elbencho

The new Network Testing Mode in elbencho v3.0

Analyzing Parallel IO BoF @ SC23

Sven Breuner
sven.breuner@gmail.com

A Storage Benchmark Tool for AI, HPC et al

Background

Who?

Sven & contributors:

AWS, Excelero, Penguin Computing, VAST Data, Zettar, …

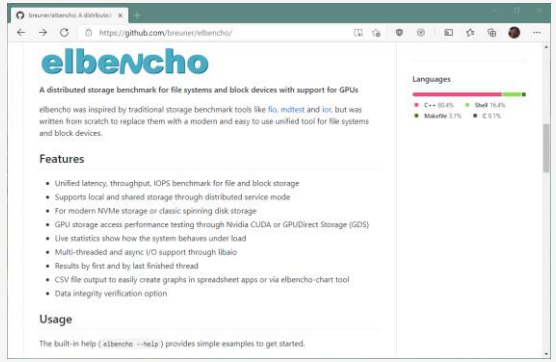
Why?

- **Find best possible performance to optimize application access**
  - Performance not obvious from HW Specs with Flash drives
- **One tool for all storage tests**
  - In contrast to specialized tools like mdtest or IOR
- **Live statistics**
  - The fio feature I appreciated the most
- **GPU support**
- **Shared file systems**
  - But without need for MPI

Where?

GitHub

https://github.com/breuner/elbencho



How to use elbencho?

Lots of small files
Benchmark path is a directory

```
$ elbencho /mnt/smb/mydir
  -r -t 3 -n 10 -N 30 -s 128k -b 128k --direct

OPERATION RESULT TYPE         FIRST DONE  END RESULT
========= ================    ==========  ==========
READ      Elapsed seconds   :         12          14
          Files/s           :        130         127
          Throughput MiB/s  :         16          15
          Total files       :       1631        1800
          Total MiB         :        203         225
```

- **-r / -w** Read or write files
- **-t** Number of threads
- **-n** Number of directories per thread
- **-N** Number of files per directory
- **-s** File size

Random read IOPS
Benchmark path is a file

```
$ elbencho -w -s 50g /mnt/smb/myfile{1..2}

$ elbencho -r -t 4 -b 4k --lat --direct --rand --timelimit 10
  /mnt/smb/myfile{1..2}

OPERATION RESULT TYPE         FIRST DONE  END RESULT
========= ================    ==========  ==========
READ      Elapsed seconds   :         10          10
          IOPS              :     130214      130214
          Throughput MiB/s  :        509         509
          IO latency        : [ min=1ms avg=35ms max=548ms ]
```

Time limit (in seconds) can be used to avoid long wait times that won't change the IOPS result

What can elbencho show you?

All the metrics of interest ☺

from a single client or coordinated across multiple clients

```
$ elbencho /mnt/smb --hosts devel1,devel2
  -r -t 3 -n 10 -N 30 -s 128k -b 128k --lat --direct

OPERATION RESULT TYPE         FIRST DONE  END RESULT
========= ================    ==========  ==========
READ      Elapsed seconds   :         12          14
          Files/s           :        130         127
          IOPS              :        130         127
          Throughput MiB/s  :         16          15
          Total files       :       1631        1800
          Total MiB         :        203         225
          Files latency ms  : [ min=2 avg=44 max=565 ]
          IO latency ms     : [ min=1 avg=35 max=548 ]
```

Hint: Distributed runs are easy (without MPI)

```
devel1:~$  elbencho --service
devel2:~$  elbencho --service
master1:~$  elbencho --hosts devel1,devel2 ...
# Or alternatively:
master1:~$  elbencho --hostsfile myhosts.txt ...
```

Live Statistics

```
Phase: READ  CPU: 1%  Active: 2  Elapsed: 10s
 Rank    %  DoneMiB   MiB/s   IOPS   Files  Files/s  Act CPU Service
Total   68       153      14    117    1224      117    6   1
    0   68        77       7     61     616       61    3   1 devel1
    1   67        76       7     56     608       56    3   1 devel2
```

elbencho for GPUs

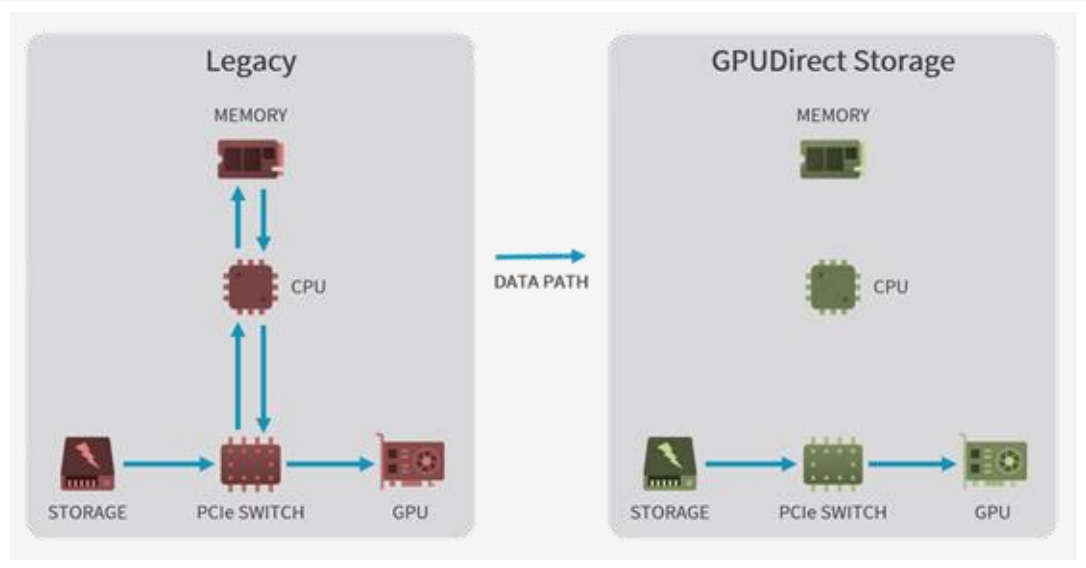
GPU data transfer via CUDA

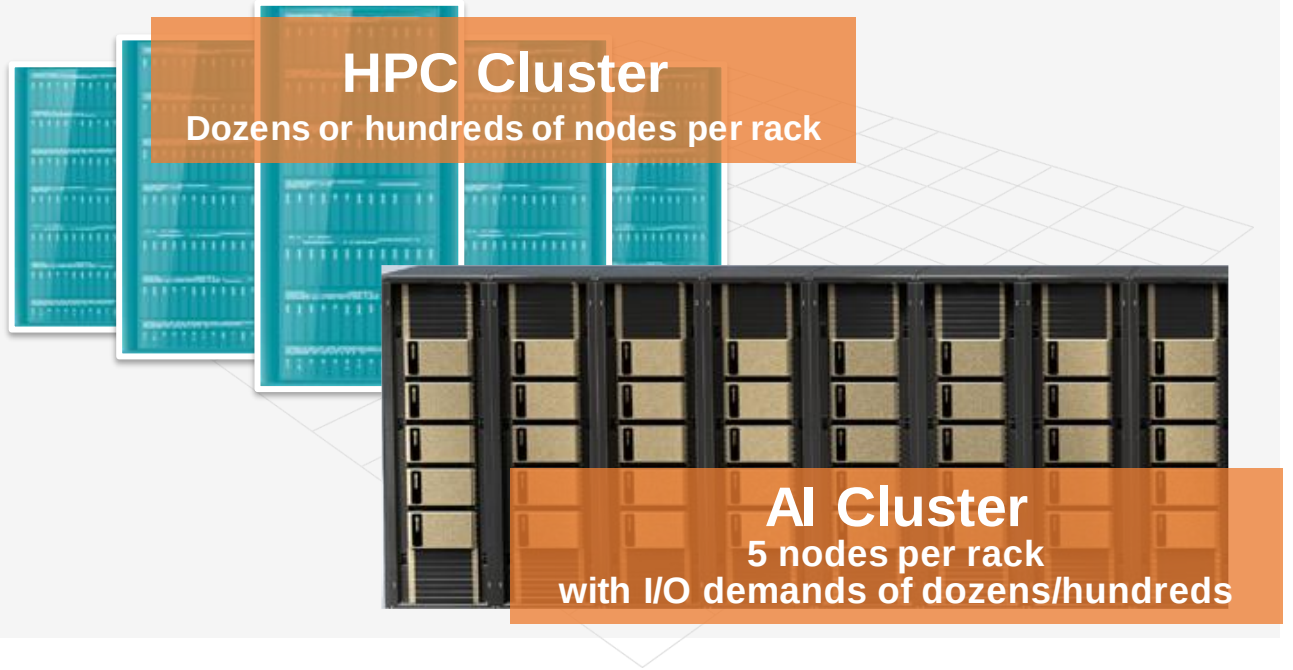
```
# 1MB random reads via host memory into GPU memory
dgx-a100$ elbencho -t 128 -r -s 10g -b 1m --direct -rand
    /data/file{1..128} --gpuids "0,1,2,3,4,5,6,7"
Result: 45.7GB/s

# Read 512000 small 128KiB files via host memory into GPU memory
dgx-a100$ elbencho -t 128 -r --direct -n 40 -N 100 -s 128k
    /data --gpuids "0,1,2,3,4,5,6,7" --cuhostbufreg
Result: 139444 files per sec
```

GPUDirect Storage (GDS)

```
# Using cuFile API for GDS
dgx-a100$ elbencho --gds --gpuids all ...
```





elbencho charts

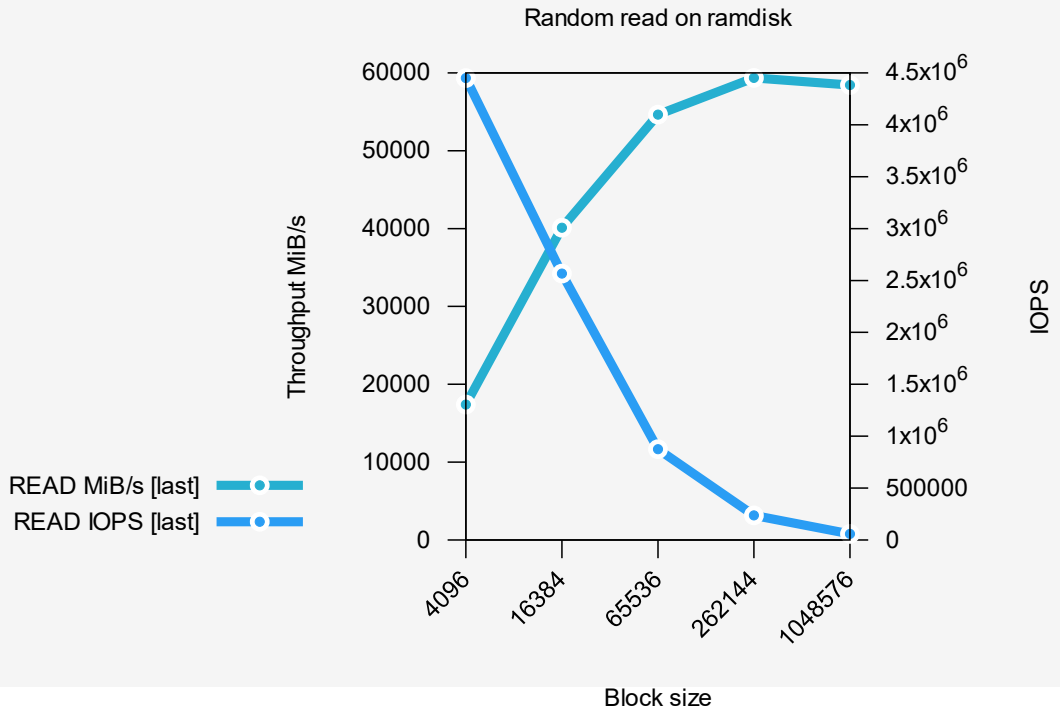
CSV file output

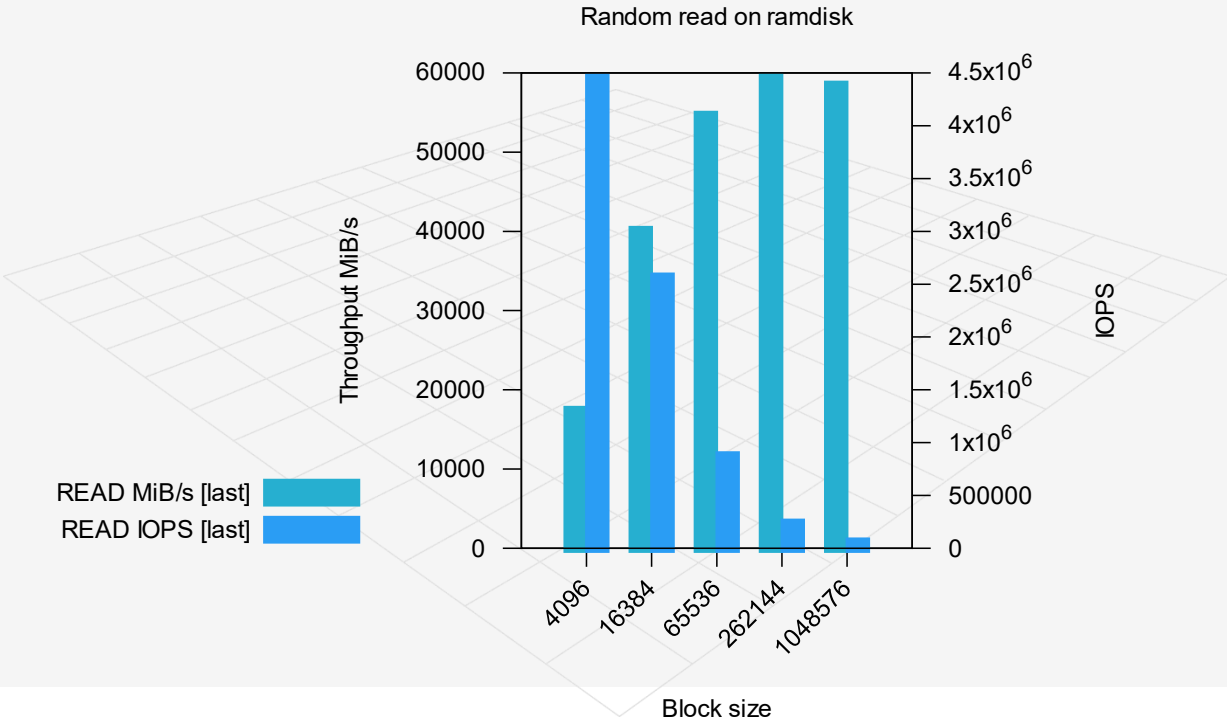
```
$ for block in 4k 16k 64k 256k 1m;
  do
    elbencho --csvfile results.csv -t 4 -r -b $block
       /mnt/tmpfs/file --timelimit 10 --rand
  done
```

elbencho-chart tool

```
# Generate line chart from CSV file
$ elbencho-chart -x "block size"
  -y "MiB/s [last]" -Y "IOPS [last]" results.csv

# Bar charts are also possible
$ elbencho-chart --bars ...
```





We live in a multi-protocol era

elbencho 2.0

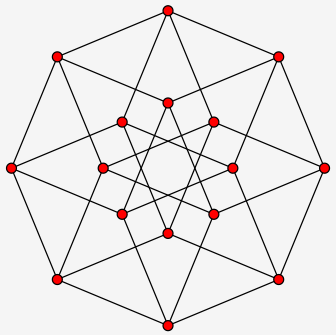
with S3 object storage support


```
$ make S3_SUPPORT=1 # Downloads a 1GB AWS SDK git repo
$ elbencho --s3endpoints http://mys3server MY_BUCKET ...
```

What's up?

elbencho 3.0

It's all about the network



```
$ elbencho --netbench ...
```

How to use elbencho's netbench mode?

Clients send/write data

Clients & servers run "`elbencho --service`"

```
$ elbencho --netbench
  --servers myserver01,myserver02
  --clients node01,node02,node03,node04
  -b 1m --respsize 1 -t 4
  -s 100g -timelimit 30
```

- **`-t`** Number of threads per client
- **`-b`** Bytes to send in each request
- **`--respsize`** Bytes to receive in each response
- **`--serversfile`**
  `--clientsfile` Hostfiles

Clients receive/read data

Reversed request/response size

```
$ elbencho --netbench
  --serversfile myservers.txt
  --clientsfile myclients.txt
  -b 1 --respsize 1m ...
```

Receive/read can switch clients and servers or switch the transfer amount in each request.

Client threads establish connections to servers in a round-robin fashion.

Simple examples

Clients send/write data

Clients and servers run "elbencho --service"

```
$ elbencho --netbench
  --servers "$(echo 172.200.5.{91,93})"
  --clients "$(echo 172.200.5.{131,133,135,137})"
  -b 1m --respsize 1 -t 1 -s 100g --lat

OPERATION RESULT TYPE         FIRST DONE   LAST DONE
========= ================    ==========   =========
NET       Elapsed time     :     35.896s     42.257s
          MiB/s write      :       10382        9692
          IO latency       :        [ avg=383us ]
```

2 servers, 4 clients with 1 thread each, 1MB-sized writes, 100GbE

- 10GB/s aggregate for all 4 clients
- Fastest client finished after 36s, slowest after 42s

Clients receive/read data

```
$ elbencho --netbench
  --servers "$(echo 172.200.5.{91,93})"
  --clients "$(echo 172.200.5.{131,133,135,137})"
  -b 1 --respsize 1m -t 2 -s 100g --lat

OPERATION RESULT TYPE         FIRST DONE   LAST DONE
========= ================    ==========   =========
NET       Elapsed time     :     12.267s     12.269s
          MiB/s read       :       15334       15839
          IO latency       :        [ avg=504us ]
```

Doubling the threads does not double the throughput, although we're not even close to the network limit.

Live stats show misbehaving clients/servers

```
Phase: NET   CPU:    3%   Active: 6   Elapsed: 10s

Latency: avg=715us

 Rank    %     DoneMiB       MiB/s        IOPS      Act CPU Service
Total    -      215909       22366       22366       24   4

    0    -      107327       11140       11140        4   5 172.200.5.91:1611 [server]
    1    -      108593       11227       11227        4   5 172.200.5.93:1611 [server]

    2   13       55750        5899        5899        4   3 172.200.5.131:1611 [client]
    3   13       55290        5681        5681        4   4 172.200.5.133:1611 [client]
    4   12       52120        5439        5439        4   3 172.200.5.135:1611 [client]
    5   12       52749        5347        5347        4   4 172.200.5.137:1611 [client]
```

Simple examples cont'd

4 threads per client

```
$ elbencho --netbench
  --servers "$(echo 172.200.5.{91,93})"
  --clients "$(echo 172.200.5.{131,133,135,137})"
  -b 1m --respsize 1 -t 4 -s 100g --lat

OPERATION RESULT TYPE         FIRST DONE   LAST DONE
========= ================    ==========   =========
NET       Elapsed time     :     12.268s     12.268s
          MiB/s write      :       22052       22052
          IO latency       :        [ avg=724us ]
---
Terminating due to interrupt signal.
```

Network saturation with 4 threads per client at 724 microseconds latency.

32 threads per client

How much latency will we get with 32 threads per client?

Simple examples cont'd

4 threads per client

```
$ elbencho --netbench
  --servers "$(echo 172.200.5.{91,93})"
  --clients "$(echo 172.200.5.{131,133,135,137})"
  -b 1m --respsize 1 -t 4 -s 100g --lat

OPERATION RESULT TYPE         FIRST DONE   LAST DONE
========= ================    ==========   =========
NET       Elapsed time     :     12.268s     12.268s
          MiB/s write      :       22052       22052
          IO latency       :        [ avg=724us ]
---
Terminating due to interrupt signal.
```

Network saturation with 4 threads per client at 724 microseconds latency.

32 threads per client

```
$ elbencho --netbench
  --servers "$(echo 172.200.5.{91,93})"
  --clients "$(echo 172.200.5.{131,133,135,137})"
  -b 1m --respsize 1 -t 32 -s 100g --lat

OPERATION RESULT TYPE         FIRST DONE   LAST DONE
========= ================    ==========   =========
NET       Elapsed time     :     34.267s     34.286s
          MiB/s write      :       22173       22503
          IO latency       :        [ avg=5.68ms ]
---
Terminating due to interrupt signal.
```

4 threads per client saturated the server ports at 724us latency.

After saturation, 8x the thread count means 8x the latency: 5.68ms

Now what?

Happy benchmarking!

Grab your copy

https://github.com/breuner/elbencho/releases/latest

https://hub.docker.com/r/breuner/elbencho

GitHub docker

Share feedback or contribute

Sven Breuner
sven.breuner@gmail.com