizes_lan_0x2_all.sh <name for the run>
fox3:$ cd pophype_ab_micro_diff_sizes_lan/
```