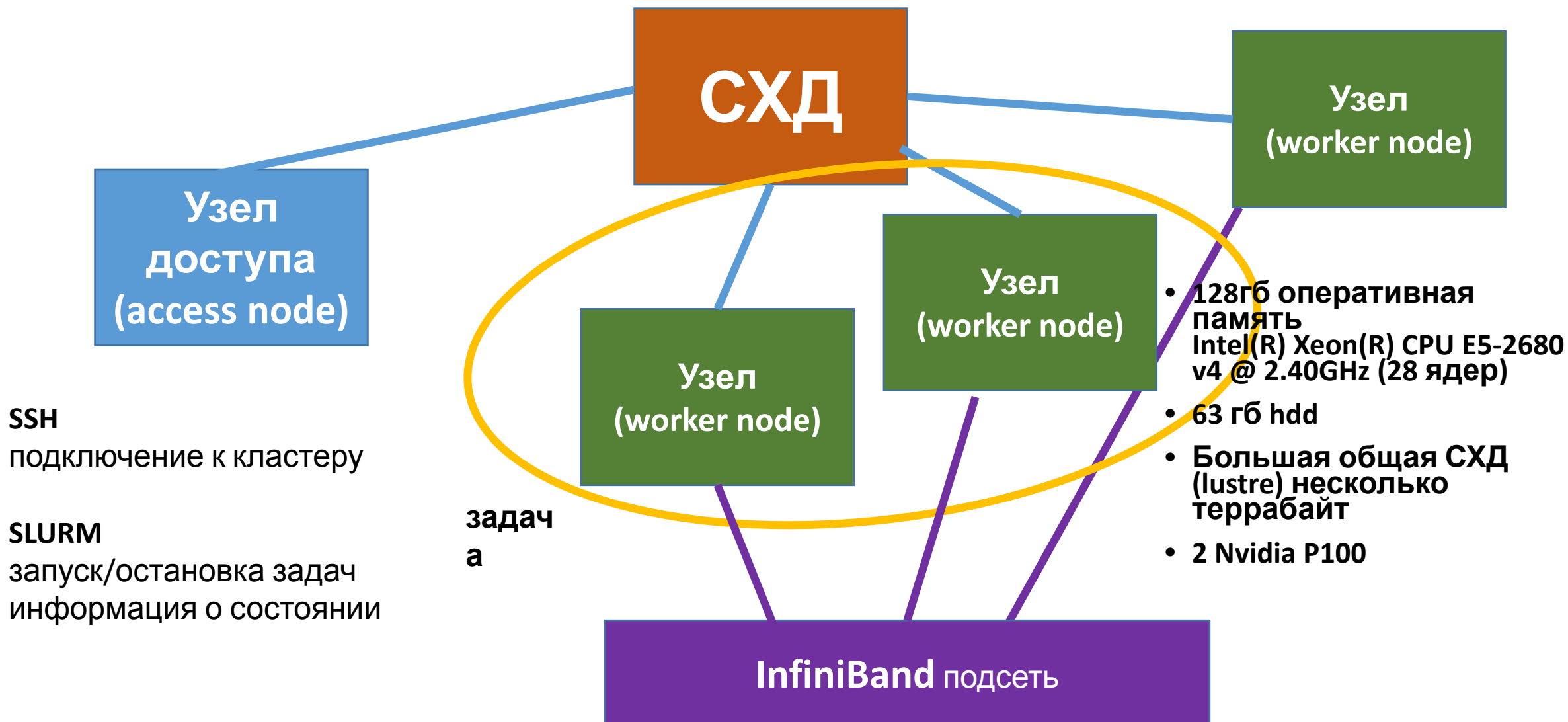


ЦСМ кластер, распределенные вычисления

Структура кластера



Общая информация

- Для работы на кластере используется SLURM

при входе на кластер необходимо подгружать его модуль

`module load slurm/17.11.5` или прописать данную команду в `.bashrc`, тогда модуль будет подгружаться автоматически

- Основные полезные команды:

- `sbatch main.sbatch` – запуск `main.sbatch` файла
- `sinfo` – просмотр состояния загруженности карточек (`alloc` – используемые карточки)
- `squeue` – просмотр запущенных задач
- `scancel <pid>` - остановка задания (доступно только для задач запущенных этим пользователем)
- `ssh n<номер узла>` - переход на узел

- Запуск задач происходит с узла доступа через `sbatch` файлы

- На узлах установлен `docker`, обучение производится в контейнерах, без доступа в интернет

- Для управления процессам используется MPI

При запуске `sbatch` файла необходимо подгружать модуль для работы `mpi`

`module load openmpi/1.10.7-gcc` или прописать в `.bashrc` (это не точно)

Запуск обучения

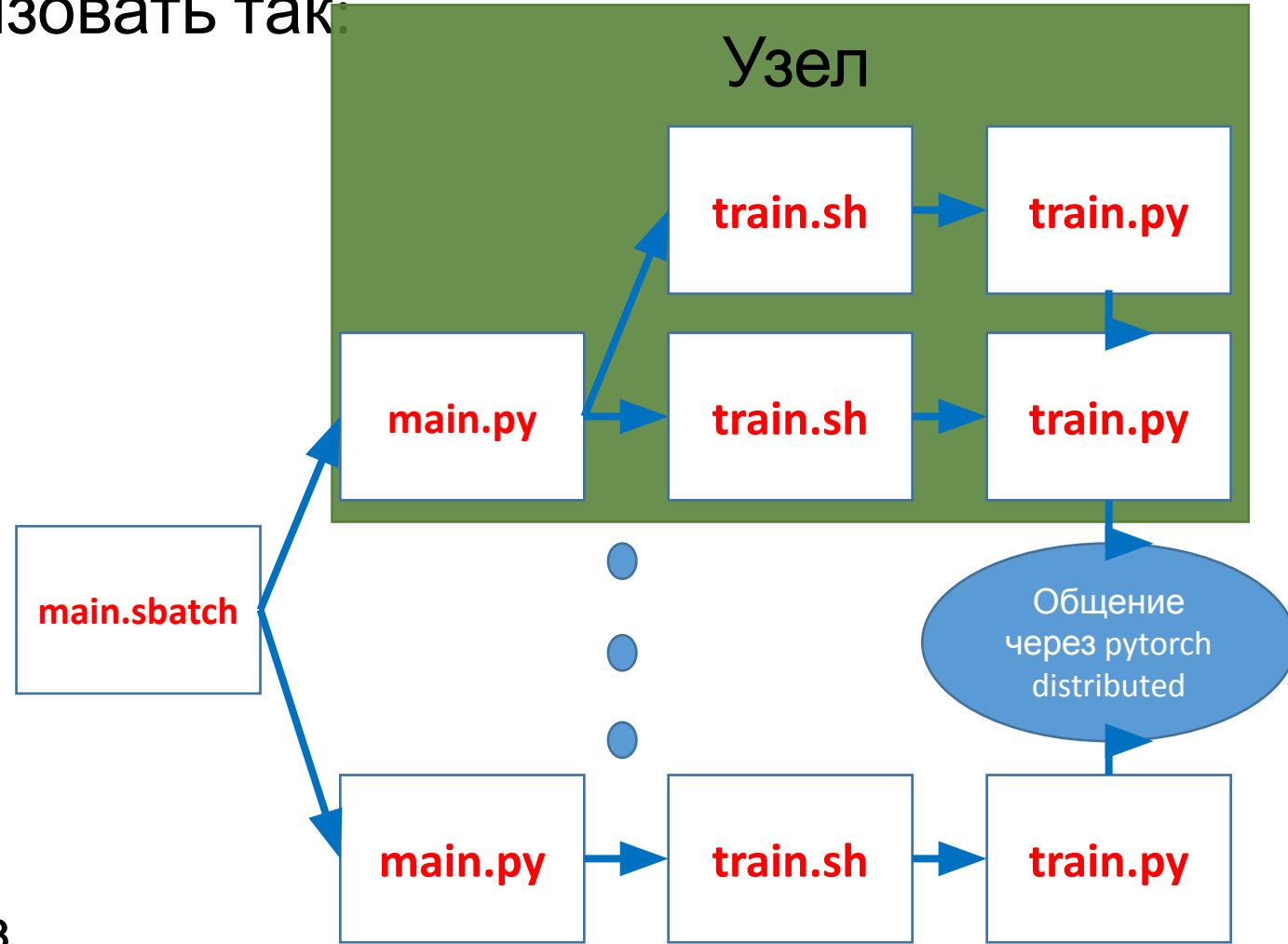
Схему обучения можно организовать так:

`main.sbatch` – загружает образы в докер и запускает файл `main.py`

`main.py` – поднимает контейнер и запускает 2 файла `train.sh` (так как 2 карточки на узле)

`train.sh` – запускает файл обучения `train.py` на необходимой карточке с необходимыми параметрами обучения

`train.py` – главный файл обучения, в котором используется `pytorch distributed` и для него приходят параметры ранга процесса и общее количество процессов. Общение между процессами происходит через файл <https://pytorch.org/docs/stable/distributed.html>.



Пример main.sbatch

```
#!/bin/bash
#SBATCH --job-name=samplenet
#SBATCH --nodes=2
#SBATCH --time=60-00:00:00
#SBATCH --partition=2xP100
#SBATCH --error=/home/iprotopopov/gosnias/
protopopov/main_experiment/log/%j_%x.log
#SBATCH --output=/home/iprotopopov/gosnias
/protopopov/main_experiment/log/%j_%x.out
#SBATCH --ntasks-per-node=1
#SBATCH --comment=eyJzb2x2.....
```

Обязательные поля:

job-name - присваиваемое имя задаче;
nodes – кол-во узлов используемые при запуске;

partition – имя нашей партиции, всегда такая;

error, output – пути сохранения ошибок и логов;

comment – тип используемого ПО, сейчас выдан для Pytorch, для другого ПО необходимо запрашивать кодировку.

module load openmpi/1.10.7-gcc

srun -l docker load -i /home/iprotopopov/gosnias/docker_images/horovod_20.tar

mpirun /usr/bin/python3

/home/iprotopopov/gosnias/protopopov/main_experiment/main.py

Примечание main.sbatch

При запуске мы всегда загружаем образ и поднимаем контейнеры, но процессы могут падать и контейнеры и образы оставаться в памяти. За их очисткой следим мы, по этому в конце или в начале sbatch файла необходимо добавить строчки:

```
srun -l bash -c 'docker stop $(docker ps -qa)'
```

```
srun -l bash -c 'docker rm $(docker ps -aq)'
```

```
srun -l docker ps
```

Пример main.py

```
import sys
import os
import subprocess

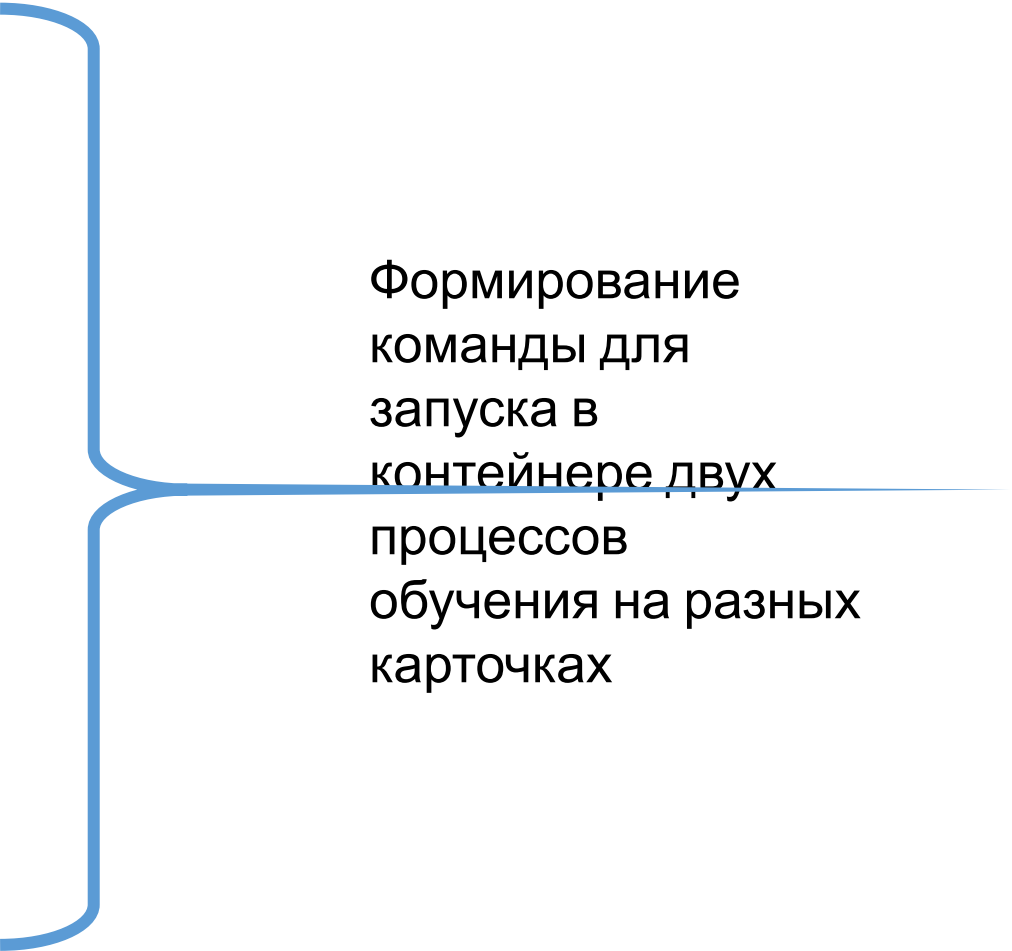
running_docker_cmd = 'nvidia-docker run \
  --privileged \
  --network=host \
  --ipc=host \
  -v /home/iprotopopov/gosnias/protopopov/main_experiment/:/code \
  -v /home/iprotopopov/gosnias/db/:/db \
  -v /root/.ssh:/root/.ssh \
  --shm-size 8Gb \
  horovod/horovod:0.20.0-tf2.3.0-torch1.6.0-mxnet1.6.0.post0-py3.7-cuda10.1 \
  bash -c "{node_cmd}"'
```



Запуск контейнера

Пример main.py

```
pytorch_node_cmd = \  
    'cd /code && CUDA_VISIBLE_DEVICES=0 NCCL_DEBUG=INFO  
    NCCL_SOCKET_IFNAME=ib0 ./train.sh {world_size} {rank0} 0 {job_id} & \  
    cd /code && CUDA_VISIBLE_DEVICES=1 NCCL_DEBUG=INFO  
    NCCL_SOCKET_IFNAME=ib0 ./train.sh {world_size} {rank1} 1 {job_id}'  
  
from mpi4py import MPI  
comm = MPI.COMM_WORLD  
rank = comm.Get_rank()  
size = comm.Get_size()  
  
node_cmd = pytorch_node_cmd.format(world_size=num_nodes, rank0=2 * rank,  
rank1=2 * rank + 1, job_id=job_id_var)  
  
running_cmd = running_docker_cmd.format(node_cmd=node_cmd)  
output = subprocess.call(['bash', '-c', 'time ' + running_cmd])
```



Формирование
команды для
запуска в
контейнере двух
процессов
обучения на разных
карточках

Пример train.py

```
dist_backend = 'nccl'  
dist_file = 'file:///code/sync_file/sync_file' + args.jobid  
dist.init_process_group(  
    backend=dist_backend,  
    init_method=dist_file,  
    rank=int(rank),  
    world_size=world_size)
```

```
net = DnCNN(1)  
net = net.cuda(0)  
net = torch.nn.parallel.DistributedDataParallel(net)
```

```
dataset_train = Dataset()  
train_sampler = torch.utils.data.distributed.DistributedSampler(dataset_train)  
loader_train = torch.utils.data.DataLoader(dataset=dataset_train,  
    num_workers=4,  
    batch_size=batch_size,  
    shuffle=(train_sampler is None),  
    pin_memory=True,  
    sampler=train_sampler)
```



Инициализация

Перевод сети

Перевод датасета

Возможно

ПОЛЕЗНОЕ

- Удобно использовать Midnight Commander: запуск командой – `mc`
- Запускаемые файлы на узле должны иметь открытый доступ `-rwxrwxr-x`, для проверки используется команда `ls -la` (для текущей папки). Открыть доступ можно с использованием команды: `chmod +x file`
- Для использования filezilla на кластере необходимо использовать ключ сгенерированный через putty для версии 2.

Опыт с предыдущей

аренды:

- используем torch.distributed в основном
- контейнер лучше взять от horovod (он корректно использует интерфейс **infiniband** без дополнительных установок)
- в контейнере обычно вы ROOT, со всеми вытекающими следствиями (в частности при монтировании папок с общего СХД)
- Данные при обучении читаются с общего СХД, каждый worker_node читает свою независимую часть (скорости хватает)
- Проверяем, что подхватился нужный сетевой интерфейс между нодами
- `scancel jobid.6 ---` убивает только бой `srun` в соответствующем `sbatch`-файле
- при нашем подходе удаление и остановка контейнеров на нашей стороне и мы за этим следим сами

Обычно в `sbatch`-файл добавляем для этого:

```
srun docker run ubuntu:latest
```

```
srun -l bash -c 'docker stop $(docker ps -qa)'
```

```
srun -l bash -c 'docker rm $(docker ps -aq)'
```