File IO and Network IO Challenges and Solutions

Warwick RSE

February 2023

SCRTP

- Runs the SCRTP desktop machines
- Runs the Task Farm system
- Runs the associated storage systems
- Runs the Clusters
  - Avon
  - Orac
  - Sulis

Aims

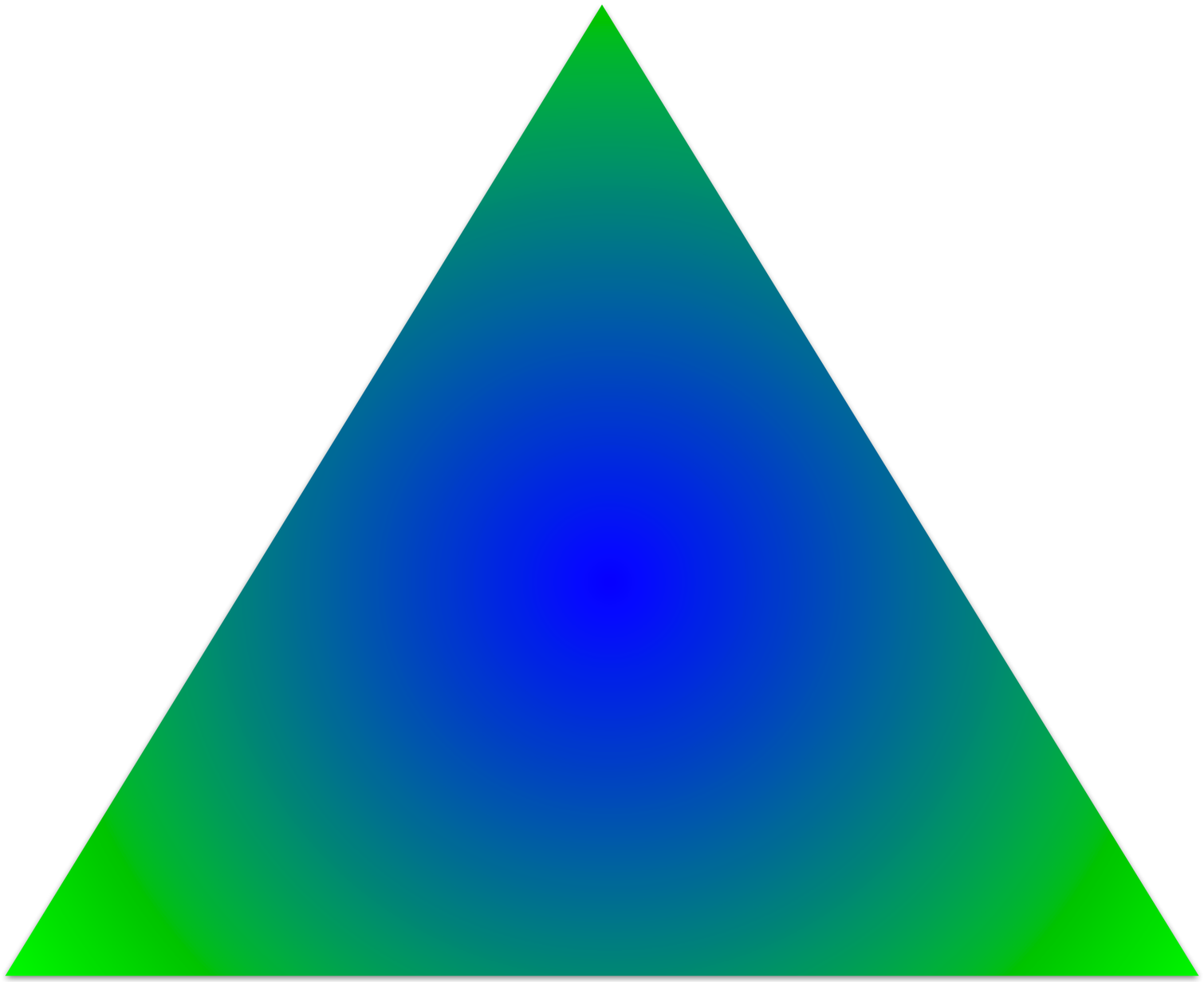
- Dealing with IO heavy workloads is a hard problem
  - Lots of data
  - Data that needs to be written fast
  - ~~Realtime data logging~~
- There are general approaches that work to improve matters
- An understanding of what is happening can help you deal with other problems

Problems

- Difficult IO problems can cause
  - Increase in total time to solution for a problem
  - Limit on scaling of parallel code
    - Generally hard to get actual scaling of IO - a **good** solution writes data at the same rate as a single thread/rank
  - Degradation of performance for everyone else on a shared system

Design Balance

Compute



Storage

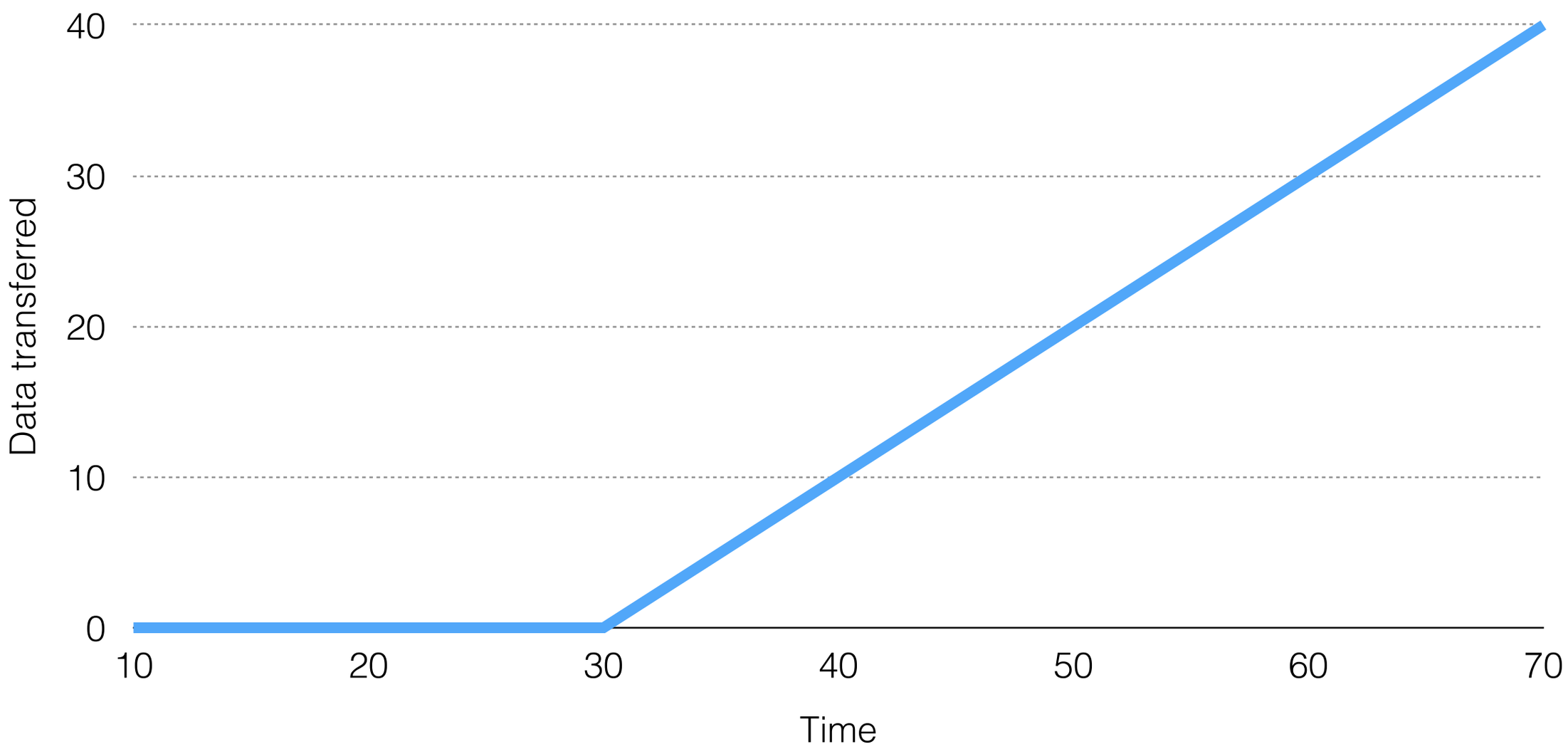
Memory

Terminology

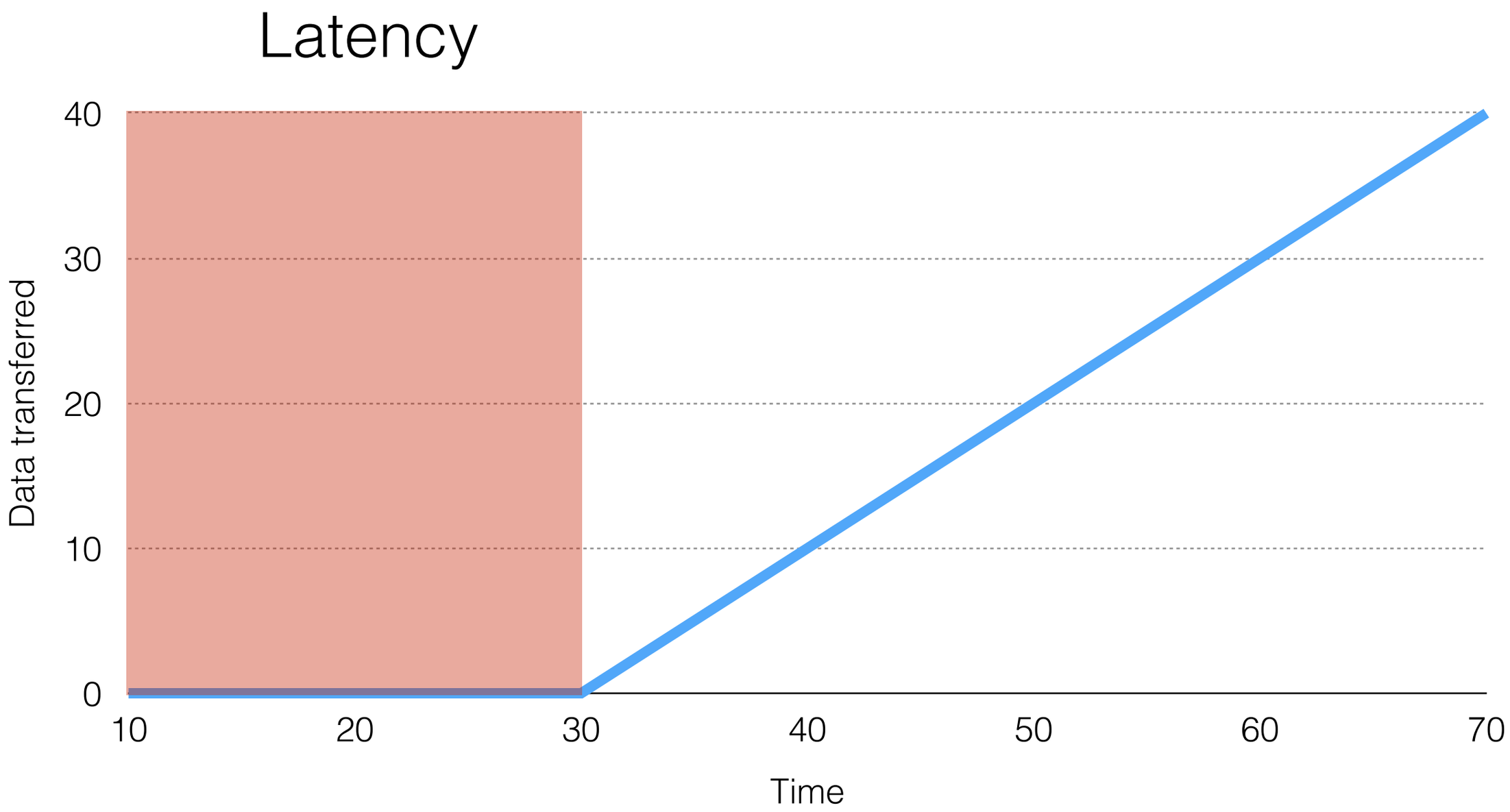
Terminology

- Latency - the time between a program asking for data and it being made available, usually measured in seconds (or milliseconds etc.)
  - One measure of network latency is the “ping time”
  - High latency is bad
- Bandwidth - the rate at which data is transferred once it is flowing, usually measured in GB/s (or MB/s etc.)
  - So you also have a higher bandwidth asking for data from memory than from disk
  - High bandwidth is good

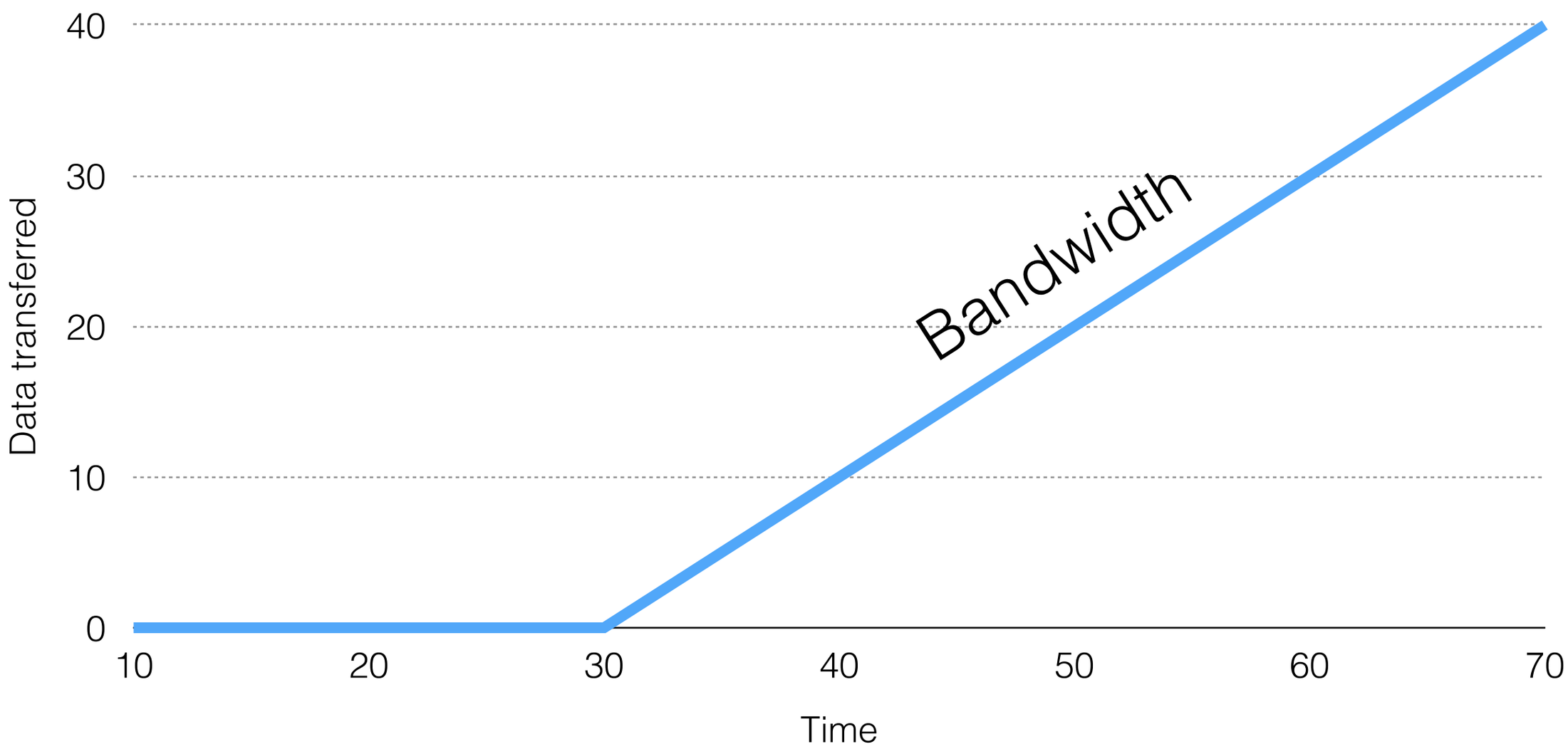
Latency



Latency



Bandwidth


40
30
20
10
0
Data transferred
10
20
30
40
50
60
70
Time
Bandwidth

Terminology

- Bandwidth matters for large files
- Latency matters for small files
- A given system will generally be tuned for one or the other
    - Some things improve both but generally either at cost or by being less reliable

What happens?

What seems like happens

- You call a write function in your code

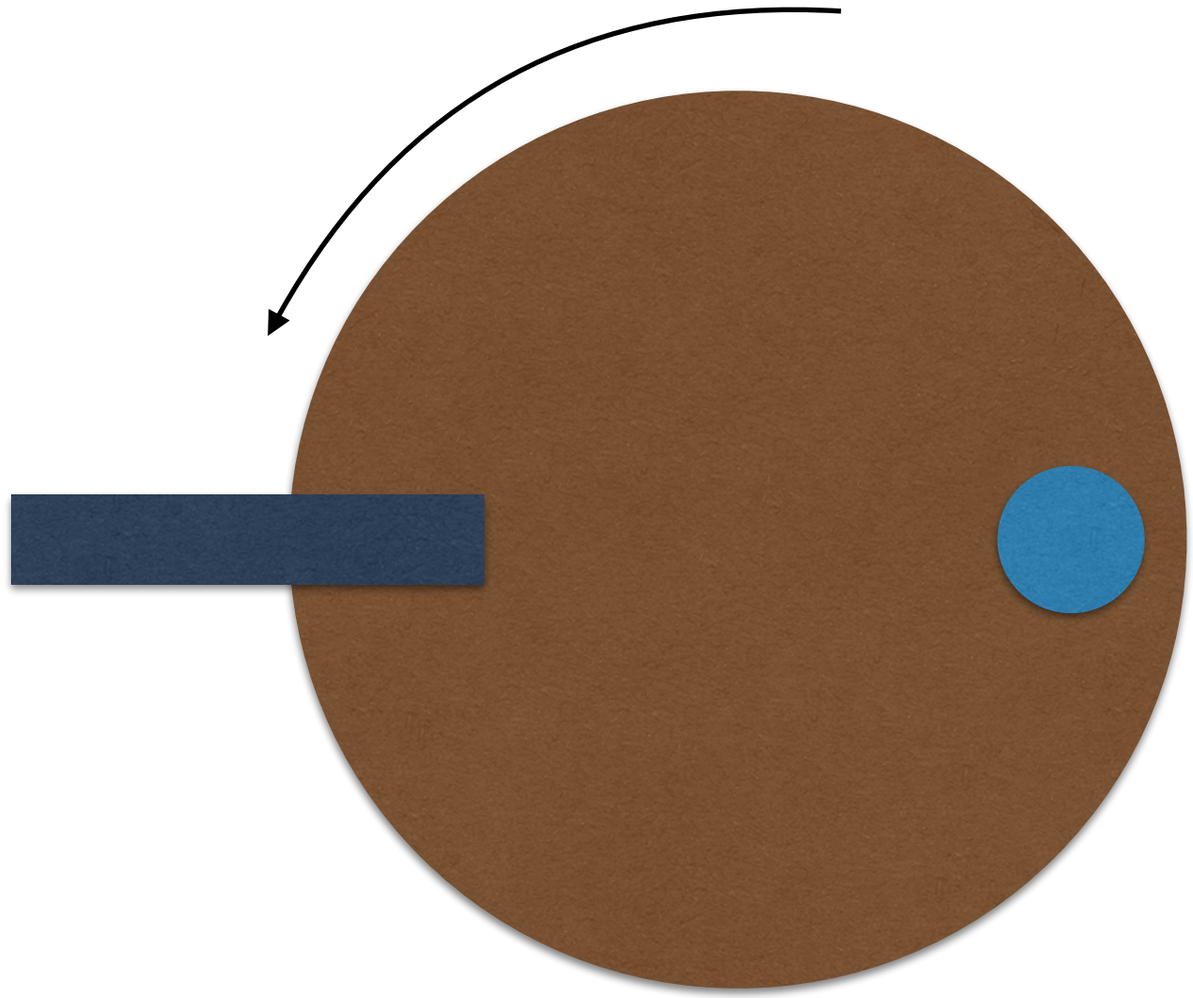
- The data gets written to disk

Closer to reality

- You call a write function in your code
    - Almost every language only specifies that this function will return when you can safely reuse the input variable
- Your language may require (C++,C) or permit (Fortran,Python) buffering in the language
- Once your language writes the data your OS may buffer output
- The filesystem may the buffer output
- There may be a RAM buffer in the hard disk unit
- Then finally data winds up on the disk

Why buffers?

- Can only write (or read) when the right bit of the disk is under the hard drive head (SSDs are different but we'll ignore them here)
- Long and unpredictable wait for every write
  - Latency!
- Performance of code becomes unpredictable



How do buffers help?

- Buffers combine output at various levels
- Multiple writes are replaced by a single write
- Rather than paying disk latency for every write you only pay it once when the buffer is actually written
- Your program can continue as soon as the output data is safely in the buffer
  - "Effective" bandwidth massively increased

What are the problems?

- Mostly buffers are unambiguously a good thing

- Even if you are both reading and writing data from a file it'll be sorted out - you will never see an "old" version of the file when reading

- The only problem is that if your program crashes before the (language/runtime) buffer is written out

  - You can manually flush the buffer to avoid this but this removes the performance benefit

  - Your code should only crash while you are debugging it so write your codes debug mode to flush log files etc. don't use it in production mode

Any surprises?

- A couple
- Closing a file generally flushes output
- Output to screen generally flushes the output buffers when a newline character is received (UNIX-like systems at least)
- In C++ writing std::endl to a stream **always** causes the buffer to flush
  - “Inserts a newline character into the output sequence os and flushes it as if by calling os.put(os.widen('\n')) followed by os.flush()”
  - If you want a platform independent newline then use os.widen(‘\n’)

Takeaway notes

- File buffers are ubiquitous and generally very good for performance
    - You can sometimes tune the sizes of some of them which for large data writes can be helpful
- It is tempting to turn them off because of the data loss problems, especially for debugging logs etc.
    - Make this an option if you do it! You don’t want it on for normal production
    - Look into alternatives
        - Code that fails gracefully under all circumstances - buffers flush properly
        - Logging to external program that doesn’t crash - database IO

Files and Filesystems

File systems

- All computers nowadays (and really since the 1960s) have **filesystems**

- That is a part of the computer operating system that manages files and directories on the disks

- It handles storing data to physical locations on the disk, mapping filenames to those physical locations and handles the hierarchical directory structure

- All of this information about how it does this is also stored on the disk (in normal systems, there are odd ones)

  - **metadata**

File systems

- On UNIX systems like Linux the metadata is mostly of the form of things called **inodes**
  - Index nodes
- Each inode refers to a single file or directory on disk
- When a file is created (and sometimes when it is modified) the **inode table** has to be modified to show where the file is being stored
- Often about 5% of a disks total capacity is used to store inode information - it’s why hard drives don’t have their nominal capacity when you finally see them in the OS

Performance

- There is a cost for updating an inode
    - In general the inode table won't be near your data on the disk
- This cost is O(1) in filesize
    - It takes as long to do for a small file as it does for a large file
- Smaller files are slower to write than larger files (per byte written)
    - Also, just opening a file can take a long time

inode exhaustion

- The other problem with many small files is that for “normal” filesystems a fixed number of inodes are created when the file system is created and more can’t be created easily

- You can exhaust the inodes by writing many small files even if you still have space on the disk for more data

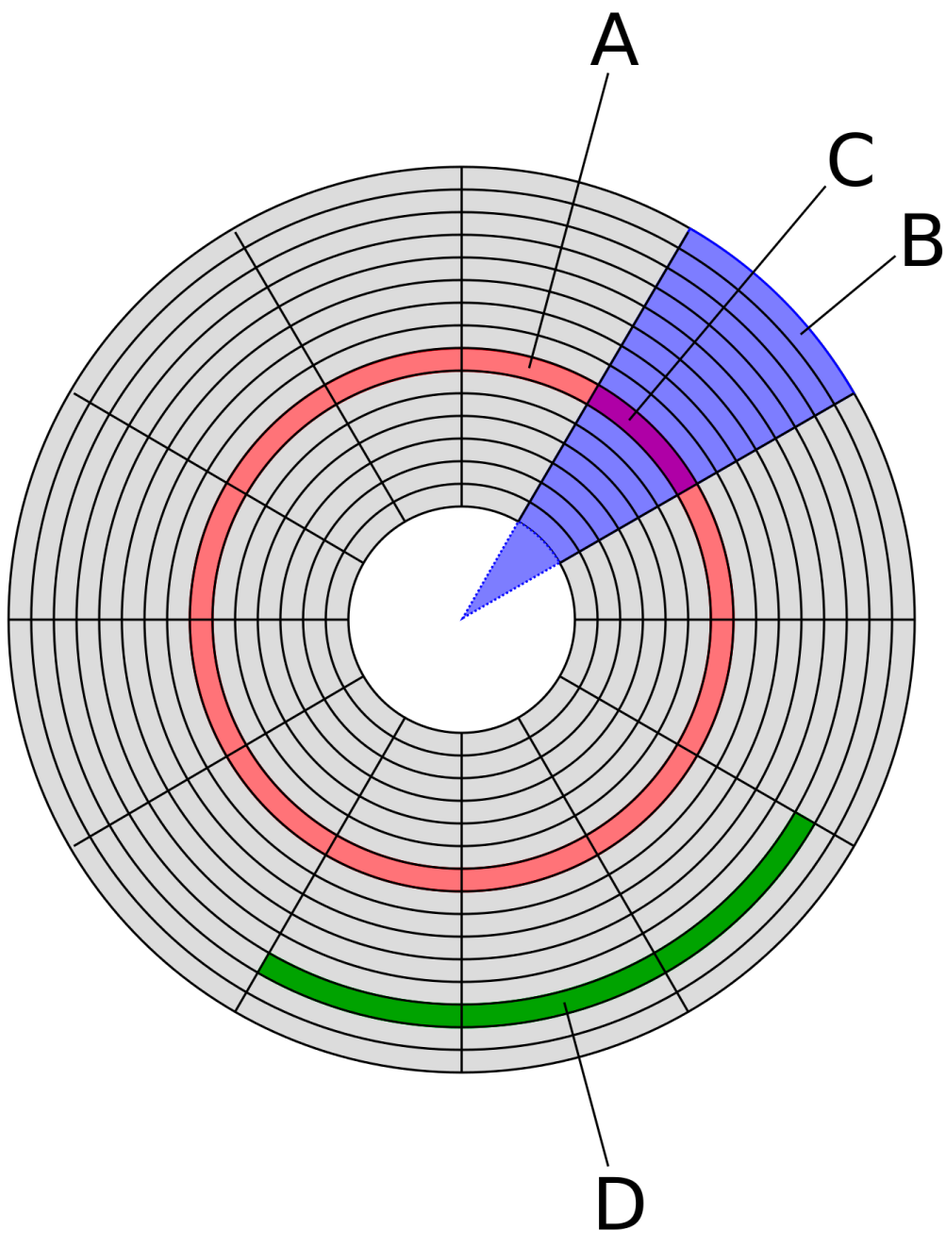
- Filesystems can be tuned for any given purpose but on shared systems it is always tuned for general use (mix of small and large files)

Takeaway notes

- Filesystems are always a tradeoff between use cases
- Very small files are often disproportionately slow to write
  - Overhead of creating file
  - Overhead of writing metadata
- Even if you can tolerate the performance drop you can run into scenarios like inode exhaustion
- **Don’t write many small files!**

Blocks and Block Devices

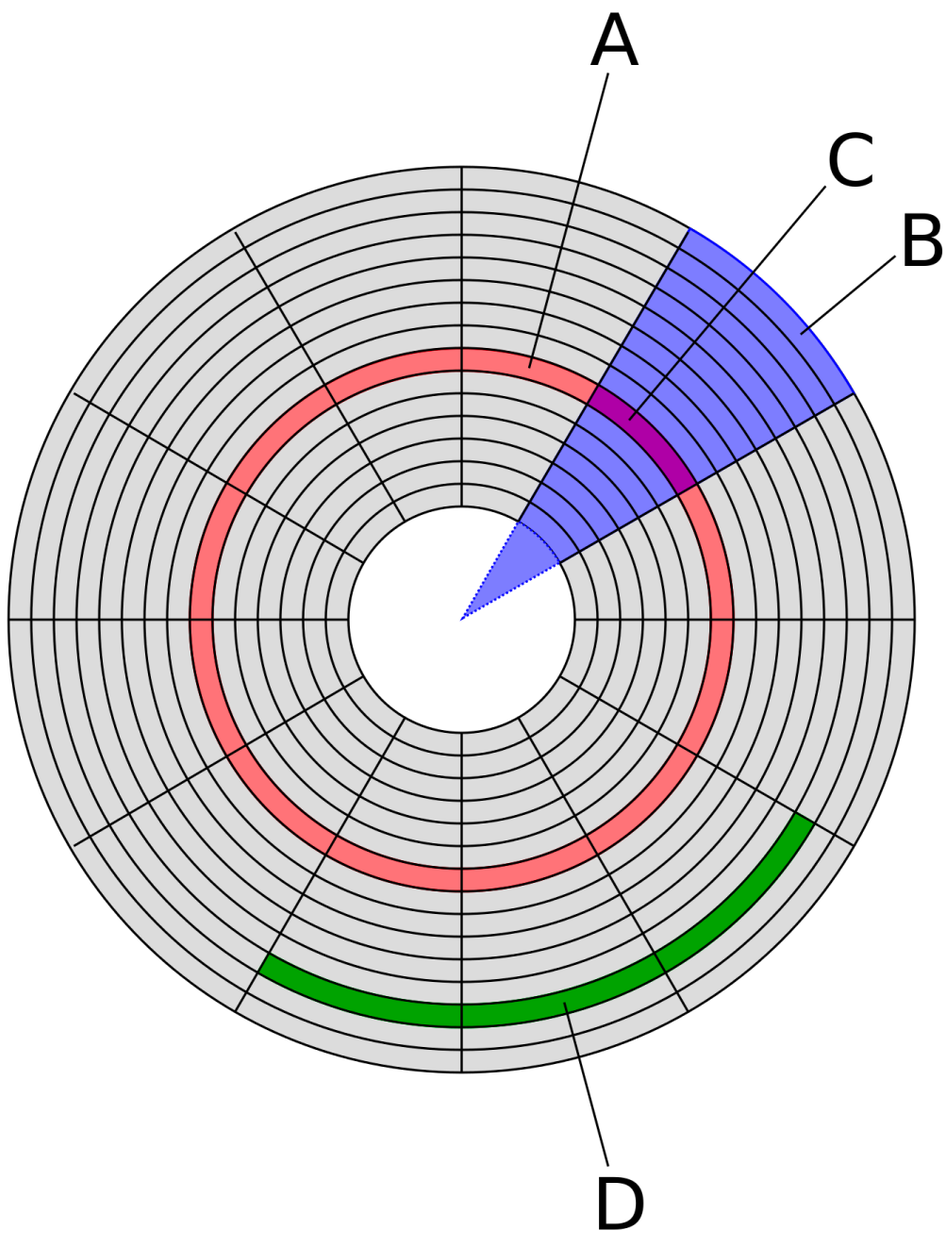
File system blocks



Heron2/MistWiz
Wikipedia, used under license

- Filesystems are abstractions over how disks actually lay out data
- For example the left shows how data may be stored on a magnetic or optical disk (actually only true for really old ones!)
  - A - Track (or cylinder for drives with multiple disks)
  - B - Geometrical Sector
  - C - Sector (usually about 4k of data)
- So a piece of data is written to a given track on a given sector (with a given **head number** if there are multiple disks)

File system blocks



Heron2/MistWiz
Wikipedia, used under license

- D shows a **block** (sometimes called a cluster or allocation unit, but we'll stick with block)
- A block is the smallest part of the disk that the filesystem writes to or reads from
- Some filesystems allow multiple files in a block (block fragments) but the most common ones don't
- Assume that files have a minimum size
  - Typically about 4kB but can be much bigger

Takeaway notes

- Filesystems don’t write files with bitwise granularity and drives don’t store data in an undifferentiated soup of bits
- Data is generally written and read in units of the filesystem block size
  - This can be much larger than the actual size of a small file
- Small files cannot take up less than a block of storage (on most filesystems)
  - Small files take up more space than their actual contents
  - Technically larger files are always rounded up to the next blocksize boundary, but 4k on a 10GB file doesn’t matter as much as 4k on a 20byte file

Network filesystems

Network filesystems

- UNIX OSes generally make filesystem boundaries seamless

  - In Windows/DOS by default you get a different drive letter for each filesystem

- On UNIX you can make a different filesystem appear anywhere - it just looks like a directory

- So on an SCRTP machine looking at the folder */usr* you are on the local hard drive, */home* you are on a network filesystem on the machine **hermatus** and */storage* you are on the machine **nef**

Network filesystems

- There are a lot of network filesystems but two that you will “normally” encounter
  - SMB - Server Message Block - originally IBM but championed by Microsoft
  - NFS - Network File System - Originally Sun Microsystems but now an IETF open standard
- We use NFS version 4 here at Warwick for all network filesystems on the “normal” machines

A bad joke

"Hi, I'd like to hear a TCP joke."
"Hello, would you like to hear a TCP joke?"
"Yes, I'd like to hear a TCP joke."
"OK, I'll tell you a TCP joke."
"Ok, I will hear a TCP joke."
"Are you ready to hear a TCP joke?"
"Yes, I am ready to hear a TCP joke."
"Ok, I am about to send the TCP joke. It will last 10 seconds, it has two characters, it does not have a setting, it ends with a punchline."
"Ok, I am ready to get your TCP joke that will last 10 seconds, has two characters, does not have an explicit setting, and ends with a punchline."
"I'm sorry, your connection has timed out. Hello, would you like to hear a TCP joke?

Network problems

- That shows a chunk of the problem with network filesystems

- Networks and network protocols are generally quite chatty

- Latency goes up massively as soon as things are going to move over a network

- Once the server is more than a few meters away from the client even speed of light delays can come into play on latency

Network problems

- Network bandwidth will be between 1Gbps and 10Gbps generally
    - Not a huge limit for a single user but can sometimes can be for many users
- Bigger problem is that both the server and the client actually have to do work to communicate
    - Much of the work is in opening and closing files **not** in writing so the total effort on the server is mostly controlled by the number of clients
    - Most of the rest of the effort is O(1) in the size of the write (I want to write this much data to this file, here is the data) so many small writes are bad (Buffers!)

Advanced Systems

- Systems like clusters and tightly integrated server systems can improve matters somewhat

- Fibre Channel systems are used to attach storage to servers

- Infiniband networking is faster and much lower latency than normal ethernet networks (less chatty!)

- Mostly money is spent where it does the most good!

    - Most money for IO infrastructure in SCRTP goes into the systems on the clusters

Takeaway notes

- Network file systems are much like local filesystems in general **BUT**
    - Latency is much, much higher
    - Bandwidth is lower, especially to and from the machine you are sat at
        - Better between datacentre machines and actually better than a normal desktop in the clusters
- Network filesystems have to be designed and tuned for their intended purpose
    - The home areas of SCRTP machines are designed for interactive sessions **NOT** heavy data output

What can I do?

Limit Opening Files

- Opening a file takes time
  - Very fast on a local machine
  - Fairly fast over a network
- This time is O(1) in file size
- Don’t open multiple files
  - Keep files open
  - Write output to one file and seek within it rather than saving to multiple files

Don’t flush buffers

- Be aware of what in your language might flush the output buffers unintentionally
  - Performance can drop drastically!
- If you need to flush buffers in case the code crashes write a debug mode!

Be careful of workflows

- A lot of science involves combining several codes into a workflow

- Mostly programs are linked together by reading and writing files

  - If programs complete very fast this can become a problem

- Combine programs into a single program (if possible)

- Use /dev/shm (shared memory filesystem)

  - **WARNING** this is a literal RAM disk - everything in it will disappear if the computer is reset

Be careful of workflows

- Workflow managers (like Dask or Snakemake) can be troublesome as well

- They often work in slightly pathological ways themselves (i.e. Snakemake when submitting slurm jobs polls the slurm daemon repeatedly to test for completion)

- It can be very easy to write a workflow that will stress a system badly

- Run real scale problems by hand and see how long each phase takes to run - check if you are going to be producing a workflow that is troublesome

Database IO

- If you have a lot of independent programs running simultaneously then it looks as though you have no choice but to have them all output their data individually - they're independent

- What works better is to have a single program running continually that takes input and writes it to disk using suitable strategies

    - Generally this will be a Relational Database Management System, or more simply a database

- Not trivial to move over to this but not horrible - email us if this is something that you are interested in

Use the right hardware

- Performance is different for different types of disk
- Testing on a random computer I get
  - 1.2GB/s on the M.2 SSD
  - 197MB/s on a RAID5 array of HDDs
  - 3.7GB/s on /dev/shm
- Latency is lower on the SSD and almost zero on /dev/shm

Use the right hardware

- **Clusters!**

- The best hardware that we have is put into the clusters

  - GPFS high performance distributed parallel filesystems with memory buffers and SSD “burst buffers”

  - Smaller “scratch” space using pure SSD based systems

- If you find yourself running into problems with performance on systems like the task farm one of the clusters might meet your needs better (especially Sulis)

Parallelism

- There is one specific case of writing lots of files that is so common that there is a “proper” solution
    - MPI parallel code where each rank writes it’s own output files
- Use MPI_IO features instead
    - Not hard to program yourself if you want to (we have some training material on the RSE webpage https://warwick.ac.uk/RSE)
    - Libraries like HDF5 and pNetCDF already use it

Final surprises

- Lots of “normal” pieces of software are written on the assumption that they are running on “normal” computers
  - i.e. low latency penalty for reading or writing files
- You might find that a piece of software that you are using has performance issues due to this when running on systems like the SCRTP where your home directory is on a network
- Check for settings to change where files are written
- Try other software?

Conclusions

- Data IO is a major problem for modern scientific computing

- Try to minimise
  - Opening files
  - Flushing buffers
  - Running multiple programs
- Use the right hardware for the job