阿里云FaaS IPFS解决方案使用说明3.0

*本说明基于阿里云ECS AMD7H12 + FPGA 机型
*在当前测试机/root/lotus-sw/目录下，已配置好了环境，已根据官方代码进行了修改和编译，可以直接运行官方P1+FPGA P2，也可根据本说明修改代码运行用户P1+FPGA P2。 /root/lotus-sw/lotus目录，使用screen直接运行

```
RUST_LOG=info  ./lotus-bench sealing --storage-dir /mnt/lotus-bench --sector-size 32GiB
--skip-commit2 --skip-unseal  --no-gpu
```

（注：拷贝本文档提供的命令时注意pdf文档换行引入的空格）
（注：FaaS IPFS解决方案目前暂仅支持32GB和64GB两种扇区）

一、测试前准备

1、 系统环境

支持操作系统：Ubuntu 18.04（内核版本4.15.0）和Ubuntu20.04（内核版本5.4.0）

2、 软件环境配置

分步安装流程：

- 安装为本项目提供的Xilinx Runtime Library (XRT) 以及相应的依赖库

Ubuntu 18.04：

```
wget http://faas-ref-design.oss-cn-hangzhou.aliyuncs.com/IPFS/xrt_201910.2.2.0_18.04-xrt.deb
apt-get -f -y install ./xrt_201910.2.2.0_18.04-xrt.deb
```

Ubuntu 20.04：

```
wget http://faas-ref-design.oss-cn-hangzhou.aliyuncs.com/IPFS/xrt_201910.2.2.0_20.04-xrt.deb
apt-get -f -y install ./xrt_201910.2.2.0_20.04-xrt.deb
```

- 安装FPGA相关工具

```
apt-get install -y ipmitool
curl -o /bin/reg_rw https://faas-ref-design.oss-cn-hangzhou.aliyuncs.com/reg_rw && chmod +x /bin/reg_rw
```

- 安装lotus环境

```
apt install mesa-opencl-icd ocl-icd-opencl-dev gcc git bzr jq pkg-config curl clang build-essential hwloc libhwloc-dev -y && sudo apt upgrade -y
```

- 安装RUST环境

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
source $HOME/.cargo/env
```

- 安装GO环境

```
wget -c https://golang.google.cn/dl/go1.15.5.linux-amd64.tar.gz -O
- | sudo tar -xz -C /usr/local
```

- 下载neptune_plus.so、libposeidon.so 和xclbin文件

```
wget http://faas-ref-design.oss-cn-hangzhou.aliyuncs.com/IPFS/IPFS
.3.0/IPFS.tar.gz
tar -zxvf IPFS.tar.gz
rm -rf IPFS.tar.gz
```

3、环境变量设置

在bashrc中加入以下内容

```
export GO111MODULE=on
export GOROOT=/usr/local/go
export GOPATH=/home/gopath
export PATH=$PATH:$GOROOT/bin:$GOPATH/bin
export GOPROXY=https://goproxy.cn
export PATH="$HOME/.cargo/bin:$PATH"
export RUSTUP_DIST_SERVER=https://mirrors.ustc.edu.cn/rust-static
export RUSTUP_UPDATE_ROOT=https://mirrors.ustc.edu.cn/rust-static
/rustup

export IPFS_GATEWAY=https://proof-parameters.s3.cn-south-1.jdclou
d-oss.com/ipfs/
export RUSTFLAGS="-C target-cpu=native -g"
export FFI_BUILD_FROM_SOURCE=1
export FIL_PROOFS_USE_MULTICORE_SDR=1
#export BELLMAN_NO_GPU=1
#export BELLMAN_VERIFIER=cpu
export FIL_PROOFS_USE_FPGA_COLUMN_BUILDER=1
export FIL_PROOFS_USE_FPGA_TREE_BUILDER=1
export FAAS_IPFS_PATH=/root/IPFS
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$FAAS_IPFS_PATH/neptune_p
```

```
lus
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$FAAS_IPFS_PATH/libposeid
on
export XCLBIN_PATH=$FAAS_IPFS_PATH/xclbin
export PATH=$PATH:/root/lotus-sw/lotus

source /opt/xilinx/xrt/setup.sh
export LOTUS_FIL_PATH=/mnt
export FIL_PROOFS_MAXIMIZE_CACHING=1
export FIL_PROOFS_PARAMETER_CACHE="$LOTUS_FIL_PATH/filecoin-proof
-parameters"
export FIL_PROOFS_PARENT_CACHE="$LOTUS_FIL_PATH/filecoin-parents"
export TMPDIR="$LOTUS_FIL_PATH/lotustmp"
export LOTUS_PATH="$LOTUS_FIL_PATH/lotus-data"
export LOTUS_MINER_PATH="$LOTUS_FIL_PATH/lotus-miner"
export LOTUS_WORKER_PATH="$LOTUS_FIL_PATH/lotus-worker"
export LOTUS_STORAGE_PTAH="$LOTUS_FIL_PATH/lotus-storage"
```

修改完成后执行

```
source .bashrc
```

4、下载官方源代码

新建工作目录为lotus-sw，下载lotus，rust-fil-proofs，filecoin-proofs-api

```
mkdir $LOTUS_FIL_PATH/filecoin-proof-parameters
mkdir $LOTUS_FIL_PATH/filecoin-parents
mkdir $LOTUS_FIL_PATH/lotustmp
mkdir $LOTUS_FIL_PATH/lotus-data
mkdir $LOTUS_FIL_PATH/lotus-miner
mkdir $LOTUS_FIL_PATH/lotus-worker
mkdir $LOTUS_FIL_PATH/lotus-storage
mkdir lotus-sw
cd lotus-sw
git clone https://github.com/filecoin-project/lotus.git
git clone -b v5.4.1 https://github.com/filecoin-project/rust-file
coin-proofs-api
```

```
git clone -b storage-proofs-v5.4.0 https://github.com/filecoin-project/rust-fil-proofs
cd lotus
```

二、配置硬件

在/root/IPFS 目录下，执行

```
./reload.sh
```

等待reload完成后，执行

```
./fpgainit
```

完成对硬件对初始化。
配置硬件的操作只需一次，但机器发生重启后需要重新配置

三、代码准备

1、修改源代码依赖

修改lotus的配置文件lotus/extern/filecoin-ffi/rust/Cargo.toml 中的[dependencies.filecoin-proofs-api]，使其依赖于本地的filecoin-proofs-api

```
[dependencies.filecoin-proofs-api]
path = "../../../../rust-filecoin-proofs-api"
package = "filecoin-proofs-api"
version = "5.4.1"
default-features = false
```

修改rust-filecoin-proofs-api的配置文件rust-filecoin-proofs-api/Cargo.toml中的[dependencies.filecoin-proofs-v1]，使其依赖于本地的rust-fil-proofs

```
[dependencies.filecoin-proofs-v1]
path = "../rust-fil-proofs/filecoin-proofs"
version = "5.4.0"
features = ["gpu"]
```

```
default-features = false
package = "filecoin-proofs"
```

或者

```
filecoin-proofs-v1 = {path = "../rust-fil-proofs/filecoin-proofs", package = "filecoin-proofs", version = "5.4.0", default-features = false, features = ["gpu"] }
```

2、调用neptune_plus_ffi.so及libposeidon.so

在rust-fil-proofs/storage-proofs/porep/src和rust-fil-proofs/storage-proofs/src目录下新建build.rs

```
cat <<EOF> ./rust-fil-proofs/storage-proofs/porep/src/build.rs
use std::{env, path::PathBuf};
extern crate dunce;
fn main() {

    let ipfs_fpga_dir_neptune = PathBuf::from(env::var("FAAS_IPFS_PATH").expect("FAAS_IPFS_PATH env var is not defined"));
    let neptune_dir = dunce::canonicalize(ipfs_fpga_dir_neptune.join("neptune_plus")).unwrap();
    let poseidon_dir = dunce::canonicalize(ipfs_fpga_dir_neptune.join("libposeidon")).unwrap();

    println!("cargo:rustc-link-search=native={}", env::join_paths(&[neptune_dir]).unwrap().to_str().unwrap());
    println!("cargo:rustc-link-lib=dylib=neptune_plus_ffi");
    println!("cargo:rustc-link-search=native={}", env::join_paths(&[poseidon_dir]).unwrap().to_str().unwrap());
    println!("cargo:rustc-link-lib=dylib=poseidon_hash");
    println!("cargo:rustc-link-search=native=/opt/xilinx/xrt/lib");
    println!("cargo:rustc-link-lib=dylib=xrt_coreutil");
    println!("cargo:rustc-link-lib=dylib=xilinxopencl");
    println!("cargo:rustc-link-lib=dylib=xrt_core");

}
EOF
```

```
cat <<EOF> ./rust-fil-proofs/storage-proofs/src/build.rs
use std::{env, path::PathBuf};
extern crate dunce;
fn main() {

    let ipfs_fpga_dir_neptune = PathBuf::from(env::var("FAAS_IPFS_PATH").expect("FAAS_IPFS_PATH env var is not defined"));
    let neptune_dir = dunce::canonicalize(ipfs_fpga_dir_neptune.join("neptune_plus")).unwrap();
    let poseidon_dir = dunce::canonicalize(ipfs_fpga_dir_neptune.join("libposeidon")).unwrap();

    println!("cargo:rustc-link-search=native={}", env::join_paths(&[neptune_dir]).unwrap().to_str().unwrap());
    println!("cargo:rustc-link-lib=dylib=neptune_plus_ffi");
    println!("cargo:rustc-link-search=native={}", env::join_paths(&[poseidon_dir]).unwrap().to_str().unwrap());
    println!("cargo:rustc-link-lib=dylib=poseidon_hash");
    println!("cargo:rustc-link-search=native=/opt/xilinx/xrt/lib");
    println!("cargo:rustc-link-lib=dylib=xrt_coreutil");
    println!("cargo:rustc-link-lib=dylib=xilinxopencl");
    println!("cargo:rustc-link-lib=dylib=xrt_core");

}
EOF
```

在rust-fil-proofs/storage-proofs/porep/Cargo.toml和rust-fil-proofs/storage-proofs/Cargo.toml 两个配置文件中加入build.rs依赖

```
#在[package]之后加入以下代码
build = "src/build.rs"
[build-dependencies]
dunce = "0.1.1"
```

修改proof.rs
proof.rs路径：rust-fil-proofs/storage-proofs/porep/src/stacked/vanilla/proof.rs
文件地址：http://faas-ref-design.oss-cn-hangzhou.aliyuncs.com/IPFS/IPFS.3.0/proof.rs
注意：在3.0版本的proof.rs文件中，相对官方源码，增加了

（1）transform_and_replicate_layers_inner_fpga函数

（2）外部函数generate_tree_r_last_fpga和generate_tree_c_with_fpga

需要改动的地方有两个：

(1)transform_and_replicate_layers：需要加入调用transform_and_replicate_layers_inner_fpga

(2)replicate_phase2：需要加入调用transform_and_replicate_layers_inner_fpga

3、加入FPGA相关环境变量

修改rust-fil-proofs/storage-proofs/core/src/settings.rs，加入FPGA相关环境变量

```
pub struct Settings {
    pub verify_cache: bool,
    pub verify_production_params: bool,
    pub use_gpu_column_builder: bool,
    pub max_gpu_column_batch_size: u32,
    pub column_write_batch_size: u32,
    pub use_gpu_tree_builder: bool,
    pub max_gpu_tree_batch_size: u32,

    pub use_fpga_column_builder: bool,
    pub use_fpga_tree_builder: bool,
    pub fpga_column_max_n: u32,
    pub fpga_tree_max_n: u32,

    pub rows_to_discard: u32,
    pub sdr_parents_cache_size: u32,
    pub window_post_synthesis_num_cpus: u32,
    pub parameter_cache: String,
    pub parent_cache: String,
    pub use_multicore_sdr: bool,
    pub multicore_sdr_producers: usize,
    pub multicore_sdr_producer_stride: u64,
```

```rust
    pub multicore_sdr_lookahead: usize,
}


impl Default for Settings {
    fn default() -> Self {
        Settings {
            verify_cache: false,
            verify_production_params: false,
            use_gpu_column_builder: false,
            max_gpu_column_batch_size: 400_000,
            column_write_batch_size: 262_144,
            use_gpu_tree_builder: false,
            max_gpu_tree_batch_size: 700_000,

            use_fpga_column_builder: true,
            fpga_column_max_n: 16384,
            use_fpga_tree_builder: true,
            fpga_tree_max_n: 16384,

            rows_to_discard: 2,
            sdr_parents_cache_size: 2_048,
            window_post_synthesis_num_cpus: num_cpus::get() as u32,
            // `parameter_cache` does not use the cache() mechanism because it is now used
            // for durable, canonical Groth parameters and verifying keys.
            // The name is retained for backwards compatibility.
            parameter_cache: "/var/tmp/filecoin-proof-parameters/".to_string(),
            parent_cache: cache("filecoin-parents"),
            use_multicore_sdr: false,
            multicore_sdr_producers: 3,
            multicore_sdr_producer_stride: 128,
            multicore_sdr_lookahead: 800,
        }
    }
}
```

4、在GO中调用相关依赖库

在extern/filecoin-ffi/bls.go 和 extern/filecoin-ffi/generated/generated.go 的LDFLAGS中分别添加以下依赖库

```
-L/opt/xilinx/xrt/lib -lxilinxopencl -lxrt_coreutil -lxrt_core -L/root/IPFS/libposeidon/ -lposeidon_hash -L/root/IPFS/neptune_plus/ -lneptune_plus_ffi
```

四、运行lotus-bench

为了防止ssh超时导致程序异常中止，需进入screen模式

```
screen -L
```

在screen里运行lotus-bench

```
cd lotus
make
make lotus-bench
RUST_LOG=info ./lotus-bench sealing --storage-dir /mnt/lotus-bench --sector-size 32GiB --num-sectors 28  --parallel 28 --skip-commit2 --skip-unseal  --no-gpu
```

然后等待程序运行结果即可

screen断开连接后，重新进入screen方法：

```
screen -ls    #显示目前所有的screen作业。
screen -x <screen作业名称>
```

screen相关命令大全：https://www.cnblogs.com/mchina/archive/2013/01/30/2880680.html

重要的事情说三遍：

测试请用screen!

测试请用screen!

测试请用screen!