# Section 1.1: IO baseline

## Exercise 1.1.1: IOPS limits

Measure the upper limit of IOPS in your VM using the dd command and a minimal blocksize issuing 4M IOPS from /dev/zero to /dev/null

Hint: dd if=/dev/zero of=/dev/null bs=… count=…

## Exercise 1.1.2: IO bandwidth limit

Measure the maximum IO performance using the dd command and a 1M blocksize from /dev/zero to /dev/null

Hint: dd if=/dev/zero of=/dev/null bs=… count=…

## Exercise 1.1.3: IO Caching

Measure the maximum IO performance using the “dd” command and a 1M blocksize writing a 1G file from /dev/zero to /var/tmp/1G.

Drop your buffer cache using

$ echo 3 > /proc/sys/vm/drop\_cachesas user root.

Measure the maximum IO performance using the dd command and a 1M blocksize reading a 1G file from /var/tmp/1G to /dev/null. Repeat the measurement and explain your observation.

## Exercise 1.1.4 IO Debugging

Inspect the IO pattern of the command “yes” using the “strace” command. You have to use Control-C to terminate the command. You can redirect the output STDOUT to /dev/null.

hint : strace <cmd> > /dev/null

## 

## Exercise 1.1.5: IO Debugging

Measure the syscall tracing overhead using the “strace” command on the baseline measurement from 1.1.1. Measure the overhead when using large blocksizes like 1M. Explain your observation

HINT: strace <cmd>

## 1.2 IO Boundness and Compression

## Exercise 1.2.0: get familiar with checksums

Work on this file and implement the run() function. You can do this to a notebook and work from there if you are more comfortable with it.

https://github.com/scampana/ManagementPhysicsData/blob/master/checksums.py

## Exercise 1.2.1: IO or CPU bound?

There are many scenarios where an application can run IO or CPU bound. A good estimator for IO or CPU bound is (for single threaded applications without lock contention and timers) the ratio of (cpu+system) over realtime. If it is close to 1 we identify an application CPU bound. E.g. an application with (cpu+sys)/real = 0.5 is certainly IO bound, for values over 0.9 it is most likely CPU bound - in particular when running inside a VM. The interpretation is that your application processes data faster than it can arrive from the storage devices/system.

As a preparation create two 1 GByte files with different contents (use a blocksize of 1M):

* create a file /var/tmp/1G.null reading zero bytes from /dev/null
* create a file /var/tmp/1G.random reading random bytes from /dev/random

Q1 Measure the time to compute an MD5, SHA1 and SHA256 checksum of these files twice each and categorize if the application runs in your environment CPU or IO bound.

Q2 Which component on your computer is the execution time defining bottleneck?

Q3 What is the checksumming bandwidth of the three algorithms?

Q4 Is it sensitive to the file contents?

Q5 What is the effective blocksize used by these applications?

The commands to compute these checksums are md5sum sha1sum sha256sum and they take your file path as an argument.

HINT: You can prepend the time command to measure real,cpu and system time of an application. Compute the (cpu+sys)/realtime ratio to classify your application.

## Exercise 1.2.2: IO compression

“gzip” is a standard compression tool to reduce storage volume of data dealing volume vs cpu time. The application to compress is gzip, the application to decompress is gunzip and takes your file path as an argument.

Q1 Measure the time to compress the two files from exercise 1.1.6 and compute the compression bandwidth with respect to the input stream!

Q2 Look at the filesize after compression. What is the compression ratio? Can you understand the difference?

Q3 Measure the time to decompress the two files and compute the decompression bandwidth with respect to the input stream!

Q4 Is the IO pattern of gzip gunzip independent of the file contents?

HINT: After the compression the file path is appended with a .gz suffix! After decompression the suffix is removed!

# 1.3 IO Latencies

## Exercise 1.3.1: Latencies for remote IO

time and strace the “curl” command to retrieve the CERN main page home.web.cern.ch. Use the strace option to display the real time as the first column (-ttt) and restrict the syscall output to network related calls (-e network).

$ curl home.web.cern.ch

Identify the network latency and explain the execution time of the command.

Hint “time strace -ttt -e network curl home.web.cern.ch”

## Exercise 1.3.2: Latencies for Remote IO

You have straced an analysis application using files on your local hard disk. You see that it issues synchronous read calls with the syscall statistics output as shown:

% time seconds usecs/call calls errors syscall

19.30 0.039031 1 20000 read

14.61 0.000143 143 1 1 connect

The realtime of the application running with local data is 10 seconds and it is CPU bound.

Assuming that the application uses the same IO model for remote file IO, which runtime do you expect for the application to read a file from a remote location with a network RTT of 20ms.

## Exercise 1.3.3: Latencies for Remote IO

The ROOT application uses an optimization called TTreeCache to improve remote data access.

Taking into account exercise 1.3.2. which kind of optimization could that be? What do you need to do to reduce latency effects?

# 2.1 RAID Technology

## Exercise 2.1.1: The magic of XOR

Redundancy algorithms are most of the time built on top of XOR operations or more general galois-field multiplications to compute parities. The XOR algebra has some very particular characteristics. Remember:

|  |  |  |
| --- | --- | --- |
| XOR | 0 | 1 |
| 0 | 0 | 1 |
| 1 | 1 | 0 |

If you compute the following parity for arbitrary length data blocks X,Y,Z:

P=X^Y^Z

This formula is used to implement a RAID-4 or RAID-5 system. What is the result of the following equation

P^X^Z^Z= ?

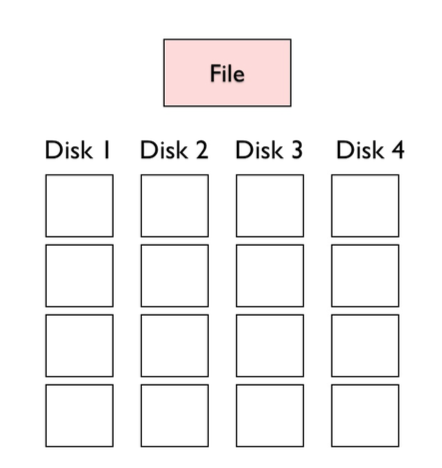
If you have a data blocks in your RAID system not fulfilling this equation, what can you conclude?

HINT: Like for the sum operation the order of the operation does not matter!

## Exercise 2.1.2: RAID 0 striping

Striping allows to involve several disks for single file IO by chunking files into equal blocks (stripe size) and storing them round-robin over many disks.

Imagine you have got a large file of many GB size. You are asked to stripe this file with the best stripe size to optimize to issue 4MB read requests from an application.

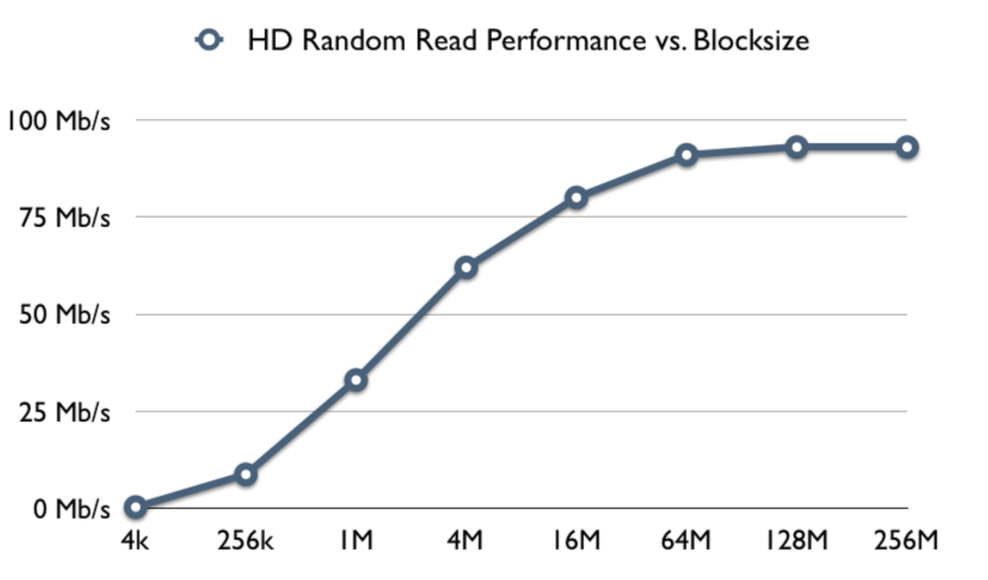


Which blocksize do you chose? Pin down the stripe pattern by labeling the empty blocks with their block number where the block number is defined as the start offset position divided by the size of a single block (blocks are numbered from 0 on …)

HINT: You should divide blocks in a way that you involve as many disks as possible to read a 4MB data block.

## Exercise 2,1.3: RAID-0 performance

Consider the following measurement of the random read performance of a single hard disk.



Imagine the following scenario:

* you have striped ten 100 GB files sequentially over four hard disks and a stripe size of 1 MB - you created one file after the other
* each client can accept data faster than the aggregated maximum performance of all hard disks (client is not IO bound)
* each client does sequential file reading issuing 16 MB block read requests (the request is issued in parallel to all concerned disks)

Which performance do you expect from each single hard disk if you download one file after the other?

Which performance do you expect from each single hard disk if you download all files in parallel?

HINT: Figure out the IO pattern happening on a single disk. You can assume that concurrent block reads (of 4MB) on a single disk are executed sequentially and not overlapping. Do you see random or sequential IO for the two cases?