Draft Notes for the Data Centre Evolution for Climate Data Deliverables

February 2017

Authors: Jens Jensen, Brian Davies, STFC

# Introduction and Executive Summary

This document describes the ESiWACE vision for the future data centre for climate data – and how we get there. It describes the current state-of-the-art both from a technical point of view (e.g. use of tapes, SMR, NVM, etc.), as well as the more abstract (desirable features of a data centre) because also the features – and the means of providing them – change with time. For example, conventional hard drives provide good storage density, but a decreasing "access density" (as capacity increases much faster than read/write speeds); thus a future data centre might combine Tier 0 (lowest latency) technologies (SCM in compute/cache, NVM, flash arrays) with Tier 3 (archive) technologies such as tape; tape provides a good storage density and excellent durability (and still has capacity for growth both in capacity, density, and read/write speeds). Similarly, developments in file systems (erasure codes, block/file replication) are expected to replace the traditional RAID disk array which has reached its limits due to the high disk capacities leading to long rebuild times.

The main focus is *Earth sciences data* and *science data* in general as there will be evolutionary pressure also from other areas of research. Even if a data centre were implemented specifically to only store climate data, it is important to take into account developments in data provisioning to other areas of science. For example, Humanities have long led the way in long term preservation of data, and HEP is increasingly relying on (or expecting to rely on) remote access to data, similar in spirit to OPeNDAP but at a lower (and faster) level. Obviously there may be technology and drivers also in industry, like "big data" and sensor networks, etc.

(This is the part of the writeup which concerns T4.3)

# Data centre implementation

## Fabric

cloud, co-location, modular data centre - container, own data centre – economy of scale (but this needs co-locating),

"Modular" fabric: Containers, Green Cube [http://www.e3c.eu/en/projects/]

[Use of SDN, SDS (see below)?]

Physical implementation of data centre – some aspects would be of interest to users of the data centre, specifically those pertaining to availability and reliability of services. Are there redundant network paths with sufficient bandwidth, are services on UPS and/or generator backup. Other obvious parameters include the number of available tape drives, and whether they are shared with other users.

Support – unusual problems will require support of IT staff, so whether it is supported 24/7 or only in office hours is relevant. For example, if a lot of data is required to be recalled from tape, it is sometimes better to ask operators to help.

## “Big Data”

Building on “big data” work to implement exascale data centres – how much is potentially useful and readily reusable?

CEPH, HDFS

Note we are focusing on the storage, but some processing is also required, e.g. for fixity, or foreseen as a requirement (e.g. preprocessing).

## Costing

Cost models – compare Terena TF-Storage work (which is public); derivations of costings based on models (Appendix)

## Storing and Encoding Data

### Introduction and Basics

The premise of RAID – Redundant Array of Inexpensive Disks – is that one builds disk storage out of arrays of disks, and can then afford to lose a disk and all the data on it, because the array has sufficient resilience in the encoding and storage of the data that the data can be recovered. Alternatively (or additionally), the encoding/storage is used to speed up the writing and/or reading, by *striping* data across disks; multiple streams can read the data off disks as the data is encoded across multiple disks, and the data can be reassembled – this performance increase makes sense as long as the I/O rate to disk is lower than that of the reassembly operation and subsequent memory/network transfer.

Traditional RAID has been widely used for a long time; indeed disk servers usually come with hardware RAID controllers in order to ensure good performance on the RAID encoding. RAID is classified by level, according to how many disks are required to store a number of bits and how the "extra" bits are stored – the extra bits could be simple replica ("mirror") bits (RAID1), a simple parity bit (RAID3), or something more sophisticated like error correcting codes (ECC). Commonly used RAID levels are 0 (striping), 1 (mirror), 5 (blocks are striped, with parity, and parity is distributed), 6 (like 5 but with two parity bits), or 10 (blocks are striped, then mirrored); there are other levels but they are less commonly used. The core design considerations that affect the choice are the desired redundancy level, performance in reads and writes, overheads (as in how many extra disks are needed), and of course costs.

If N bits are stored with additional M bits of parity (or more generally error correcting bits), storage people tend to use the short-hand notation N+M[[1]](#footnote-1); mathematicians working with error correcting codes look more generally at how many bit errors the code can *detect* and how many it can *correct*. In the storage case, detecting bit errors is generally less useful, since the data would be checksummed at a higher (typically file) level; however, it could be useful in detecting which block got corrupted – as opposed to saying "there is an error in at least one block somewhere in this file which is made out of one million blocks." If the data has been sent across a communications channel, then it makes sense to ask to have only the one (or few) block(s) re-sent that had errors, as opposed to resending the whole file.

In this discussion we have talked about blocks, rather than files. Most filesystems split files into blocks (padding the last block out to be a full block size), and when managing RAID for example, it is advantageous to do so with blocks rather than files, so they are all at the same size. Indeed, blocks can improve performance since blocks of the same file can be read in parallel if they are on different disks. File systems implemented at higher levels such as HDFS or CEPH also use blocks as the unit of storage; in this case it does not make sense to use RAID as the block management is implemented at the level of the distributed filesystem, as opposed to the disk server's operating system.

In considering the units of storage, RAID looks at placing and replicating blocks on disks. HDFS can replicate blocks across racks, so a failure within one rack which potentially affects more than one disk in the array (e.g. caused by vibration) or more than one server in the rack (a power failure) can be rebuilt at a higher level. Similarly, files are replicated across data centres to provide a high level of redundancy and/or high availability[[2]](#footnote-2) [XKCD].

### RAI\* - State of the Art

In some sense we can view RAID as a special case of error correcting codes – while the basic view looks at the "extra" number of bits (i.e. disks) required, we also need to consider the capability of the code to detect and correct (bit) errors, as well as how the "extra" bits are kept. We need to look at how expensive the code is computationally – a parity is a simple XOR, but more sophisticated codes require more mathematics and hence more calculations. For example, the popular BCH codes require some mathematics to set up and select, but once chosen, the syndrome decoding makes them relatively easy to decode.

Other practical things to consider include the time to rebuild upon failure, and possibly the bandwidth needed to rebuild upon failure, which is the subject of current research. If we aim to built RAIT – redundancy across tapes – we also need to consider the read and write streams: if N tapes need to be mounted (in the N+M notation from above) just to read/write a single data block, the operation can get quite expensive; also the "extra" (non-data) bits need to be calculated and temporarily stored somewhere while the encoded blocks are being written (unless one has N+M drives available for simultaneous writing, or one recalculates the encoding N+M times, once for each tape.) For N>2, this does not seem practicable – however, we shall consider the options below.

Running code near the data, workflows

RAIT vs replication vs media costs, #drives and usage, level of encoding (e.g. classic 4+3 BCH code) and features of the code (detecting bit flips or other errors (à la checksum), correcting for bit loss – NCSA have used 8+2), mechanism for file recall. Rebuilding on data loss, bandwidth required for rebuild (and time) - and probability of incurring another loss during rebuild time. Also, how data sets get encoded and written to allow the most efficient recall – tape families, tarring files, reducing write time to tape (migration, resiliency) may affect packing order for datasets that take a long time to come in, cf. DLS - trade-off between time spent on disk waiting for additional data vs need to get data onto tape ASAP. Can be modelled through risks of loss of data on a disk server, and mitigated through replicating or resiliencing the disk storage.

Purpose of RAIT/replication - is it done for resiliency, or for increased data write/read rates (parallel streams), cf. RAID0 vs RAID1 as the simplest example and the other encodings in between.

Example: 1TB written to tape, replicate the second out of the initial disk copy, so attaining resiliency earlier, vs writing the fully encoded set; time needed before the data can be safely gc'ed from disk. Dependency of number of available drives for writing. Where is the RAITing done? and how are the redundant/parity bits kept until an available drive/tape is ready for storing them? What is the overhead on RAIT in terms of caluclations – are they comparable to checksumming? (Parity is just XOR but more complicated codes would be more complicated.) Is the most efficient implementation just to use HPSS, or to develop a new version which needs to be supported. Can customers be assured that we can recall the data from tape even if software ceases to be supported? is vendor lock-in an issue? (cf. CleverSafe encrypting data at rest, which our customers didn't like, but is now being discussed in the context of cloud data encryption at rest)

Using RAIT to optimise recall, as not the whole set needs to be mounted, so you could choose those tapes that held other relevant data over tapes that just hold the parity. [Can this be tested further – is this a research project?!]

Tiered storage, HSM, e.g. currently using disk/tape in models like D1T0 etc., but could also be other flavours of storage, e.g. NVRAM, SSD, cf. SAGE project.

Federated data service, e.g. cloud replication, high availability, and replication in iRODS (Indigo/Alloy), EUDAT, WLCG, Indigo Datacloud, commercial replication (NetApp, WOS/DDN), ESiWACE T4.2. StorageD and ET used for climate data at STFC/CEDA. Different systems are used at partner sites - should/can a single tape solution be applicable to all sites/needs/ - difference in applications, whether it is a working repository, backup, replication, archive, access methods and patterns. Difference between ESiWACE partner sites - and/or sites studied – ECMWF different data volumes and software stack.

How differences between sites can be measured – types of data, types of infrastructure, capacity, types of users and applications supported – Met Office through to ECMWF, NCAR, GDFL. NCSA, NERSC.

Supporting existing data and interfaces at the front end – but may do Cunning Stuff™ at the backend. Is the infrastructure shared (fair share of the hardware), how do the different sharing models affect the use of the data centre by different customers/users. E.g. tape drives being shared and used opportunistically (or in an emergency), or dedicated – where is the scale where you don't need to, or cannot afford to, share. Toward the multi-exabyte, need to initially share but your data volumes are growing and eventually you need/want dedicated resources – planning for growth both in the data centre and on the customer end – from data store to applications (future proof - like using 64 bit addressing, IPv6, Unix time overflow in 2035 or thereabouts).

Needs to be integrated into the lower level storage (storage fabric, networks) - which test results are available – are there any tests we should do in ESiWACE? Should we do simulations/models and/or practical tests on the production infrastructure?

Power and greenness of data centre, location – gov't/EU requirement, save twice by reducing CPU energy. Ratio of disk to tape – and ratio of number of drives to number of tapes/slots (e.g. 1:500, RAL has about 1:200) – obviously depends on the use of the tape store. Modelling worst case scenario – high priority recall of large volumes (low likelihood, high impact) vs ongoing “random” read/write activity (high likelihood, low impact). [is this like a risk model – would further risk modelling be useful?]

Add numbers on disk server wattage – powering up a drive, disk server. MAID is more energy efficient, but drives also consume more current on spinning up (typ. 3x), and could have higher failure rates on being powered up and down intermittently . Essential to scaling that data consumes no power at rest.

Meeting the right IO rate, both to/from tape - reads/write, and to/from applications. Cf. PANASAS and

### Tape

The "death of tape" as storage media has been predicted many times, but tape still provides a reliable and cost-effective way of archiving data. In comparison to disk, SSD, and NVRAM, tape still provides the cheaper option to store data – when including the purchasing costs of the media (and depreciation) and the operating cost of keeping data accessible (electricity, cooling, operators, etc.) Tape does require a tape library which is a significant investment but is often easily extensible (more drives, more tape slots), and even disk requires investment in disk servers and racks.

<Table typical costs of tape/disks capex/opex as of 2nd half 2016>

As tapes are physical media which are written sequentially, data is best accessed in the same way: to get a single file from the tape, the robot will need to fetch the tape and insert it into a vacant tape drive, the drive will need to wind forward to the location on the tape and read it – so getting data off the tape is obviously more effective if data is read back sequentially. Random (scattered) access across tapes can be much more inefficient – by several orders of magnitude than an optimised recall – in comparison, SSD and NVRAM have no moving parts and are quite well suited to random access (but also have much lower capacity than tape). Hard disks also have scatter issues but a lower seek time than tape.

It follows that when writing data to tape, it is best to write data together that belong together, e.g. data from a backup or data from the same dataset – data that is likely to be recalled together. Similarly, when large volumes of data are recalled from tape, it is essential to order the recall requests so data found on the same tape are recalled in the same read operation, i.e. while the tape is mounted.

In comparison, WLCG where tape storage is provided only at the Tier 1 (and Tier 0) sites, and tape is used mainly for archiving, individual users are banned from Tier 1s.

Another approach is to queue the recall request, so the tape system has a set of recall requests; when enough requests have accumulated, the tape is mounted and read – if the whole tape is read back, data can then be cached locally on disk in case additional related read requests come in.

~250 MB/s to tape (2016),

Splitting a single dataset across tapes will in principle allow the data to be recalled faster, as tape drives can stream the dataset in parallel, and the data can be reassembled if required – this assumes that there is enough bandwidth for the parallel streams and the receiving system(s) have sufficient capacity to receive all the streams. For example, when transferring data with GridFTP, it is not uncommon to set the default number of parallel streams to 10 – this is to make more efficient use of the networks between the sender and the receiver and in practice the optimal number of streams is set experimentally. If using parallel reads from tape, we of course also assume that a sufficient number of tape drives is available; otherwise some tapes would have to be queued to be read.

<plot GridFTP with different # streams>

System is not expected to handle a single

Is the system built to optimise a single workflow, or to optimise the aggregate throughput – scheduling, also the number of drives reading/writing at a given time; also dependent on the data access patterns.

Access patterns generalise to all of the HSM, e.g. caching hot files, multiple reads cached at higher level (anything from SAGE?) Data model can affect volumes and configuration. Julian's disciple and SpectraLogic simulator – can we develop a simple model. Vendor supported/supplied access methods may reduce the TCO.

Data on tape can be accessed as files but can also be stored and/or accessed as blocks, depending on system (TODO: compare block storage like HDFS, HPSS), - this varies the volume of data that needs to be recalled, resp. written.

Causes of bit loss/errors/flips. Demagnetisation - time before data needs to be rewritten – 5 yrs for disk, 20 yrs for tape (repack). Loss of media due to wear, mechanical problems, etc. - number of times media can be read (resp. written), before it can be considered unreliable.

Tape data loss, cf study done for DLS; recovery rates of files on tapes being sent off for recovery. How does RAIT reduce the wear on tapes, and hence improve (possibly) costing, or lifetime. Higher costs if tape lifetime is (say) six years, before you need to read the data and rewrite it; not affected for a specific number of tape mounts/reads.

WORM vs WORN – tape lifetimes. How does RAIT affect these two extremes? In WORN, saves you tape media for potentially the same resilience (depending on the model for the loss of a tape, because the tape might be doing something else, too). In WORM, can lead to increased restore speed due to parallelisation assuming that enough drives are available. Can also spread the wear across more tapes, but you'd still need N tapes (for N+M) for each recall rather than 1 tape for each recall of the file. In particular, if your robot can hold only 64 drives, then you get 64/N simultaneous recalls

RAIT: 10TB/tape, 80 TB on 8 tapes, 16 tapes with direct duplicates, 10 for 8+2 encoded. Whole tape recall vs individual scheduling, chaotic block scheduling – how does it affect the lifetime of the media?

What if only a part of the file/dataset needs to be recalled? Can it be read from 1 tape or does it need 8? In the first extreme case, we have each byte split with a bit on each tape, so eight tape reads are needed just to recover any single byte. At the other extreme, a whole tape is a block, so the file is split sequentially into 8 tapes à 10TB per tape, and with two parity tapes; any subset of the file can be recovered from the single tape it is on (like RAIT0) (except in the obvious case where it straddles the boundary between two tapes), and then adding two parity tapes, like the RAIT3 example below.

However, most files are not 80 TB, so to make use of this extreme, files will need to be aggregated in a way which would allow individual files to be recovered from a part of the block (which is one tape in our example.)

Note PANASAS will replicate for small files and erasure code for larger files.

Are there advantages to choosing N (e.g. for RAIT3, see below) where N tapes can be stacked together? Is it easier to read the tapes when they are stacked together? Read patterns, as in either reading the whole tape from start to finish, as opposed to reading a segment – 12 hours to read 10 TB.

Scheduling reads: D1Tn is always preferable, but if you have D0Tn, scheduling is needed to ensure the efficient read of tape. Example: if the interface makes it easier to ask for a full (gigantic) dataset even if only a subset of it is actually needed. If lots of small objects are needed, they need to be queued and then reordered so objects on the same tape get recalled together (example – StorageD recall exercise).

So 8+2 is probably horrible for tape storage given the #drives to get stuff back, but if the block was the whole tape (or you needed to read blocks across the whole tape), i.e. the 80 TB dataset example, then 8+2 might be more efficient (or less inefficient) - check? Application workflow, or the disk caching in front: will define the volume of data needed to be recalled. Which is the optimal RAIT code, if any?

RAIT modes:

* RAIT0: split files, as illustrated above.
* RAIT1: mirror
* RAIT3: like RAIT0, but with parity tapes added; leave parity tapes alone till they're needed, only use the N tapes for reading. Is there a computational advantage to not doing the syndrome decoding by using the first eight non-parity tapes? First N tapes are worn evenly at the rate of 1/1.
* RAIT5/6: any N tapes can read, so distribute read load across N+M tapes – any N tapes sufficient for reading. If chosen properly, all N+M tapes are worn evenly at the rate of N/(M+N)
* RAIT10/01:

For each of these, consider the efficiency and safety of writing (how long does it take to write the whole file to tape, given how many drives are available), and how quickly can the whole file, or a segment of the file, be read back. How long would it take to restore if a tape breaks? Given even probabilities (or probability increasing with wear), how likely is the overall loss of data?

(Do we have enough data to say anything about tape loss statistics?) If a tape is lost, is it more likely to lose M+1 tapes given that you need to read N tapes out of the set to rebuild your data? At what stage would you rebuild the whole dataset on fresh tapes, given general wear of tapes or loss of a single tape?

Block level/block sizes – storing a

Access patterns/Latency is low enough – how much can be cached on disk, and how do applications ask for data – e.g. srmBringOnline().

Ratio of drives to tapes – efficiency, each library has a maximal number of drives they can support. Typical ratios are

Stochastic modelling – vs worst case (pathological)

Model for recalling – and for the cost. 10^6 for the library, 10^4 for the drive, 10^2 for the tape. When is a new library required?

Model for loss of data.

Model for ratio of disk cache capacity to tape capacity for HSM-use, assuming random access patterns.

Tape families – generic tape pool; aggregator giving hints on data placement – by namespace, or hints as in SRM space token. One spot directory = tape family. Each spot generally needs its data to be co-located, in a single file/tarball/aggregation/tape/tray/rail/library.

### Beyond RAI\* - erasure codes

As mentioned above, erasure codes (error correcting codes) can provide equivalent or better durability than RAID, yet have more manageable recovery time (depending on how they are implemented)[[3]](#footnote-3)

## Packaging and data placement

### SotA(?)

Files can, of course, be of different sizes, ranging from zero bytes to in principle the maximum allowed by the filesystem implementation. However, storing and recalling files directly can often be impractical, one reason being that the size may not be optimal for the storage medium. The old DOS FAT filesystems famously stored data in blocks, so a one byte file could typically occupy a 32KiB block, leading to lots of wasted space on the disk – this, in turn, leads to accounting problems, because the user would argue that one byte of space is used, whereas the system could claim that 32KiB was used. More modern filesystems also use blocks, typically 1KiB to 64KiB. Early Linux filesystems had a fixed number of blocks, indexed by inodes, the number being set when the filesystem was created and set as a trade-off between the expected number of inodes (typically one for each file/directory), performance (viz., avoiding file fragmentation as well as avoiding scattering seeks across metadata and data), and the space occupied by the inodes themselves. A modern filesystem such as ext4 introduces "flexible block groups" and "lazy block group initialisation" and many other features to allow a single filesystem to grow to hundreds of TiB and yet manage files efficiently and get good general performance on reads and writes. Higher level features such as LVM enable more dynamic management of the filesystem, e.g. by adding and removing physical disks to the filesystem; more generally, storage systems such as DPM and iRODS manage different storage resources at a level above the filesystem and let administrators implement data policies.

For tape storage, the equivalent problem is the overhead of writing a file to tape, particularly if a file header is written with the file. At the other end of the scale, a file may be larger than the size of the tape, so will need to be split across tapes.

(LTFS?)

(Duplication?) The bottom line is that, just like the filesystem tries to keep the files' blocks together, it is often useful to aggregate files into – let's call them collections. It makes it easier to manage files that "belong" together (part of the same application or dataset) and it makes it easier to move and store the aggregation as opposed to doing it one file at a time. The cost of this, though, is that the aggregation needs to be built, resp. unpacked, which, unless it is being streamed, requires double the storage space, albeit only temporarily. This takes extra space and time, so the cost must be outweighed by the improved efficiency of storing/recalling and transferring files together. Indeed, aggregations may make sense at many different levels: files in the same dataset should be aggregated into a larger collection (which is itself, technically, a file); the size of a collection should be set to one which can be efficiently managed by the data store – and conversely, a file which is larger than the collection size will be split into chunks which are themselves stored as collections. Collections that belong together should then be placed together on the same tape so a recall of all collections are likely to be (optimised to) read from the tape during the same tape mount and read operation. Going further, tapes that "belong" together should be placed in the same tray (if trays are used), or if enough drives are available could be placed in different locations in the tape library so they can be read simultaneously (see the RAIT discussion above.)

Data placement study at RAL – covers CEDA data

(HDFS, HPSS, GPFS)

In practice, a mix-and-match model may be necessary, which is able to implement efficient storage, transfer, and recall of the data across a range of file sizes, access/use patterns, and availability and durability parameters. When making data available in a tiered storage hierarchy, a typical strategy will make data available in the fastest tier (e.g. SSD) when it is accessed, and then moving it to slower tiers as it ages, particularly if it is not accessed any more

## Performance and optimisation

Analysing data logs: how they are analysed and the results are brought to

## Data Processing

One of the common themes of research in storage has been to bring specific types of *processing* closer to the storage. Obvious first steps in storage-side processing is checking basic data integrity (“did we get all the bytes, and in the right order?” as well as updating metadata systems (“we have a new file”). The next steps were to automate metadata extraction in a file-type dependent way, so a process would recognise the file type and apply file-specific checks on its *structural* integrity (“is this a well-formed HDF5/PDF/Word document”). These types of checks are widely used in production infrastructures.

The next obvious extension to this was twofold: one proposed direction would bring higher level features closer to the device, to remove the load on filesystems. For example, in conventional magnetic hard drives one originally had to know the number of cylinders, sectors, and so on, but LBA vastly simplified supporting hard drives without having to reconfigure the operating system (or file system), and in a similar vein, hard drives can do their own checking and management of bad blocks. [It may be worth investigating this further, as LFTS could provide similar benefits for tape? or are those benefits irrelevant for the purposes we are discussing here?]

A research extension by SNIA proposed drives that provided direct object storage with metadata, thus bringing some of the work required by the filesystem into the storage device; the flip side is that the interface would not be full POSIX but more like an object store. This particular research, however, never came to production, and we must conclude that there were not sufficient requirements for it to have a market.

The other direction was in storage management services that would implement programmable storage side workflows, such as iRODS. Generally microservices would need to be added in order to support new file types and change the rules, but the system would be customisable to provide relatively simple workflows which were used to manage metadata but could be extended to do server side processing.

Grids provide similar features but in a different way, as there is technically no difference between storage-side processing and user-side processing: they are done using the same mechanisms, but essentially in different data centres. WLCG operates a tiered model where data is generated at a Tier 0 centre, copied to Tier 1 centres whence it is copied to Tier 2s (these should not be confused with data centre tiers which are essentially the other way around.) Raw processing of data – what we would call storage side processing – is done at the Tier 0 and 1s, with no end user access at all. End user analysis of data would happen at Tier 2s and 3s, and data which need archiving would be copied back to the local T1 [diagram]

An alternative approach is to treat the storage as a basic service – like an object store, typically – and let higher level workflow engines manage the writing and processing of the data as a part of a (loosely coupled) workflow. This approach would also work well provided it can get notifications back from the store. Indeed, CASTOR2 was originally written on top of LSF, but LSF was removed (around ~2010?) in favour of a bespoke “transfer manager” when it was found that original versions of CASTOR didn’t scale well with the WLCG workloads[[4]](#footnote-4).

A related approach is to make it easier to develop datastore-aware HPC applications, as described in section XXX. As an example of the state of the art, SAVU [Wadeson] is an example of a modern science application which provides a framework for a specific highly specialised task (in this case tomography), yet is relatively easily extensible by researchers through plugins, so researchers need not have specialised knowledge about the data storage [todo (maybe): compare how SAVU aligns with the storage-awareness proposed here?, and how it might extend to climate tools such as CDO?!]

## Staging and Caching

RAL conducted a data placement study of CEDA data.

State of the art: ERADAT, the BNL extension of (90PB) HPSS [Yu]. The main question is whether every recall can *always* be automated or at which stage would it need operator intervention in order to me maximally efficient?

State of the art – NERSC CORI (2017?) [Dosanjh], NVRAM burst buffers; 28 PB secondary disk (Lustre) supporting data placement “close” to the CPU; thus a further staging process is required anyway (cf. diagram). Also JASMIN’s PANASAS with 15PB (using ET for temporarily staging data in order to free up fast disk)

[NUMA?]

Is it necessary to generate models for this? Is it useful? For example, the allocation of drives to each individual user community, and optionally prioritising recalls.

Could the same be applied for other types of file operations that are scheduled, such as replication and integrity checking. It would be useful to do opportunistic operations: if a tape (resp. collection) is mounted (resp. fetched/unpacked), take the opportunity to act on other collections (files/objects), instead of waiting for their scheduled operation.

# Crystal Ball

Previous predictions – how well did they perform?

# Scalability – towards exabytes

This section looks at scaling a data centre towards the Exabyte mainly with today’s (2017) or tomorrow’s (next year or two) technology. The section Future below attempts to look further ahead. If we can build an Exabyte data centre today, it will be smaller tomorrow – if we cannot afford to build one today, at least we can highlight the barriers to building one, and analyse in the Future section when we might overcome these barriers.

## Scaling through hybrid storage models

Hybrid data centres – combining cloud resources with traditional data centres. Allocating resources on demand for science use: requires interfaces and transfer tools for either automated transfers or done deliberately by the user (e.g. selecting data sets for analysis in the cloud)

One of the requirements for (exa)scalability is that data consumes no power when at rest. At present, a few technologies support this, e.g. SCM, MAID, tape.

## Data models for scaling (including the disk cache)

Automated replication will enable data to be replicated between sites. Solutions for this are available at many levels: rsync has of course been used for many years, but may not scale well as it requires a secure channel across the WAN, and typically ssh is used.

Grids can achieve scalability (such as WLCG’s XXXPB) by enforcing controlled data models, so it is worth looking at those in more detail.

Climate as Hyperscale – going beyond “enterprise storage” (5,000+ servers) – use of SDS for resilience ; use of JBOD/JBOF and mixed media, e.g. active vs passive data models. Benefits from dovetailing with commercial hyperscale customers (Google, Azure, etc.), benefiting from their clout with vendors

## Scalability – the Exabyte data centre

Other than storage (next section), we need to consider the data centre architecture when scaling (section XXX) – as well as the directions we scale in. Let us begin with the directions: among the classical “data Vs”, volume and velocity are the most obvious but also veracity. Other directions could include number of concurrent users and concurrent transfers, the latter particularly when users share bandwidth with replication services.

## Scalability – storage

Of all technologies available today – magnetic hard drives and SMR, SCM, SSDs, tape currently provides the highest durability of all media, and the most cost effective storage both in terms of procurement and running costs. Obvious disadvantages of tape is that data is nearline or offline, so there is an access latency; also, tapes deal consistently less well with small files[[5]](#footnote-5), and files are stored sequentially. Less obvious disadvantages include the need to write (non-WORN) data to aid the future recall of data, so data that is recalled can be read efficiently off each tape

### Collections

It is worth investigating further the small files problem and the need to store data together that is likely to be recalled together, because they share an obvious solution. Related data ‘units’ (typically files) are put together in ‘collections’ (using tar, or simply concatenating them) and the collections are then archived. (The analogous thing could be happening at a higher level, e.g. with HDF5 data which itself may contain different data objects.) Obviously the archive then needs to keep metadata for the individual units (AIU), particularly when the metadata is required for the extraction, as well as for the collections (AIC). A disadvantage of this approach is the whole collection is recalled from tape even if only a part of it is actually required – this is usually not a huge disadvantage because the tape is mounted and read anyway, regardless of how much data is read by it – it is mainly a disadvantage if the collection is very much larger than the required data. A related disadvantage is that recalling data from a collection requires temporarily up to double the disk cache space, because the whole collection is recalled from tape onto work space on a disk, and then has to be unpacked (this problem can of course be mitigated if one unpacks only the data that is actually needed from the collection.) It follows that collections should not be too large – they would become unwieldy, incurring too much overhead when only a small section of them are required – nor should they be too small, as the larger number of them would tend to limit the advantage of having them in the first place. A good compromise size is not less than ten gigabytes, and perhaps not more than a few hundred; at tape read speeds of, say, 330MB/s, a gigabyte is read in three seconds, and once the tape is mounted and has started reading the first gigabyte, reading another gigabyte or two is not huge overhead, reading 10 GB will take 30 seconds which should be seen against a time to mount and seek which is likely to be 10 mins and upward (up to an hour for a busy tape system). 10GB can readily be stored on most disk servers and unpacked. Also, if a tape has a capacity of 10TB[[6]](#footnote-6), it will hold 1000 such collections which seems like a reasonable compromise.

The converse is that a file may be too large to fit onto a tape, and some tape archive systems do not cope with this; the problem is then solved by having the collection store a part (chunk) of the file. Splitting a file into chunks of possibly varying size is helpful also when the system is trying to pack tapes efficiently, because there may be a tape with space left but not enough to hold the whole file, and packing would obviously be more efficient if this space could be used. The disadvantage to this approach is that the file is split, and to restore the whole file, it becomes necessary of course to read all the tapes that hold chunks of the file, and then have enough space to reassemble the whole file.

It can be seen that unless higher level services provide the collection feature, the datastore service needs to provide it, in order to provide efficient tape storage. It follows that metadata need to be kept both for the individual data units (AIU) and the collection (AIC), and even if it is implemented internally it may make sense to expose the collection feature to the users in order that they can transfer their data efficiently – as otherwise they would unpack a single collection into, say, a million files, which then need to be transferred individually – it would clearly make sense to have the user transfer the collection itself, and then unpack it locally. Indeed most systems that allow users to select individual files from an archive will usually package (and compress) the files before transfer (typically a zip file, so Windows-based users can unpack them.) In this case, the archive has extracted the required files and placed them into a temporary directory, then created the zip file for the user and had the user transfer/download the zip file.

The discussion of collections may remind the reader of the discussion of stripes in section XXX-RAID. If there is a particular stripe/block size in the underlying storage, it will make sense to limit a collection to fit into a single block if the block size is not too small, of course, or otherwise to align with an integer number of blocks. See Diagram XXX (HSM+cache vs unpacking data)

One obvious alternative to collections is to guarantee that files always have a suitable size, and, if they occur, deal with small files in a different way. Small files could be dealt with by storing them on disk or other non-tape media, and achieving a durability comparable to tape by replicating them – after all, they are small, and replication is not too onerous in terms of volume. As for creating larger files without collections, one example is mentioned above, HDF5 – a HDF5 file can itself contain different data types, so data can be aggregated into larger HDF5 files which can then be ingested efficiently into the tapestore. The advantage to this approach, compared with collections, is that it clearly obviates this extra level of complexity, because there is only a need for the tapestore to manage files. The disadvantage is that one loses control over the size of the file: first, data creators have to be trusted to only store files once they have reached an appropriate size (and then not change them), and secondly, collections can be designed to fit block or stripe sizes (if used, as mentioned above), to improve performance – it is less obvious that the same performance gains are achieved with files that span a range of “appropriate” sizes. [Would parallel HDF5 benefit? E.g. if more data is stored in the file and parallel access is used to access each part. Also, climate data may have more regularity – e.g. similarly sized data blocks?]

Another alternative to the collection feature is of course to not store data on tape, but to leave it on slower disk or MAID systems. While this is generally less scalable, less durable, and less cost effective than tape (see next section), it is a valid solution for some types of data. One would still ‘zip up’ files for user transfer, but need not worry about storing individual files, however small. Typically such services are run (at scale) on distributed filesystems such as CephFS, HDFS, or Lustre, and are often part of the service provisioning of a data centre. The PANASAS system run for JASMIN currently has a capacity around 15PB and is one of the largest of its kind in the world (that we know about.) It may make sense to operate such an infrastructure in front of a tape archive (such as with HPSS) or independently, or specifically as storage services for a compute cluster (such as the JASMIN case.) As mentioned in the introduction, we expect a future data centre to serve the joint roles of archive and working repository, and a combination of infrastructure fabrics will be needed to meet those requirements.

## Distributed Filesystems

Distributed filesystems also scale fairly well – Lawrence Livermore report a 55PB Lustre system, NERSC has, in addition to HPSS, a Lustre system

However, to achieve good scalability when a HPC system accesses a distributed filesystem such as Lustre, it will help to be “aware” of the underlying storage and how it is best accessed. For example, NERSC have found that I/O performs better on Lustre when it is aligned to Lustres stripes.

### HSM

Studies of enterprise data repositories (ref) show that infrequently accessed data (WORSE) is often kept on fast and expensive storage, and more cost effective storage could be achieved through a hierarchical model where WORSE data is held in slower and more cost effective storage – and with tape as the slowest and most cost effective[[7]](#footnote-7). HSMs have been around for decades

Tape can already provide Exabyte scale libraries: density varies between hardware vendors, and some assume in their promotional material that data compresses by a factor 2.5. If we assume no compression at all and that each tape/cartridge stores 12.5TB (as of 2017), it takes 80 tapes to store 1PB and 80,000 tapes for 1EB (for, obviously, single tape copies, but compression and expansion for erasure code could offset each other). Assuming a cost of €250 per tape for Enterprise class tapes (which have the lowest BER and could store data for 30 years[[8]](#footnote-8)), it would cost €20M for the media. Tapes, of course, consume no power when they are not being used, and WORN/WORSE data could be stored outside of the tape libraries (offline, requiring operator intervention to read), thus reducing the need for tape libraries. In other words, for WORN/WORSE data, the data centre will already have operators and tape libraries and drives, but will not need additional operators and drives to scale: it is sufficient to buy the extra media, write them, and store the tapes offline, as long as adequate storage space is available – with physical security, fire/flood protection, with temperature and humidity suitable for long term (decades) storage. Scalability is achieved simply by adding media. In contrast, adding disks requires disk servers, even for MAID (spin-down) disk storage.

Operating a tape archive includes three types of routine operations on tapes beyond data access.

* It is good practice to occasionally “wake up” the data to checksum it; this is part of preservation services, but is also good practice on other data.
* Repacking tapes is occasionally required when data can be deleted or overwritten. In general, for primary and derived scientific datasets, we can assume WORM/WORO/WORSE, i.e. write once – however, in practice there are cases where data is cleaned up and deleted, and, eventually tapes have “holes” and will need repacking. Repacking is typically done by reading all the data off tape into a disk server, and then pack the data back together. Obviously, attention should be paid to data “belonging” together (to aid efficient recall of data, that they get read in the same read operation) – see also the section on Collections above. Of course once data is recalled for repack, it should also be checksummed (at least the collection).
* The third case is where data is migrated, from one media type to another. In a tapestore this happens typically when new media with sufficiently higher write densities become available; the old data is then written to new media and, over all, frees up space for growth. Migration is a thus a strategy for sustaining growth within a data centre (as opposed to expanding it, physically), but has the obvious disadvantage of incurring the extra media cost. In particular newer higher-density media are often more expensive, particularly enterprise class. A data centre’s migration strategy will typically include the cost of media migration in the pricing of its services (save, perhaps, for data stored only for a short period of time), and may also skip a generation in order to be more cost effective (particularly if it is necessary to buy new tape drives as well, although may be possible to offset the costs by selling the old drives) It is not unusual for a tapestore to support multiple tape media with different densiites (and ages) at the same time. Finally, a migration exercise will, of course, also perform both repacking and checksumming.

# Current areas of relevant research

The purpose of this section is to describe active areas of research which are (or may be) relevant to the implementation of an exascale data centre, or to highlight directions that need further investigation, or simply research which exist but has not been brought to bear on a sufficient scale in current data centre implementations.

* Minimising rebuild times for erasure coded storage (specifically, minimising the inter-node transfer)
  + Compare with Rapid RAID Rebuild? – mostly interesting during rebuild times?
* Is there any research in preparing data for HPC access (cf. Features)? If there isn’t, can/should we do it?
* Automating the data lifecycle
  + Machine readable SLAs (cf. FP7 projects)
  + Automating the application of data policies and licences (apply research over the past >10y)
* Improving building efficient science applications (maybe?)
  + On the data centre end, by optimising for access, and providing different storage for different types of data
  + On the application/scheduler end, by making it easier to do the right thing – scientists don’t always understand data centres and HPC, and
* Checkpointing application state has been supported for a long time, earliest in research operating systems like XXX which would do so automatically. Commonly used interfaces like Java persistence frameworks give a SQL-like definition of data which is serialised in applications. More generally, many programming languages support persistence frameworks and some support sufficient introspection to be able to implement persistence transparently (Python, Julia, LISP through MOP). More recent research done at BSC (ref?) has looked at developing in-application data structures which are transparently fetched and persisted from a datastore [XXX – check status]
* Depopulating storage arrays (JBOD, JBOF, JBOT…),
* Tape rebuild as a background operation (as with CASTOR Repack), compare to automated maintenance task such as disk cache CANBEMIGR and GC.
* Hints for data recall – if redundant blocks available, optimise for low latency recall, slower blocks needn’t bother once the faster ones have got enough blocks to reconstruct data. (Suitable for models) - as opposed to striping where the slowest stripe sets the pace.
  + Other hints on stage-in could be for expected sequence – how does this marry with hints to abandon a stage-in of a slow block?
* Does the disk server models (for simulations and cost modelling etc) extend to tape – use of learning algorithms to optimise placement – or is it more like combinatorial optimisation?

# Future

## Pressure/drivers for future directions

for climate it would be volume and "velocity" of data; perhaps veracity (i.e. QA).

Obviously other areas have drivers for data centre futures as well, particularly the "big science" and IoT scenarios.

## Technology

### HSM, storage, and caching

Tape store🡪slow disk🡪fast “disk”🡪{application cache, SCM}🡪RAM🡪CPU cache

(note parallel data unpacking from collections to arrays etc., cf. diagram)

### Tape

Scalability of tape – e.g. expecting 40% increase in volume per year; 20% increase in access speed, so access density will decrease, but not on the rate of HDDs. Even as of 2016 there are 180TB-200TB tape cartridges in the lab. In separate developments, the already high durability (a BER of 1019) of tape is improving.

Will HDDs disappear? Will SMR fill a gap for high density HDD – would suit WO\* use cases, but is the increment in capacity sufficient? Is MAID relevant or will it be obsoleted by, say, SSD development?

## Usage Models

## FET

# References

http://xkcd.org/1737/

http://xkcd.org/1506/

http://xkcd.org/908/

1610.04513

1610.04498

[DCC] http://www.dcc.ac.uk/sites/default/files/documents/publications/DCCLifecycle.pdf

S. Dosanjh: Exascale and Extreme Data Science at NERSC, HEPiX, Fall 2016, <https://indico.cern.ch/event/531810/contributions/2289875/>

B. Lavoie: *The Open Archival Information System (OAIS) Reference Model*, 2nd ed. (2014), OCLC

T. Quan,J Botts, L Gerhardt, D Jacobsen, D Paul, S Canon, W Bhimji, D Bard, T Declerck: Running HEP Workloads on the NERSC HPC Systems, HEPiX Fall 2016, <https://indico.cern.ch/event/531810/contributions/2330787/>

N. Wadeson, M. Basham: *Savu: A Python-based, MPI Framework for Simultaneous Processing of Multiple, N-dimensional* *Large Tomography Datasets*, …

D. Yu, J Lauret: Effective Access to Massive Amounts of Tape-Resident Data; HEPiX (Fall) 2016; <https://indico.cern.ch/event/531810/contributions/2298920/>

Energy Sciences Network (ESNET): Science DMZ, <http://fasterdata.es.net/science-dmz/>

SAS *Statement on Auditing Standards* <http://www.aicpa.org/InterestAreas/InformationTechnology/Resources/Auditing/Standards/Pages/default.aspx>

# Appendix – Models

(use this space for models, e.g. probabilistic modelling of )

1. In particular, duplication could be written as 1+1, triplication as 1+2 [↑](#footnote-ref-1)
2. There have also been proposals and trials to replicate individual blocks across data centres in order to provide confidentiality of data at rest (no single data centre can read the data; it would require re-constituting blocks from a critical subset of data centres) – however, this work is less relevant to us here. [↑](#footnote-ref-2)
3. Many RAID levels are essentially special cases of error correcting codes, implemented at the whole-disk level as opposed to (say) the block level. [↑](#footnote-ref-3)
4. The problem was essentially that certain LHCb jobs would prestage about 20 files, and each prestage request would schedule a workflow of eight or so tasks [↑](#footnote-ref-4)
5. The small file problem is not unique to tape; one finds it also in CEPH, for example. [↑](#footnote-ref-5)
6. It doesn’t help that some tape media/library vendors *assume* a compression ratio of 2.5:1 in their marketing materials; thus the “one Exabyte” tape library may actually “only” hold 400PB of “native capacity.” It is akin to the terabyte/tebibyte discussion used for hard drives, only even more misleading. [↑](#footnote-ref-6)
7. (In the enterprise case, there would obviously be a significant initial cost if they were to procure a tape library, but there are cloud “solutions” also for tape storage, for BC/DR, and for archiving. [↑](#footnote-ref-7)
8. BNL reported accessing 15 year old tapes (9940B) and successfully accessing data from them. [↑](#footnote-ref-8)