# 超算平台 Singularity 容器运行简介

张文帅

中国科学技术大学超级计算中心

2023 年 05 月 29 日

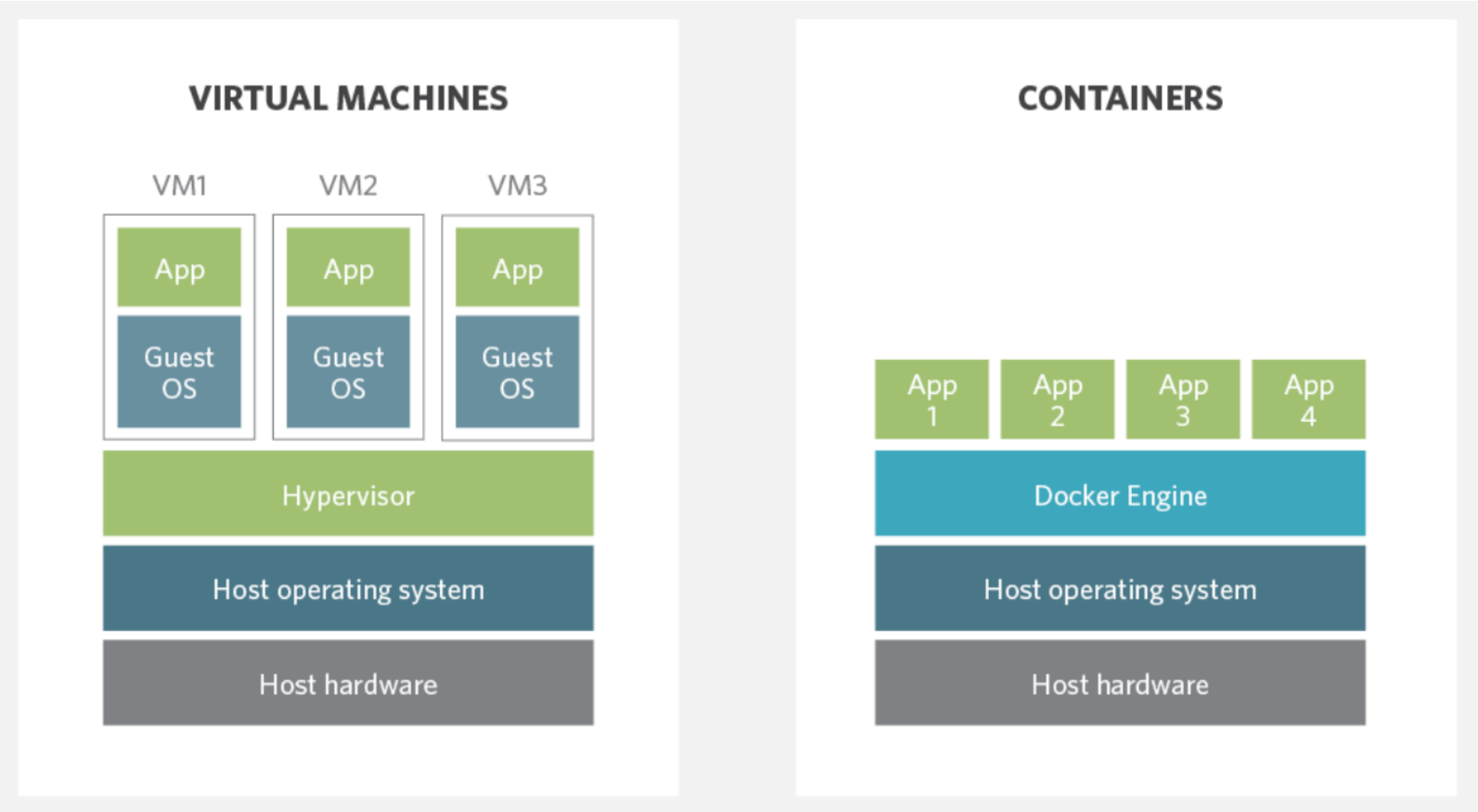
# 目录


1. 容器简介
   - 容器与虚拟机
   - 容器平台对比
   - 性能与迁移测试
2. Singularity 容器
   - Singularity sif/simg Image
   - Singularity Overlay Image
   - Singularity Command
   - Singularity & LSF
   - Singularity & SLURM

# Outline


1. 容器简介
   - 容器与虚拟机
   - 容器平台对比
   - 性能与迁移测试
2. Singularity 容器
   - Singularity sif/simg Image
   - Singularity Overlay Image
   - Singularity Command
   - Singularity & LSF
   - Singularity & SLURM

# 容器与虚拟机




以 Docker 容器为例

# 容器的优势

1. 解决应用对特定系统版本的依赖问题，使得多种系统版本下的程序可以在同一个平台无缝连续的运行。
2. 封装复杂依赖关系的应用，使得开发人员直接生成具有生产能力的镜像，减轻随工程量不断增加带来的持续上升运维压力（不过也增加了镜像系统的运维需求）。
3. 更有效的隔离资源，严格限定作业运行时可使用的资源空间，降低作业间的互相影响。
4. 通过作业迁移手段，增强作业运行崩溃时的断点续算能力，减少计算浪费，并增加节点的利用率。
5. 相比传统虚拟机，达到接近原生系统环境的性能。

# 容器分类

- 系统级容器

  **OpenVZ**

  具有强大的自我定制的 kernel，旧版本可在 CentOS6 系统基础上安装，最新版本更改为发行整套系统，不再支持基于 CentOS7 安装。尤其，Checkpoint/Restore 底层做了较大的变更，放弃自有的成熟的在 Kernel 内部工作的机制，改用基于用户层的 CRIU 技术，虽然 CRIU 社区与技术前景更好，但当前并不太成熟。

  **LXC/LXD**

  当前由 Ubuntu 主推，为 Docker 0.9 版本之前使用的容器底层技术，支持热迁移，使用标准 Linux 内核，不需要定制的补丁，能获得较好的上游社区支持。

- 应用级容器

  **Docker**

  IT 工业应用最多的容器方案，特色是标准化了使用 Dockerfile 创建容器的过程方法，使得应用容器更加方便易行，但是对于 HPC 应用，附加功能有些过于庞大，对 MPI 应用并不友好，同时很重要的安全隔离做的不好，在 HPC 这种裸跑用户应用的生产环境，很容易造成安全问题。

  **Shifter**

  由 NERSC 开发，较早且比较成熟的 HPC 下的容器方案，具有一个 Image Gateway，用户可直接从 dockerhub 选择容器或自己定制化，执行容器化的应用。安全隔离方面有所改进，但是仍有需要 root 权限的地方。其被 Torque、Slurm 等多种资源调度平台支持，只是系统的配置与用户使用仍有一些不方便。

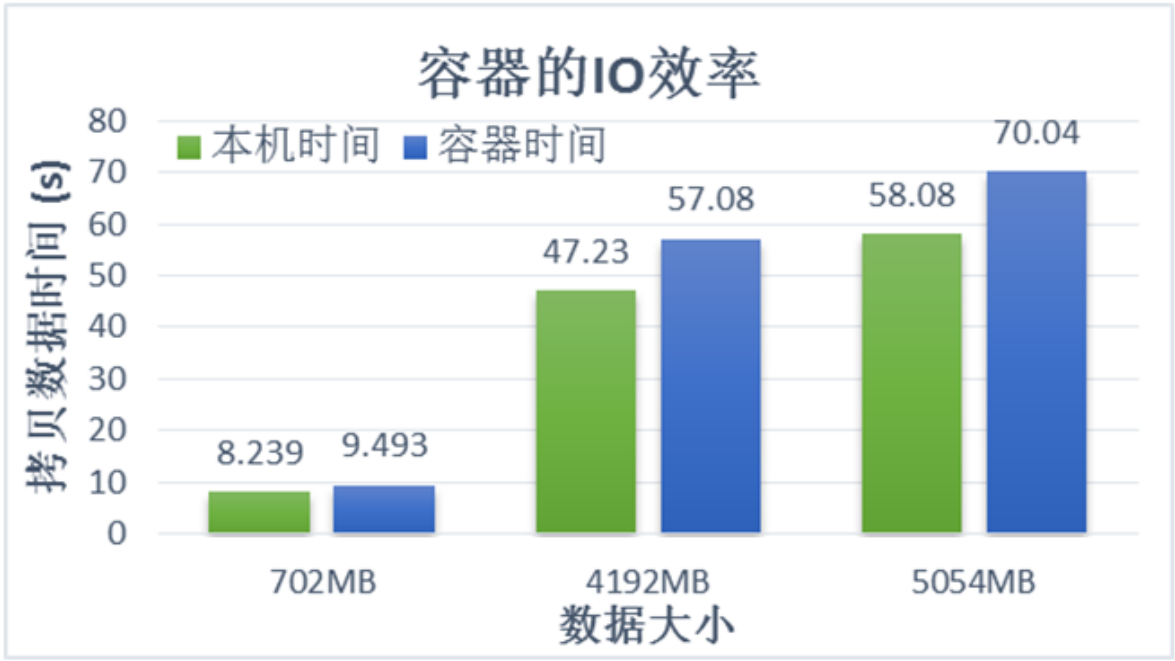
  **Singularity**

  定位与 Shifter 相似，但是在安全隔离上面做的最好，其仅限定在用户空间内启动并运行容器，所以应用的运行完全被隔离不会影响到系统的安全。其提供了从 Docker 镜像直接 pull 并转换为自有镜像格式的方法，也维护有一个自有的镜像库。用户的使用方式也易学易用。

# 更多容器比较

| Attribute | self-compile | virtual machine<br>hypervisor | Shifter<br>chroot | Singularity<br>priv. ns. | Docker<br>userns* | rkt<br>userns* | NsJail<br>userns | Charliecloud<br>userns |
|---|---|---|---|---|---|---|---|---|
| **Workflow (G1)** | | | | | | | | |
| User-defined kernel and settings | · | ✓ | · | · | · | · | · | · |
| Use package managers, e.g. `apt-get`, `yum` | · | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| No conflicts with host software | · | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Industry-standard image build | · | · | ✓ | · | ✓ | ✓ | · | ✓ |
| Reproducible image build | · | · | ✓ | ✓ | ✓ | ✓ | · | ✓ |
| **Resources (G2)** | | | | | | | | |
| No privileged or trusted daemons | ✓ | ✓ | · | ✓ | · | ✓ | ✓ | ✓ |
| No additional network infrastructure | ✓ | · | ✓ | ✓ | · | ✓ | ✓ | ✓ |
| Network filesystems see no UDSS metadata | · | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Direct device access | ✓ | · | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Direct filesystem access | ✓ | · | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Direct high-speed network access | ✓ | · | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Simplicity (G3)** | | | | | | | | |
| Implementation language | n/a | varies | C, Python, C++, sh | C, sh, Python | Go | Go | C | C, sh |
| Lines of code | n/a | varies | 19,000 | 11,000 | 133,000 | 52,000 | 4,000 | 800 |
| No resource manager-specific code | ✓ | ✓ | · | ✓ | ✓ | ✓ | ✓ | ✓ |
| No communication framework-specific code | ✓ | ✓ | ✓ | · | ✓ | · | ✓ | ✓ |
| No root operations on center resources | ✓ | · | · | · | · | ✓ | ✓ | ✓ |
| No guest supervisor process | ✓ | · | · | · | · | ✓ | · | ✓ |
| No cache, configuration, or other state | ✓ | · | · | · | · | · | ✓ | ✓ |

# 容器性能测试结果


容器的IO效率
本机时间
容器时间
拷贝数据时间 (s)
80
70
60
50
40
30
20
10
0
8.239
9.493
47.23
57.08
58.08
70.04
702MB
4192MB
5054MB
数据大小


VASP自洽运行时间(96原子)
本机时间
容器时间
计算时间(s)
40000
30000
20000
10000
0
1373
1445
5932
6155
14732
15101
32970
33563
3
5
7
9
计算参数：K-Points大小

# 容器作业的迁移测试

■ **Checkpoint 与 Restore 时间**

这里测试了不同进程数与算例数的 Gaussian 容器的 Checkpoint 和 Restore 时间，测试时的内存占用、临时计算数据大小以及测试结果见下表。检查点建立与重载时间与当时的存储占用大小直接正比相关。

| Gaussian 容器的 checkpoint 和 restore 时间 | | | | |
|---|---|---|---|---|
| CPU 数 x 同时运行算例数 | 内存 (GB) | 临时文件 (GB) | Checkpoint | Restore |
| 24x1 | 32 | 7.4 | 5 分 11 秒 | 3 分 32 秒 |
| 12x2 | 24 | 2.8 | 3 分 53 秒 | 2 分 39 秒 |
| 8x3 | 15 | 3.6 | 2 分 05 秒 | 1 分 31 秒 |
| 6x4 | 28 | 6 | 4 分 58 秒 | 3 分 17 秒 |

■ **热迁移时间与内存占用**

本次 Gaussian 程序热迁移的硬盘文件有 31GB，内存文件、计算的临时数据文件以及迁移时间见下表。此次测试数据中，尽管同时运行的算例数不同，但是总的线程数相同，且容器全部数据的存储占用相差不多，故而热迁移时间相差较小。

| Gaussian 容器热迁移时间 | | | |
|---|---|---|---|
| CPU 核数 x 运行算例数 | 内存(GB) | 临时文件(GB) | 热迁移时间 |
| 24x1 | 35GB | 3.3GB | 114 分 14 秒 |
| 8x3 | 31GB | 7.9GB | 109 分 34 秒 |
| 6x4 | 30GB | 6.2GB | 103 分 56 秒 |

# Outline


1. 容器简介
   - 容器与虚拟机
   - 容器平台对比
   - 性能与迁移测试
2. Singularity 容器
   - Singularity sif/simg Image
   - Singularity Overlay Image
   - Singularity Command
   - Singularity & LSF
   - Singularity & SLURM

# Singularity 镜像获取 I

## pull from Singularity Image:

```
$ singularity pull shub://vsoch/hello-world
```

## pull form Docker Image

```
$ singularity pull docker://godlovedc/lolcow
```

## build from Singularity shub or Docker docker://

```
$ singularity build hello-world.simg shub://vsoch/hello-world
$ singularity build lolcow.simg docker://godlovedc/lolcow
```

## build from recipe file (definition file)

```
# singularity build example_tf_gpu.simg Singularity.latest
# singularity build --sandbox /path/to/image-root-directory/ Singularity.latest
```

这里需要 Root 权限，但可以在自己机器上编译镜像（squashfs 格式）或镜像目录（sandbox）后上传。在 sandbox 模式下，生成容器就是在指定目录下生成一个系统 root 根目录（里面包含/bin /lib /etc 等），类似 chroot 工作模式，可对系统直接修改定制。

# recipe file (definition file) I

```
 1 bootstrap:docker
 2 From:nvidia/cuda:8.0-cudnn6-devel-ubuntu16.04
 3
 4 %environment
 5
 6     LD_LIBRARY_PATH=/host-libs:/usr/local/cuda/extras/CUPTI/lib64:/usr/local/cuda/lib64
 7     export LD_LIBRARY_PATH
 8     PATH=/usr/local/cuda/bin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
 9     export PATH
10
11 %post
12
13     export DEBIAN_FRONTEND=noninteractive && \
14         apt-get update && apt-get upgrade -y --allow-unauthenticated && \
15         apt-get install -y --allow-unauthenticated \
16             build-essential \
17             cmake \
18             cuda-drivers \
19             curl \
20             git \
21             libfreetype6-dev \
22             libpng12-dev \
23             libssl-dev \
24             libxpm-dev \
25             libzmq3-dev \
26             module-init-tools \
27             pkg-config \
28             python \
29             python-dev \
30             python-tk \
```

# recipe file (definition file) II

```
31              python3 \
32              python3-dev \
33              python3-tk \
34              rsync \
35              software-properties-common \
36              unzip \
37              zip \
38              zlib1g-dev \
39              openjdk-8-jdk \
40              openjdk-8-jre-headless \
41              vim \
42              wget
43
44      apt-get clean
45      rm -rf /var/lib/apt/lists/*
46
47      # bazel is required for some TensorFlow projects
48      echo "deb [arch=amd64] http://storage.googleapis.com/bazel-apt stable jdk1.8" >/etc/apt/
            sources.list.d/bazel.list
49      curl https://bazel.build/bazel-release.pub.gpg | apt-key add -
50
51      export DEBIAN_FRONTEND=noninteractive && \
52          apt-get update && \
53          apt-get install -y --allow-unauthenticated \
54              bazel
55
56      curl -O https://bootstrap.pypa.io/get-pip.py && \
57          python get-pip.py && \
58          rm get-pip.py
59
```

# recipe file (definition file) III

```
60      pip --no-cache-dir install \
61              h5py \
62              ipykernel \
63              jupyter \
64              matplotlib \
65              numpy \
66              pandas \
67              Pillow \
68              scipy \
69              sklearn
70
71      python -m ipykernel.kernelspec
72
73      echo "/usr/local/cuda/lib64/" >/etc/ld.so.conf.d/cuda.conf
74      echo "/usr/local/cuda/extras/CUPTI/lib64/" >>/etc/ld.so.conf.d/cuda.conf
75
76      # Install TensorFlow GPU version
77      pip uninstall tensorflow-gpu || true
78      pip install --upgrade tensorflow-gpu==1.4
79
80      # keras
81      pip install --upgrade keras
82
83      ##############################
84      # now do the same for python3
85
86      curl -O https://bootstrap.pypa.io/get-pip.py && \
87          python3 get-pip.py && \
88          rm get-pip.py
89
```

# recipe file (definition file) IV

```
90      pip3 --no-cache-dir install \
91              h5py \
92              ipykernel \
93              jupyter \
94              matplotlib \
95              numpy \
96              pandas \
97              Pillow \
98              scipy \
99              sklearn
100
101     python3 -m ipykernel.kernelspec
102
103     # Install TensorFlow GPU version
104     pip3 uninstall tensorflow-gpu || true
105     pip3 install --upgrade tensorflow-gpu==1.4
106
107     # keras
108     pip3 install --upgrade keras
109
110     ##############################
111
112     # make sure we have a way to bind host provided libraries
113     # see https://github.com/singularityware/singularity/issues/611
114     mkdir -p /host-libs /etc/OpenCL/vendors
115     echo "/host-libs/" >/etc/ld.so.conf.d/000-host-libs.conf
116
117     # required directories
118     mkdir -p /cvmfs
119
```

# recipe file (definition file) V

```
120     # root
121     cd /opt && \
122         wget -nv https://root.cern.ch/download/root_v6.10.02.Linux-ubuntu16-x86_64-gcc5.4.tar.
                gz && \
123         tar xzf root_v6.10.02.Linux-ubuntu16-x86_64-gcc5.4.tar.gz && \
124         rm -f root_v6.10.02.Linux-ubuntu16-x86_64-gcc5.4.tar.gz
125
126     # xrootd
127     cd /opt && \
128         wget http://xrootd.org/download/v4.7.1/xrootd-4.7.1.tar.gz && \
129         tar xzf xrootd-4.7.1.tar.gz && \
130         cd xrootd-4.7.1 && \
131         mkdir build && \
132         cd  build && \
133         cmake /opt/xrootd-4.7.1 -DCMAKE_INSTALL_PREFIX=/opt/xrootd -DENABLE_PERL=FALSE && \
134         make && \
135         make install && \
136         cd /opt && \
137         rm -rf xrootd-4.7.1.tar.gz xrootd-4.7.1
138
139     # stashcp
140     cd /opt && \
141         git clone https://github.com/opensciencegrid/StashCache.git
142
143     # build info
144     echo "Timestamp:" `date --utc` | tee /image-build-info.txt
```

# Overlay 用户空间二层镜像 I

## 二层镜像意义与原理

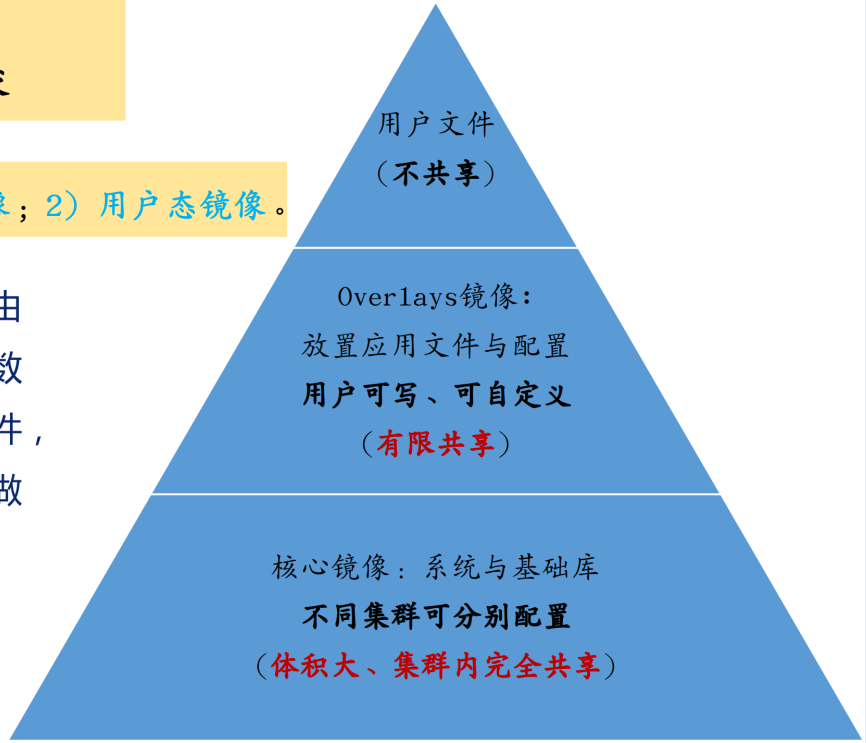
**问题**

1. 解决用户自定义镜像的需求
2. 解决容器系统镜像的共享与数据压缩需求

**方法**

基于 Overlays 技术构建两层镜像： 1）系统镜像；2）用户态镜像。

系统镜像由管理员负责维护，对用户只读；用户态镜像可以由用户自由写入。前者空间中更多放置用户间共享的系统与函数库组件；后者空间中更多放置用户个性化的应用、配置与插件，这部分亦属于应用的一部分，可以在使用同一应用的用户间做有限的共享。两层镜像组成了应用运行环境。




**效果**

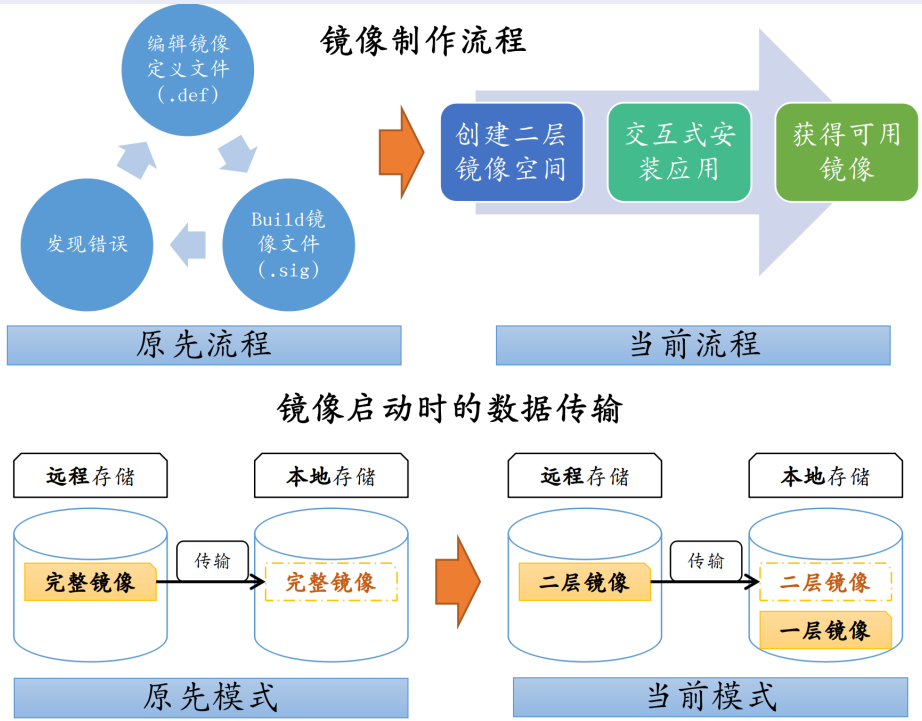
在共享同一个底层的系统镜像的情况下，方便用户构建不同应用版本的Singularity容器。

# Overlay 用户空间二层镜像 II

## 二层镜像示意图

**基于Overlay构建二层应用镜像**，解决容器使用上的如下难点：

1. 用户在构建应用镜像时可以交互式的编辑编译源程序文件，方便即时的修正错误，避免频发构建的时间浪费。
2. 用户为不同版本、甚至不同种类的应用构建的二层镜像，可以共享同一个底层的系统镜像文件，用户在加载容器应用时，可分别独立的指定一层系统镜像与二层应用镜像的位置。因此，一层系统镜像文件，通常也是容器镜像的主要数据部分，可以缓存到计算节点本地，作为共享数据使用，高效的利用了存储空间，也减少了容器应用启动时的带宽占用，加快了容器应用启动时间。



# Overlay 用户空间二层镜像 III

## 准备无 Overlay 层的底层镜像

载入 singularity 环境 module load singularity/3.5.3 后，执行

```
$ singularity shell ~/singularity/centos-basic-7.7.sif
Singularity> cat /etc/os-release
NAME="CentOS Linux"
VERSION="7 (Core)"
ID="centos"
ID_LIKE="rhel fedora"
VERSION_ID="7"
PRETTY_NAME="CentOS Linux 7 (Core)"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:centos:centos:7"
HOME_URL="https://www.centos.org/"
BUG_REPORT_URL="https://bugs.centos.org/"

CENTOS_MANTISBT_PROJECT="CentOS-7"
CENTOS_MANTISBT_PROJECT_VERSION="7"
REDHAT_SUPPORT_PRODUCT="centos"
REDHAT_SUPPORT_PRODUCT_VERSION="7"
```

# Overlay 用户空间二层镜像 IV

## 生成 Overlay 二层用户镜像

此步中，宿主系统 centos7 默认自带的 mkfs.ext3 工具版本较老，没有-d 参数支持，无法创建用户可写的镜像，需要使用高版本的 e2fsprogs 工具 (如 v1.46.2)

```
1 $ dd if=/dev/zero of=overlay.img bs=1M count=500
2 500+0 records in
3 500+0 records out
4 524288000 bytes (524 MB) copied, 0.362755 s, 1.4 GB/s
5 $ mkdir -p overlay/upper
6 mkfs.ext3 -d overlay overlay.img
```

# Overlay 用户空间二层镜像 V

## 载入用户层镜像，作为用户可写的数据空间

```
1 $ singularity shell --overlay overlay.img ~/singularity/centos-basic-7.7.sif
```

此时，普通用户已经可以在没有 **bind mount** 的目录中写入用户的文件了，而且可以将用户文件写入到原本需要 **ROOT** 权限的目录中，例如/根目录：

```
1 Singularity> echo "added by normal user" > /test.txt
2 Singularity> cat /test.txt
3 added by normal user
4 Singularity> exit
5 $ singularity shell --overlay overlay.img ~/singularity/centos-basic-7.7.sif
6 Singularity> cat /test.txt
7 added by normal user
```

如上演示，所写入的 **/test.txt** 文件被永久保持在二层镜像 **overlay.img** 中，可以在重新启动容器后继续使用。

# Overlay 用户空间二层镜像 VI

## 升级系统文件

在前一步写入操作中，用户可写入的文件不能是已存在的非当前用户的文件，也就是无法更新系统镜像中已有的系统文件，无法进一步升级系统程序。Singularity 为 ROOT 权限设置了较多限制，防止在容器中提权为 ROOT 权限，然后影响宿主系统的安全。
为解决一层系统镜像中的文件的升级问题，用户依然需要登录到一个可以拿到 ROOT 权限的 B 主机中（可以是具有 ROOT 权限的 Docker 容器），在其中，同样加载 Overlay 镜像来升级安装已在系统镜像中存在的程序。

```
 1 Singularity> yum info git | grep -E "Version|Release|Package"
 2 Installed Packages
 3 Version     : 1.8.3.1
 4 Release     : 21.el7_7
 5 Summary     : Fast Version Control System
 6 Available Packages
 7 Version     : 1.8.3.1
 8 Release     : 23.el7_8
 9 Summary     : Fast Version Control System
10 Singularity> yum install -y git
11 [......]
12 Singularity> yum info git | grep -E "Version|Release|Package"
13 Installed Packages
14 Version     : 1.8.3.1
15 Release     : 23.el7_8
16 Summary     : Fast Version Control System
```

更新安装的系统程序，也会保存在第二层的 Overlay 镜像中，用户可以将此镜像，拷贝回计算集群中，并不需要再拷贝底层系统镜像。

# Singularity exec I

## 加载 Singularity 容器环境

```
1 [wszhang@tcadmin wszhang]$ module load singularity/3.0.2
2
3 [wszhang@tcadmin wszhang]$ singularity pull shub://vsoch/hello-world
4 Progress |==================================| 100.0%
5 Done. Container is at: /home2/wszhang/vsoch-hello-world-master-latest.simg
6
7 [wszhang@tcadmin wszhang]$ singularity exec
      /home2/wszhang/vsoch-hello-world-master-latest.simg grep LTS /etc/os-release
8 VERSION="14.04.5 LTS, Trusty Tahr"
9 PRETTY_NAME="Ubuntu 14.04.5 LTS"
```

## 加载 GPU: --nv，加载系统目录: --bind /opt

```
$ singularity exec --nv --bind /opt /.../vsoch-hello-world-master-latest.simg
```

## 容器 Shell 界面

```
$ singularity shell /.../vsoch-hello-world-master-latest.simg
```

# Singularity exec II

## 如何容器化具有商业授权组件的应用？是否应该在容器中包含 Intel MKL 之类的体积巨大的库?

为了简便构造并公开分发此类应用容器，可能不方便在容器中包含这些工具链，此时，可以选择在容器内引用外部依赖库（只是与容器技术愿景有所背离）。假设，我们有程序库处于宿主系统的/opt 目录，我们需要：

- 编辑文件/etc/ld.so.conf.d/host-libs.conf，添加库路径：

  ```
  $ singularity shell -w -B /opt /home2/wszhang/singularity/sandbox/ vim /etc/ld.so.conf.d
  /host-libs.conf
  ```

  （其中 -w 参数表示将保存对容器的修改）

- 更新容器中的共享库 Cache: /etc/ld.so.cache$ singularity shell -w -B /opt /home2/wszhang/singularity/sandbox/ ldconfig

# LSF 下提交 Singularity 作业 I

```
 1 [wszhang@tcadmin wszhang]$ bsub -n 4 -q testv3 -oo %J.log mpijob singularity exec
       /home2/wszhang/vsoch-hello-world-master-latest.simg hostname
 2 Job <1268775> is submitted to queue <testv3>.
 3
 4 [wszhang@tcadmin wszhang]$ cat 1268775.log
 5 Sender: LSF System <lsfadmin@node284>
 6 Subject: Job 1268775: <mpijob singularity exec /home2/wszhang/vsoch-hello-world-master-latest.
       simg hostname> in cluster <tc4600> Done
 7
 8 Job <mpijob singularity exec /home2/wszhang/vsoch-hello-world-master-latest.simg hostname> was
       submitted from host <tcadmin> by user <wszhang> in cluster <tc4600>.
 9 Job was executed on host(s) <4*node284>, in queue <testv3>, as user <wszhang> in cluster <
       tc4600>.
10 </home/nic/wszhang> was used as the home directory.
11 </home2/wszhang> was used as the working directory.
12 Started at Results reported on
13 Your job looked like:
14
15 ------------------------------------------------------------
16 # LSBATCH: User input
17 mpijob singularity exec /home2/wszhang/vsoch-hello-world-master-latest.simg hostname
18 ------------------------------------------------------------
19
20 Successfully completed.
21
22 Resource usage summary:
23
24     CPU time :                                   0.49 sec.
```

# LSF 下提交 Singularity 作业 II

```
25     Max Memory :                             -
26     Average Memory :                         -
27     Total Requested Memory :                 -
28     Delta Memory :                           -
29     Max Swap :                               -
30     Max Processes :                          -
31     Max Threads :                            -
32     Run time :                               1 sec.
33     Turnaround time :                        2 sec.
34
35 The output (if any) follows:
36
37 node284
38 node284
39 node284
40 node284
```

# LSF 下交互式提交并测试 Singularity 环境 I

```
[wszhang@tcadmin wszhang]$ bsub -n 28 -q k80 -Is singularity shell --nv --bind /opt
      /home2/wszhang/singularity/vsoch-hello-world-master-latest.simg
Job <1358694> is submitted to queue <k80>.
<<Waiting for dispatch ...>>
<<Starting on k803>>
Singularity vsoch-hello-world-master-latest.simg:~/eclipse_project/newgit/abacus-NewGit/ABACUS
      .1.0.0/source> nvidia-smi
Thu Mar 14 20:32:55 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 396.26                 Driver Version: 396.26                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla K80           Off  | 00000000:05:00.0 Off |                    0 |
| N/A   27C    P8    27W / 149W |     11MiB / 11441MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla K80           Off  | 00000000:06:00.0 Off |                    0 |
| N/A   32C    P8    29W / 149W |     11MiB / 11441MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla K80           Off  | 00000000:85:00.0 Off |                    0 |
| N/A   30C    P8    27W / 149W |     11MiB / 11441MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla K80           Off  | 00000000:86:00.0 Off |                    0 |
| N/A   28C    P8    29W / 149W |     11MiB / 11441MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
```

# LSF 下交互式提交并测试 Singularity 环境 II

```
27 |  GPU       PID   Type   Process name                              Usage      |
28 |=============================================================================|
29 |  No running processes found                                                 |
30 +-----------------------------------------------------------------------------+
31 Singularity vsoch-hello-world-master-latest.simg:~/eclipse_project/newgit/abacus-NewGit/ABACUS
       .1.0.0/source> which
       nvidia-smi
32 /usr/bin/nvidia-smi
33 Singularity vsoch-hello-world-master-latest.simg:~/eclipse_project/newgit/abacus-NewGit/ABACUS
       .1.0.0/source> exit
```

# SLURM 下提交 Singularity 作业

在 Slurm 作业管理系统中，没有 LSF-bsub 工具所支持的直接运行可执行命令的功能，主要通过以下两种方法进行工作：

1. 将非交互式命令写入 SLURM 脚本中提交

```
1 #!/bin/sh
2 #An example for MPI job.
3 #SBATCH -J job_name
4 #SBATCH -o job-%j.log
5 #SBATCH -e job-%j.err
6 #SBATCH -N 1 -n 40  -p GPU-A100 --gpus-per-node=2 --qos=gpujoblimit --time="9-23:0:0"
7 module load singularity/3.5.3
8 mpirun singularity exec /home2/wszhang/vsoch-hello-world-master-latest.simg hostname
```

2. 使用 salloc 提交交互式的 SLURM 容器作业

```
1 salloc -N 1 -n 40 -p GPU-V100 --gpus-per-node=2 --qos=gpujoblimit --time="9-23:0:0" srun --pty
      bash
2 singularity shell --nv --bind /opt /home/wszhang/singularity/vsoch-hello-world-master-latest.
      simg
```

# 联系信息

**中国科学技术大学超算中心**

办公室科大东区新图书馆一楼东侧 126 室

电话 0551-63602248

信箱 sccadmin (a) ustc.edu.cn

主页 http://scc.ustc.edu.cn