# IO_URING

Dinesh Reddy

April 2023

## 1 Abstract

With the slowing of Mores Law, hardware trends moved towards parallelism. This exposes the need for asynchronous system calls. IO_URING is a simple to use interface introduced in the recent Linux Kernels is being rapidly adapted for various IO – network and block. We show the efficacy of IO_URING in file-copy utilities by improving the performance by roughly an order of magnitude for copying medium and large files.

## 2 IO_URING

Hardware is moving towards parallelism.

Linux aio is complicated.

IO_URING uses shared memory between user and kernel space. Hence, we have advantages like "zero-copy", batching...

## 3 Experimetal Setup

### 3.1 System Specifications

I am using a Ubuntu 20 machine.

Though the selection of hardware doesn't clearly take advantage of performance benefits associated with parallel hardware, we still get some advantage in terms of minimizing system calls and zero-copying.

We performaned every experiment 5 times.

For the experiments involving IO_URING benefits, we used 4K blocks.

To make sure that block cache doesn't impact the performance, we mount a virtual disk and access the files from the virtual disk.

Listing 1: Create a Virtual disk

```
sudo mkdir ./virtualdisk
sudo dd if=/dev/zero of=./virtualdisk/vdisk.img bs=1G
    count=10
sudo losetup /dev/loop40 ./virtualdisk/vdisk.img
sudo mkfs.ext4 /dev/loop40
```

Listing 2: mount and unmount when using

```
sudo mount /dev/loop40 ./virtualdisk
sudo dd if=/dev/urandom of=./virtualdisk/largefile bs=1
    G count=1
# access the file
# ...
sudo umount virtualdisk
sudo losetup -d /dev/loop40
```

### 3.2 Understanding metrics - Latency

As shown in the pseudocode, we measure latency by the time it takes for the read to finish.

This would capture the time it takes

Listing 3: Psudocode

```
int latency_main(int argc, char **argv) {
    fd = open("virtualdisk/largefile", O_RDONLY | O_DIRECT); // Open the file

    int block_num = opts.opt_random ? rand() % NUM_BLOCKS : 0;
    if(opts.opt_synchronous) ret = lseek(fd, block_num*BLOCK_SIZE, SEEK_SET);
```

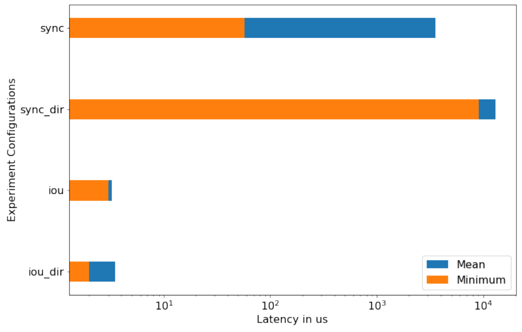

Figure 1: 4K block random read latency comparision.

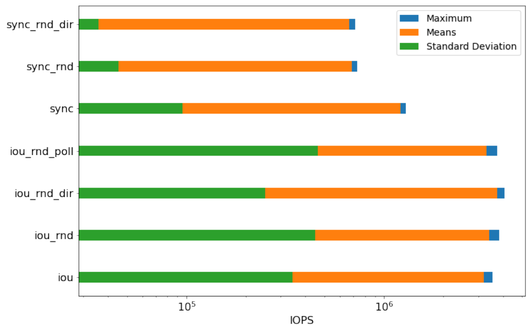

Figure 2: 4K block random read IOPS comparision.

```
    else io_uring_setup_and_init(block_num);

    gettimeofday(&submit_time, NULL); // after submitting all the request
    if(opts.opt_synchronous) ret = read(fd, read_buffer, BLOCK_SIZE);
    else{
        io_uring_submit(&ring);
        ret = io_uring_wait_cqe(&ring, &cqe);
    }
    gettimeofday(&end_time, NULL); // End the timer
}
```

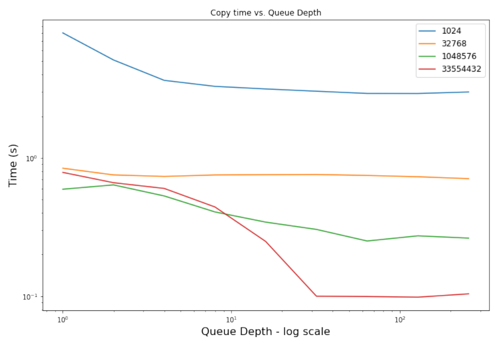

Figure 3: 4K block random read latency comparision.

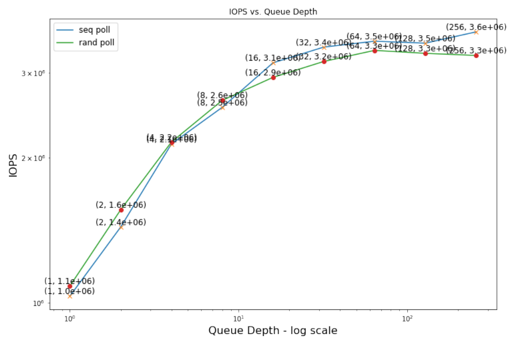

Figure 4: 4K block random read latency comparision.

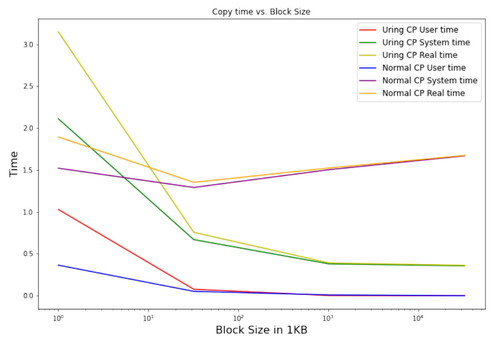

Figure 5: 4K block random read latency comparision.

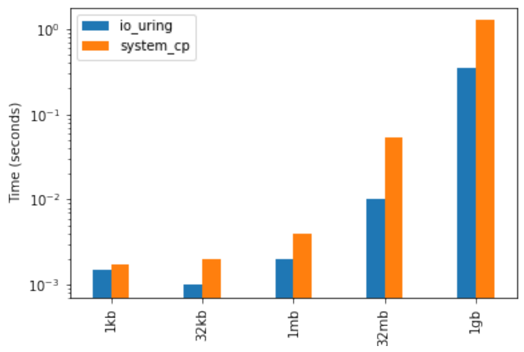

Figure 6: 4K block random read latency comparision.

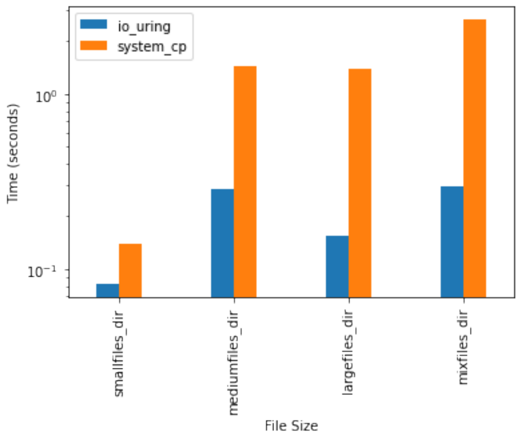

Figure 7: 4K block random read latency comparision.

### 3.3 Understanding metrics - IOPS

### 3.4 Types of experimets

## 4 Performance benefits of IO_URING

### 4.1 Impact of Queue Depth

### 4.2 Impact of Block Size

## 5 Evaluation

### 5.1 cp vs uring_cp

### 5.2 cp -r vs uring_cp

## 6 Conclusion

IO_URING is complicated to use. It is challenging not only to implement the functionality but also to harness the performance with correct configurations and settings.

### 6.1 Referances

Please find the code at my git repo git@github.com:chintadinesh/adv_os-project.git.