High Throughput HEP Data Processing at HPC

Carlos Cocha

Project Mentor: Viktor Khristenko, Maria Girone (CERN)

Project Description:

- High Energy Physics (HEP) community employed High Throughput Computing (HTC) type of facilities for LHC data processing and physics analyses.

- With recent convergence of AI and HPC, a single but modular and flexible type of facility could replace single-purpose based environments.

- The goal of this project is to take several types of workloads:

  - Compute bound LHC event reconstruction and I/O driven physics analyses (ML or DL).
  - Evaluate their effectiveness at HPC scale.
  - Scale very different workloads, but still both data driven, and understand the limitations of the existing model
  - Report peculiarities of HPC systems under heavy dataflow load.

Project Description: Pin point!

- Take I/O driven applications/mock-ups and scale them out on HPC system (CSCS Grand Tave).
- Obtain and visualize performance metrics (e.g. bandwidth utilization).
- Adapt existing HEP benchmarking suite towards I/O related benchmarking.

Project Plan: Week 1-2: Start up

- Get up and running @CSCS Grand Tave:
  https://user.cscs.ch/access/running/grand_tave/
- Checking container options
- Shifter, Sarus
- Can not natively bring LHC/CMS applications
- Mock up tests with fio:
  https://github.com/axboe/fio

Week 1-2: Follow up on fio

- **Task 1:** Learn to write fio config files: https://github.com/axboe/fio
  - FIO is a open-source synthetic benchmark tool initially developed by Jens Axboe.
  - FIO can generate various IO type workloads be it sequential reads or random reads, etc.., based on the options provided by the user.

---- seqred.fio ----

```
[seq-read]
iodepth=256
ioengine=libaio
bs=4K

runtime=300
time_based=1

numjobs= 4
size=1g
rw=read

directory=${FIO_DIRECTORY}
```

-> Job name
-> will submit two sequential IO requests at a time.
-> Defines how the job issues I/O to the file. Libaio for Linux native I/O.
-> The block size in bytes used for I/O units, e.g., 4K for reads, writes and trims.

-> terminate processing after the specified period of time.
-> fio will run for the duration of the runtime specified even if the file(s) are completely read or written

-> create the specified number of clones of this job spawned as an independent thread or process.

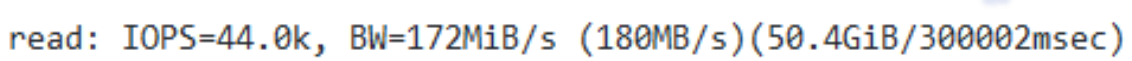
-> Type of I/O pattern. Accepted values are: read; write; trim; randread; randwrite; rw,readwrite, randrw.

-> storage volume path to be tested, e.g., /scratch/snx2000/ccocha

```
$ fio seqread.fio
```

Week 1-2: Follow up on fio

- **Task 1:** fio output




```
test: (g=0): rw=read, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=256
...
fio-3.27
Starting 4 processes

test: (groupid=0, jobs=4): err= 0: pid=255355: Fri Jul 16 03:45:10 2021
  read: IOPS=44.0k, BW=172MiB/s (180MB/s)(50.4GiB/300002msec)
    slat (usec): min=10, max=103305, avg=79.21, stdev=586.17
    clat (usec): min=10, max=1054.4k, avg=23178.42, stdev=82845.05
     lat (usec): min=22, max=1054.4k, avg=23258.48, stdev=83127.23
    clat percentiles (msec):
     |  1.00th=[    5],  5.00th=[    5], 10.00th=[    5], 20.00th=[    5],
     | 30.00th=[    5], 40.00th=[    6], 50.00th=[    6], 60.00th=[    6],
     | 70.00th=[    6], 80.00th=[    6], 90.00th=[   17], 95.00th=[   27],
     | 99.00th=[  439], 99.50th=[  489], 99.90th=[  852], 99.95th=[  944],
     | 99.99th=[ 1011]
   bw (  KiB/s): min= 1336, max=657552, per=100.00%, avg=181887.21, stdev=42912.42, samples=2363
   iops        : min=  334, max=164388, avg=45471.82, stdev=10728.11, samples=2363
  lat (usec)   : 20=0.01%, 50=0.01%, 100=0.01%, 250=0.01%, 500=0.01%
  lat (usec)   : 750=0.01%, 1000=0.01%
  lat (msec)   : 2=0.01%, 4=0.01%, 10=85.58%, 20=4.58%, 50=5.68%
  lat (msec)   : 100=0.01%, 250=0.23%, 500=3.48%, 750=0.32%, 1000=0.10%
  lat (msec)   : 2000=0.01%
  cpu          : usr=9.79%, sys=29.32%, ctx=536446, majf=0, minf=1180
  IO depths    : 1=0.1%, 2=0.1%, 4=0.1%, 8=0.1%, 16=0.1%, 32=0.1%, >=64=100.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.1%
     issued rwts: total=13199458,0,0,0 short=0,0,0,0 dropped=0,0,0,0
     latency   : target=0, window=0, percentile=100.00%, depth=256

Run status group 0 (all jobs):
   READ: bw=172MiB/s (180MB/s), 172MiB/s-172MiB/s (180MB/s-180MB/s), io=50.4GiB (54.1GB), run=300002-300002msec
```

Week 1-2: Follow up on fio

- **Task 2**: Script things up to run fio tests against shared storage
  - Use slurm to allocate/submit for 1,2,4,8,16,32, … nodes
  - Collect logs from fio for each node.

```
#!/bin/bash
#SBATCH --partition=normal
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --wait-all-nodes=1
#SBATCH --job-name=cmssw_m2n_iobench
#SBATCH --time=20:00
#SBATCH --output=/users/ccocha/FIO/fio-tests/seqread/4_node/%j.log
#SBATCH --error=/users/ccocha/FIO/fio-tests/seqread/4_node/%j.err

# general settings
fio_njob=2
fio_directory=/scratch/snx2000/ccocha/4_node
LOGSDIR_TOP=/users/ccocha/FIO/fio-tests/seqread/4_node

# based on the above
LOGSDIR_JOB="$LOGSDIR_TOP/job_${SLURM_JOB_ID}_fiojob_$fio_njob"

# create a common dir for this job for the logs/results
[ -d "$LOGSDIR_JOB" ] && rm -rf "$LOGSDIR_JOB"
mkdir $LOGSDIR_JOB

# run the stuff on each node for each fio_njob
srun fio_seqread.sh $LOGSDIR_JOB $fio_njob $fio_directory
```

Week 1-2: Follow up on fio

- **Task 2:** Script things up to run fio tests against shared storage
  - Use slurm to allocate/submit for 1,2,4,8,16,32, …
  - Collect logs from fio for each node.

```
…
# run fio for instance 0
if [ $INSTANCE -eq 0 ]
then
   fio /users/ccocha/FIO/fio-tests/seqread/seqread.fio >> output.log &
   # loop and collect network usage
   # break out when the above process is finished
   PID=$!
   while true;
   do
      [ -n "${PID}" -a -d "/proc/${PID}" ] || break
      timestamp=$(date +%s )
      #bytes=`cat /sys/class/net/ib0/statistics/rx_bytes`
      bytes=( $timestamp $(ifconfig | grep "UP\|RX packets"|awk '{print $5}') )
      printf " ${bytes[*]} " >> netlogs
      unset bytes
      sleep 1
   done
   # save the net logs
   mv netlogs $LOGSDIR_FIO_NJOB/
else
   # for all the other instances
   fio /users/ccocha/FIO/fio-tests/seqread/seqread.fio >> output.log
fi
# move the logs if exist
mv output.log $LOGSDIR_FIO_NJOB/
```

Week 1-2: Follow up on fio

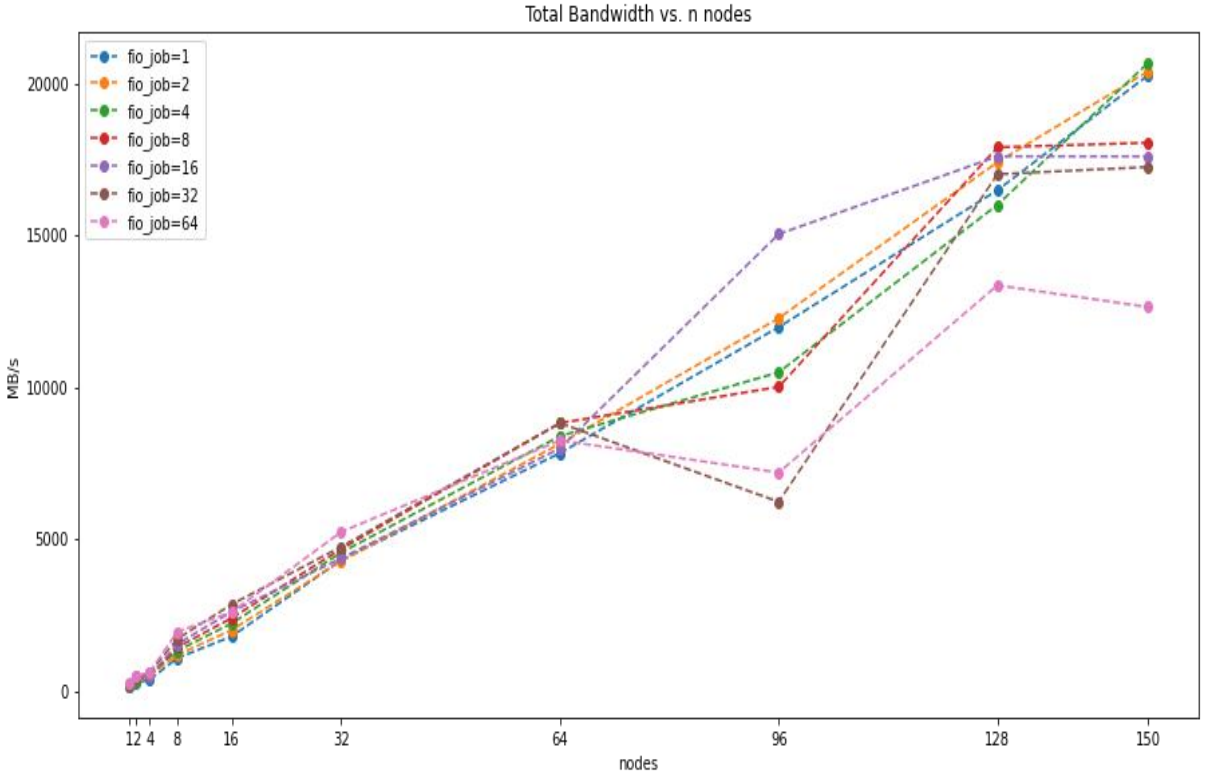
- **Task 3:** Analyze results of Task 2
  - Write up analyzer scripts to process logs from Task 2.
  - Create a plot, e.g. with python using matplotlib, showing ”total bandwidth” vs “n nodes”.

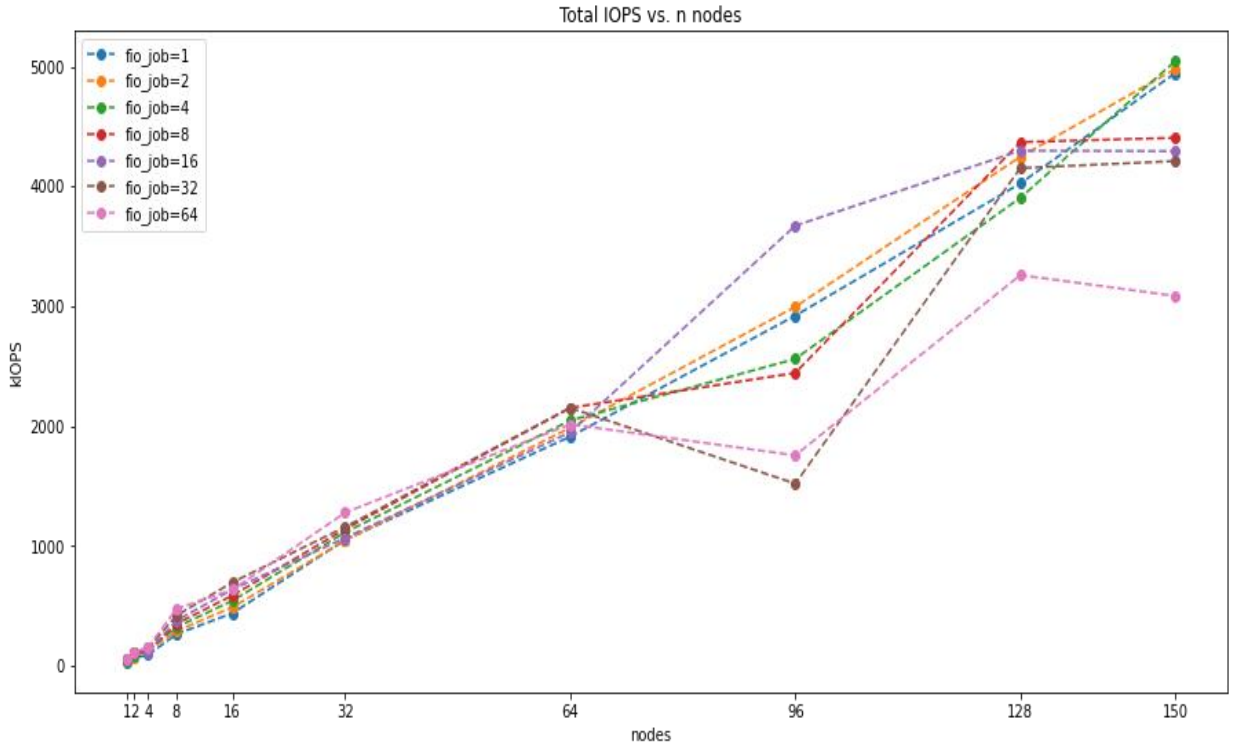
```
def bw_ios(filename, option):
    '''
    Function to read BW or IOPS from a log file
    '''
    fopen = open(filename, mode='r+' )
    fread = fopen.readlines()
    val=0
    for line in fread:
        if "IOPS" in line:
            raw = [s for s in line.split() if option in s][0]
            value=''
            for k in raw[len(option)+1:]:
                if k!='k'and k!='K' and k!='M' and k!='G' and k!='B' and k!=',': value+=k
                else: break

            #Map the correct units
            unit = raw[len(option)+len(value)+1]
            if unit== 'k'or unit== 'K': factor=1e3
            elif unit== 'M': factor=1e6
            elif unit== 'G': factor=1e9
            else: factor = 1
            value=float(value)

    return value*factor
```

Week 1-2: Follow up on fio

- **Task 3:** Analyze results of Task 2
  - Write up analyzer scripts to process logs from Task 2.
  - Create a plot, e.g. with python using matplotlib, showing ”total bandwidth” vs “n nodes”

SEQUENTIAL READ:

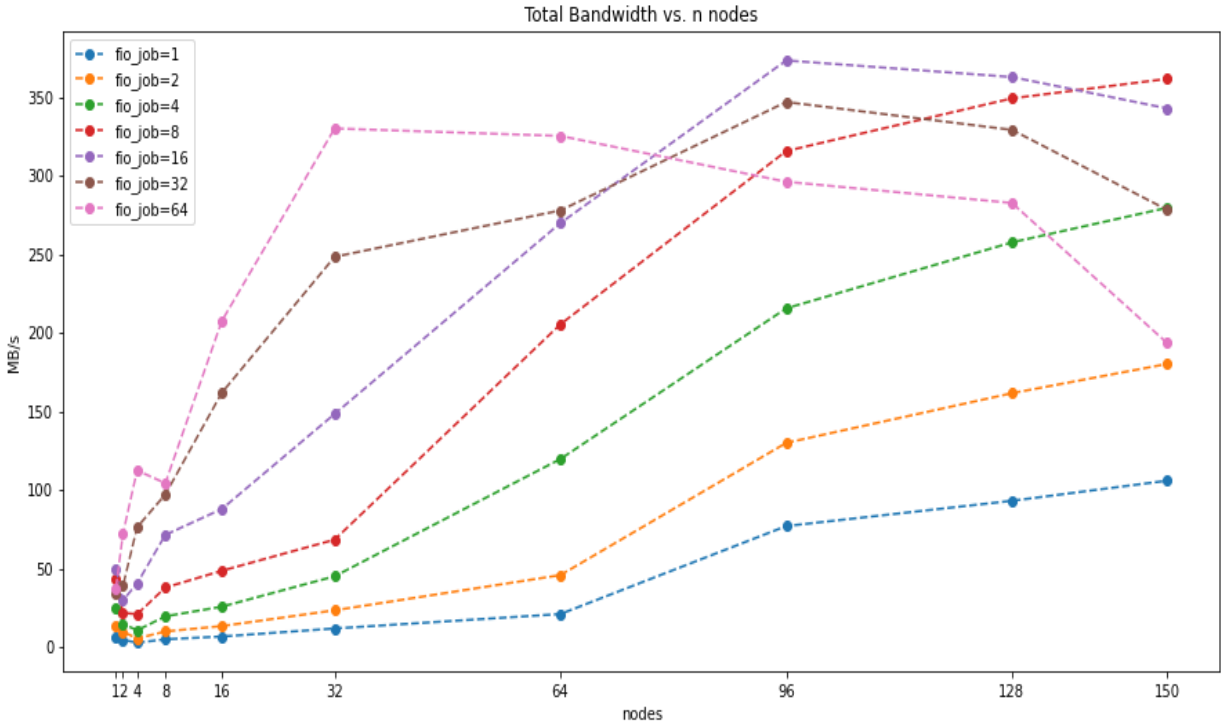


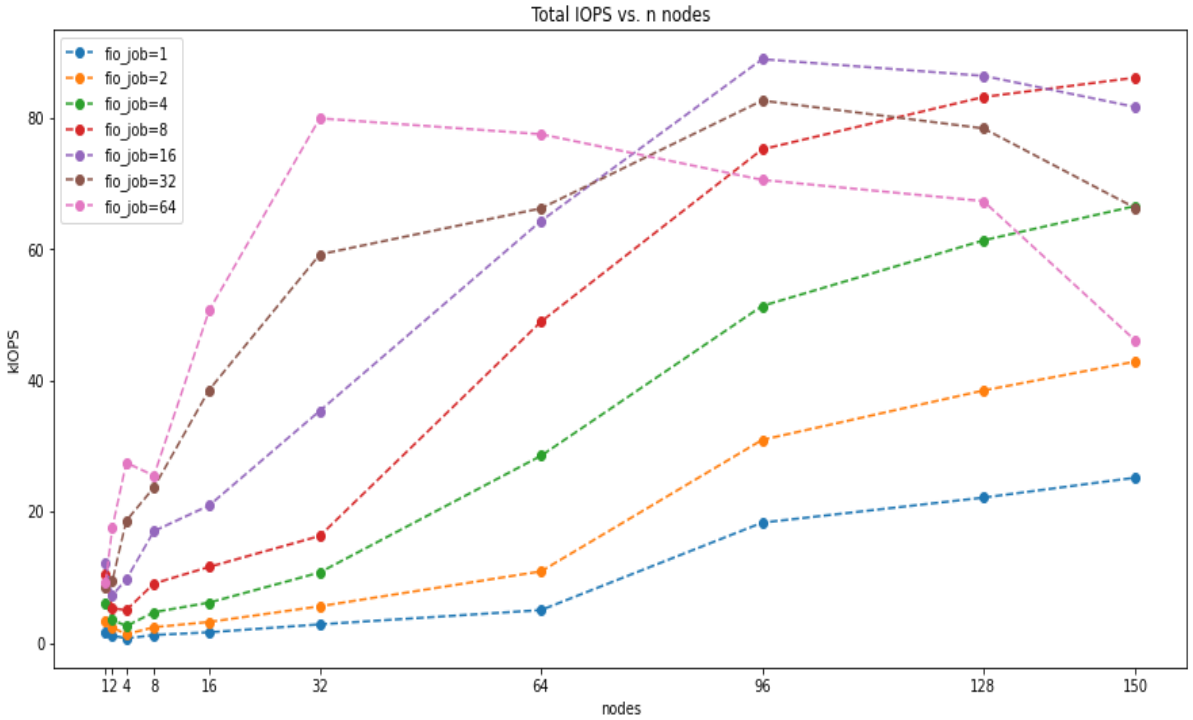


Week 1-2: Follow up on fio

- **Task 3:** Analyze results of Task 2
  - Write up analyzer scripts to process logs from Task 2.
  - Create a plot, e.g. with python using matplotlib, showing ”total bandwidth” vs “n nodes”

RANDOM READ:





Thanks for your attention !

Week 1-2: Follow up on fio

- **Task 4**: Add simple monitoring
  - Linux exposes various counters thru “files”, e.g. /sys/class/net/ib0/statistics/rx_bytes
  - Ib0 – is the network interface, the rest should be quite standard
  - Rx_bytes – number of bytes received at the network card level
  - Use “rx_bytes” to record this every 1s during the fio execution (on each node of course)
  - Create a plot/graph, e.g. with python using matplotlib, showing once again “total bandwidth” vs “n nodes”
  - In principle if u compare this method with what fio tells us, they should not be very far apart…

- **Task 5 Extra**: Containerize fio testing
  - Now, instead of running fio natively, we need to produce docker images and run them (with Sarus @CSCS)