Big Computing Challenges
Part 3: Files and Folders

"The Angry Penguin", used under creative commons licence
from Swantje Hess and Jannis Pohlmann.

Warwick RSE

December 2022

Filesystems

What is a Filesystem?

- Computer disk contains a series of 1s and 0s
- Something has to know where one file ends and the next starts
  - Filesystem does this
- Aside: How do bits become text, images, music?
  - Files have encodings - embedded data saying what they are
- Files have rough minimum size to contain anything useful

There’s more than one sort of filesystem, with pros and cons as with most technology. Some are designed to be very robust, making sure files can’t be easily lost. Others are designed for certain sorts of access patterns. On cluster facilities the two you might encounter are GPFS and Lustre.

Many versus Much

- No real point designing a system to accommodate so many files that each can only contain a single byte
- But no point actually limiting this - empty files are allowed
- Systems end up with two limits:
  - Total amount of data
  - Total number of individual files/folders

Losing Count

- Common systems need to have one so-called ‘inode’ per file/ folder
  - index node - unique location in filesystem index
- Total number of inodes is limited
- Occasionally codes which produce many many small files (or deep directory structures) exhaust this long before they hit total size quota
- Nothing sysadmins can really do (buy a new filesystem?)
- We’ll get to the things you can do shortly!

Quotas

- Filesystems on shared resources rely on **quotas** to limit how many files you can have and their size
- Logging in to many systems needs file write access
  - Bust your quota and you can be locked out!
- Quotas are slightly soft
  - Nearly always flexible enough to let you fix if you mess up
  - Might be able to ask for temporary boost if you REALLY NEED IT
    - Check system docs

Soft quotas can be implemented in various ways but generally they work by giving you a certain amount of time in which everything still works but you have to go back below your quota before the time expires or you won't be able to use any more disk space.

Mostly soft quotas are also supplemented by hard quotas that you really can't go past.

Quotas aren't **designed** to lock you out generally if you exceed them but logging into a computer quite often generates various forms of file and the log in fails if these files can't be created. Hence if you can't create files because you have reached a quota, you can't log in

Metadata Problems

- Big parallel file systems have other complexities
  - Dealing with files and information about them (their metadata) needs compute
  - Write too many files too fast and you can stress "metadata servers"
  - Even if you aren't stressing I/O rates

This is unlikely to happen in general, but it is yet another reason why making lots of tiny files is "impolite". See later for some options to improve this if you need them

Burst of Speed

- I/O rates on e.g. Sulis can hit 10GB/sec for write
  - Sounds like a lot, but still not enough for some applications
- Slick but pricy solution and a more affordable alternative? USE BOTH!
- Like adding cache space to processor (smaller but slicker than RAM/memory)
- Filesystems can employ “burst buffers” to manage high loads
  - Consult docs/support if IO demands impact you

At 10 GB/sec you could fill a 1TB quota in a few minutes, so this is quite a fast rate. However it is possible to have this impact your code run times. Another thing to look into only if you need to.
Burst buffers often employ cutting edge storage technology, often locally on each node (so no network connection at all). Codes write to disk normally, but the system intercepts this (under some circumstances) and stashes the writes on these very fast disks. Then they more slowly spool out to the main storage. Often there is nothing you need to do - consult the docs to see.

Scratch That

- Some systems offer a scratch space for temporary data
  - Often optimised for speed
  - Not backed up! Temporary only
    - Some systems purge (clean out) scratch space automatically
      - After job finishes, daily, weekly etc

Scratch space is mostly meant for temporary files only needed while a code runs for example. Different clusters use it a bit differently, so you need to check the docs as usual.
For instance, some systems put the scratch space on the nodes directly, so your access to it disappears once your job completes. Others let you access it always, but clean it up (delete files) automatically.

Out of the Archives

- At the other end of the scale is long term storage
  - May be large amounts of data but infrequently accessed
  - E.g. archival space
- Some systems have tape drives (actual magnetic tape, really) for this
- Others have archive stores with slower uplinks etc
- You know the drill by now: see docs/support!

For instance, at some places there are physical magnetic tapes which are stored in a giant filing system. To get your data, a tape robot has to locate the tape, physically move it to the tape drive, load it up, and then you can start to read it. As you might imagine, this takes a bit of time! Some locations even need you to request data hours or days in advance although this is more common for shared data sets, not cluster output storage - see a few slides later for more

Moving Things Around

- Talking about uplinks, even if you have space for data, can you afford to shift it?
- Recall Bandwidth - amount of data per second but also latency - delay prior to transfer
- Time to transfer files depends on BOTH
- Can also depend on how MANY links from you to them

By the way, for some sorts of processing you really don’t want to be reading files over a network link! You need to transfer them to somewhere local and work from there, or ensure they are loaded into your local machines memory (if size permits). If you don’t, your data processing might slow to a serious crawl

Getting a Move On

- Old joke: sneaker net - PhD student sent up the corridor with a hard-drive, courier sent from CERN with a DVD …
- Not really feasible now as volumes are simply so large BUT
  - Still might want to think about WHERE physically your data resides

For us, this is mainly an anecdote to remind you to keep in mind that processing data that is physically many miles away can introduce inefficiency in your process. You might want to do processing directly on the cluster machine and transfer only the results to your local machine. You might produce intermediate data, or even plots, this way.

The sneaker-net really existed, both for transfer speeds and security. You really could request data from sites and would receive it in the form of a DVD in the post. Security wise, this comes up for data sets with security implications, where they might be kept behind an “air gap” i.e. on a computer which is not connected to any network, to make it as secure as possible from hackers. You’ll know if you end up dealing with anything like this, because you’ll be signing paperwork first!

Out of the Archives Again

- Also need to consider archival data restoration
  - Those big cheap storage centres have to cut costs somehow
  - Might have a lead time to to request data
  - Might have very limited bandwidth
  - FIND OUT BEFORE you need to know

Data in the Clouds

- With cloud systems, you MUST check their charging model
- Some might charge you twice
  - Once to store data
  - Once EVERY time you access it
- Try not to keep pulling data out of the cloud!

We keep mentioning cloud charging - don’t panic, but do be aware that there are all these different facets you might need to look at.

Problems with Files

- Reducing amounts of data to make preserving it easier is tempting but do beware!
- You need to make sure to preserve data such that results can be reproduced
  - Might mean keeping only inputs etc
  - Might be able to keep some sort of intermediate data
  - Have to think about this before starting

Research Value

- But ought to also consider “additional research” value
  - Do you know exactly how much data you need to output/store?
  - Might somebody else be able to process your data and do other work with it?
    - Should you enable this?
  - Research council wishes?

Too Many Files

Too many files

- Codes that produce many small files lead to many problems
  - Slow for you - opening files is overhead, small writes are less efficient
  - Bad for filesystem - use up inodes, hammer metadata servers
  - Hard to work with - consider listing, moving, combining

This is about that filesystem and those metadata servers again. They have to do work to find and “understand” your files, which is an overhead - it takes time and processing power. They have to do work to create a new file, keep a record that it exists, where it is etc.
Moving files requires rewriting the records of where they are, how big they are etc.
All of these things tend to “cost” an amount of time/processing per file, so a single big file takes one unit, and the same data spread into 100 files takes 100 units.

Too many files

- What to do?
- First thing to try is always cooperative solution
- Look at what your code or approach can already do
  - Can you simply output less data?
  - “In-code data reduction” - output derived quantities directly
  - Look for ways to combine files - perhaps you have an option to output variables to combined or separate files?

Too many files

- If you really need all the files (or can't spare the time/effort) consider second-order solutions
  - Tar/zip archives after files are produced
    - Make this part of your workflow
  - Write directly into an archive
    - Can be wrapped around existing code
- WARNING: Zip is lossless, but damage to archive might make entire dataset unrecoverable

Zipped or compressed files can be in a form where file corruption renders the whole thing un-extractable. This means IF a file is damaged, you might lose the lot. Such damage is unlikely, but might be more serious, than losing one or two individual files from a set. When you are assessing risks, you ought to consider the likelihood and the acceptability both.

Try Database IO?

- For High Throughput computing not uncommon to have lots of tiny writes and lots of data
- Maybe files aren't the answer at all!
  - Ballpark figures: thousands of files, <1kB per file
- Writing to a database instead might be a solution
- Better than an archive, stays readable

Course on this coming soon at https://warwick.ac.uk/research/rtp/sc/rse/training/databaseio

Why Database IO?

- Database IO has direct benefits too in some cases
- Processing tasks with subsets of data from multiple output files
  - E.g. read a single output variable across many files
  - Especially if you have complex searching requirements within the data in your files
- If you want to do runs with complex inter-relations in runs
  - if this run shows X then we run Y, else run Z
- Good reproducibility solution - connect inputs to outputs even in complex ways

READ it back

- Backups don’t count if you can’t restore from them: data storage is useless if you can’t read it back
- Key use of database here is to be able to read your data back and know how/when it was created
- You want to **connect inputs to outputs**
- Can add complex searching and sorting
- Help reproducibility by leaving audit trail - add enough data to know exactly what you did

Department of Redundancy Department

- A key idea for writing *good* databases is avoiding redundancy
- If you have the same data in two places, and they don't match, which is true?
- Formally called "normal form" and entire books on it exist
- For this, just keep in mind you want data in one and only one form and location

Relationships

- Databases can store not just data, but its inter-relationships
- Relations can be:
  - One to one - a person has a (legal) name
  - One to many - a person can own more than one car (*)
  - Many to one - many people may own a particular *model* of car (**)
  - Many to many - in a business fleet of cars, each may be driven by several employees over a day, and an employee may drive more than one

    * One usually has to identify a single primary owner, e.g. the person who owns the title

    ** Suppose one needs to contact all owners of a model to inform them of a fault

Database Terms

Terms

- This series is about classic SQL type databases
- These contain tables which contain a series of fixed-purpose and fixed type columns
- Tables can have many rows, each being a single record
- Description of the tables and their columns is called the schema
- Schemas should change very rarely - plan your database before you start!

Terms

- “Talking to” the database uses queries - commands that the database engine runs, usually returning some data
- Can be very simple (get everything from table X) to extremely complicated (reconstruct complete records of all pets whose owners also own a hamster and like the colour green, which (the records) were last changed on a Tuesday)
- Queries are commands - they can add, change and delete records
- Be aware of the idea of “injection” - accidental or malicious inserting of bad commands into others (see also later)

An example

Id	Pet name	Pet species	Owner id
1	Dobbin	horse	1
2	Fido	dog	2
3	Felix	cat	1
4	Nibbles	hamster	3
5	Ginger	cat	2

Id	Name	Address	Phone
1	Alice	1 The Road	07700 900 123
2	Bob	17 Fir Place	07700 900 124
3	Charlie	8 Main Street	07700 900 174
4	Diane	723 Long Lane	07700 900 184

An example

- Notice:
  - Alice owns two pets
  - We don’t list pets in the “owners” table - see next slide
  - Diane has no pets in our database and this is fine
  - We can really easily manage operations like:
    - Send a single letter to every owner who owns any cats
    - Send a separate letter to every pet (care of their owner)
    - Count how many horses there are
    - Contact any owners who have both horses and dogs

An example

- Why don’t we just use a single table?
- We could have multiple columns in each owner record
  - Limits total number
  - Difficult to look up owners from animal types (have to check every one of those columns)
- We could use multiple owner records, one per pet
  - We have to repeat the address, phone etc. What if they don’t match? What if we have to update the contact details?

An example

- Columns in SQL databases have fixed types
  - Fixed-size text (e.g. for an order ID)
  - Free text (note usually has performance implications)
  - Numbers - integers, real numbers etc
  - Dates
- Choice of type - performance, search ops

Data Processing

Processing Data

- Don’t forget if your first step is to generate data you have to analyse it!
  - Might need HPC/HTC for this step too
- Intermediate steps - DRY, partial processing, other handy steps
- Recall transfer rates - don’t try and analyse over a network link!

Processing Data

- Remember that transferring data is slow (always) and possibly expensive (cloud)
- Want to do as much data analysis as possible before moving data
  - In code data reduction (again)
  - Reduction or analysis on HPC/HTC machine
    - When asking for time don't forget to ask for time for data reduction
    - Can be a good use of interactive sessions

Databases

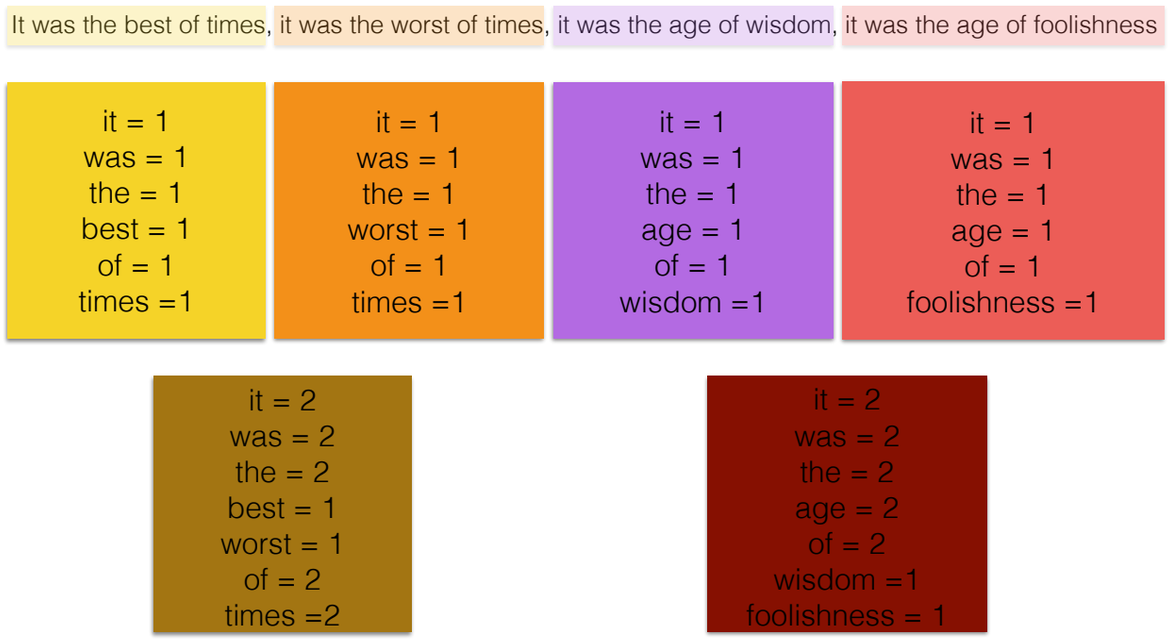
- Mostly database IO doesn't change data processing much
- File opens become database data fetches but the analysis is very similar after that
- Very large datasets from Big Compute can also come into the realm of Big Data
  - There are tools for dealing with very large data
  - "MapReduce" is a good place to start learning

MapReduce

It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness

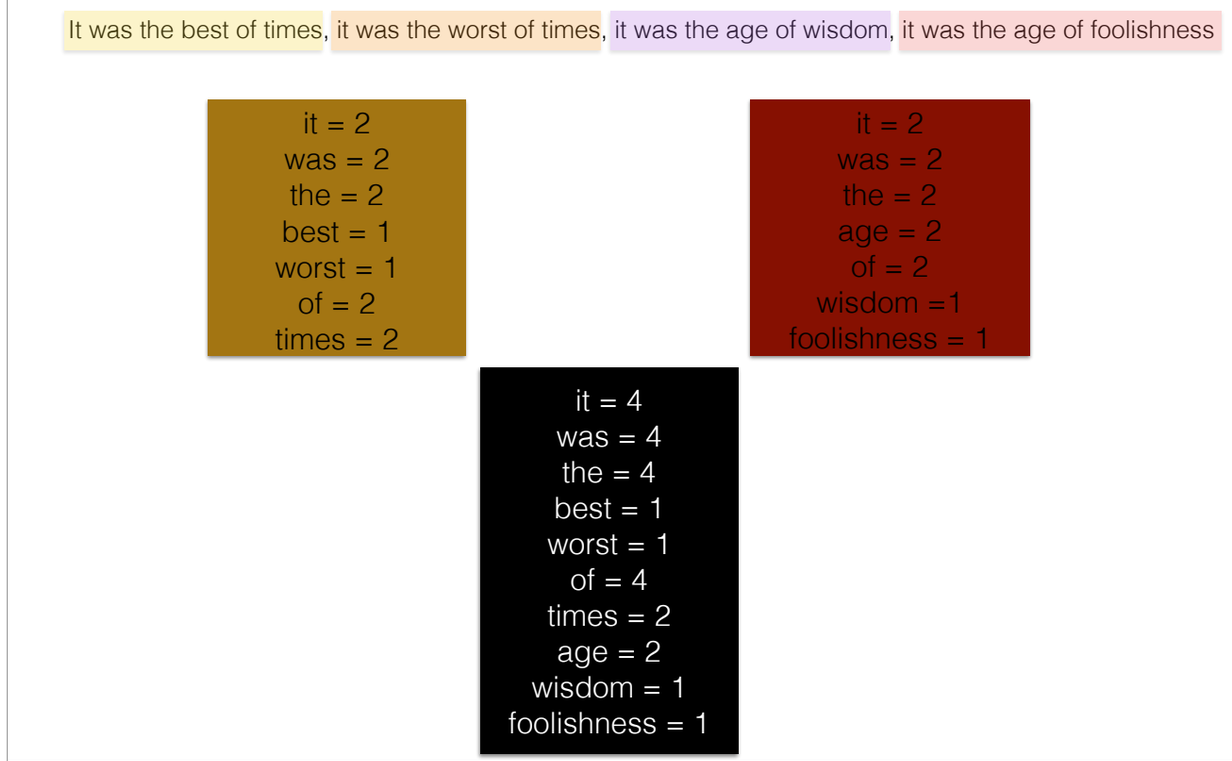
Suppose we are counting how many times each unique word appears in this sentence. For example, ‘best’ is there once. We could go through word by word and make a tally of each new word and increment its count. But this can use only one processor. What if we want to exploit more? More on next slide.

MapReduce



Let’s suppose we have 4 processors to exploit - we can split the sentence in 4 and count each segment separately.
Then, we add together the counts from these 4. But we can do better! Rather than add them together, first, plus second, plus third, plus fourth, we add them pair-wise.
Add first and second together. Add third and fourth together. More on next slide

MapReduce



Now we add the “1+2” chunk to the “3+4” chunk and get final counts.

This is an example of a map-reduce process. First we define how to *map* an input to an output, in this case the map is “take a sentence chunk and produce a list of unique words and their counts”. Then we define how to *reduce* between outputs, how to combine them into a collective output.
So far, this is an interesting way to look at things, but not actually much bonus. BUT this is such a common paradigm that systems exist dedicated to optimising the process. Hadoop (common in Big Data) for instance, is an entire cloud offering just for this. If you can cast your problem this way, you might be able to take advantage of a huge amount of dedicated work that has gone into systems like this.