Improving Performance of Tape Restore Request Scheduling in the Storage System dCache

Lea Morschel[1,*], *Krishnaveni* Chitrapu[4], *Vincent* Garonne[2], *Dmitry* Litvintsev[3], *Svenja* Meyer[1], *Paul* Millar[1], *Tigran* Mkrtchyan[1], *Albert* Rossi[3], and *Marina* Sahakyan[1]

[1]Deutsches Elektronen-Synchrotron DESY, Notkestr. 85, 22607 Hamburg, Germany
[2]Nordic e-Infrastructure Collaboration (NeIC), University of Oslo, Norway
[3]Fermi National Accelerator Laboratory, Batavia, USA
[4]National Supercomputer Center, Linköping University, Sweden

Abstract. Given the anticipated increase in the amount of scientific data, it is widely accepted that primarily disk based storage will become prohibitively expensive. Tape based storage, on the other hand, provides a viable and affordable solution for the ever increasing demand for storage space. Coupled with a disk caching layer that temporarily holds a small fraction of the total data volume to allow for low latency access, it turns tape based systems into active archival storage (write once, read many) that imposes additional demands on data flow optimization compared to traditional backup setups (write once, read never). In order to preserve the lifetime of tapes and minimize the inherently higher access latency, different tape usage strategies are being evaluated. As an important disk storage system for scientific data that transparently handles tape access, *dCache* is making efforts to evaluate its recall optimization potential and is introducing a proof-of-concept, high-level stage request scheduling component within its *SRM* implementation.

1 Introduction

In recent years the volume of scientific data that needs to be stored and processed is increasing towards the exascale level [1]. Hierarchical storage systems, which combine different storage technologies of varying characteristics, such as low latency hard disk drives as well as magnetic tape, and move data between locations based on specified access requirements, provide an economical solution to meet capacity and throughput requirements necessary to store and process this huge amount of data. The less expensive a technology is, the higher the access latency, which one attempts to minimize by smart access and caching strategies.

The *dCache* [2] software is an open-source distributed storage system that is widely used in the scientific community, particularly in the area of high energy physics. It is primarily a disk based file storage system that is dynamically scalable to hundreds of petabytes and capable of transparently migrating data to and from a connected tape storage system. Because tape storage was originally intended for long-term archival and backup purposes with rare large-scale recalls in case of emergencies or rare scheduled events, no optimization for recalling data from tape is currently implemented, relying instead on the capabilities of connected tape

*e-mail: lea.morschel@desy.de

storage systems. In order to meet the increasingly important need of efficiently recalling tape-resident data, we are hoping to improve this mechanism independently of storage and request pattern characteristics.

2 Background

In the field of high energy physics, a major exascale challenge is approaching with the planned upgrade of the *Large Hadron Colliger* (LHC) [3] at CERN to the high luminosity configuration (HL-LHC) in the second half of the 2020s [4]. The expected post-exponential growth in data requires infrastructures, tools and workflows that are able to handle these new challenges. Calculations have shown a large discrepancy between the resources available, assuming a flat-budget model, and the requirements for storage and computing power. Affordable storage space is even more problematic than the extra computational power needed [5], which is why the less expensive tape storage medium is being considered.

2.1 The ATLAS Data Carousel

One of the current efforts to use tape storage more intensively is the so-called *data carousel* concept [6], which is being investigated and implemented by the ATLAS collaboration [7]. The approach builds on existing hierarchical storage architecture of nearline tape storage behind a smaller layer of disk space. The goal is to enable large-scale recalls of tape-resident data while minimizing the loss in performance that is to be expected due to the robotic access and inherently sequential nature of tape storage. In order to minimize the tape access overhead while maximizing disk space utilization, the idea is to no longer allow random requests of files from tape, but instead to stage and process a sliding window of only a fixed percentage of the total data volume. After the data on disk has been processed, it is discarded and the next chunk is cycled in from tape.

The basic unit of analysis and therefore request by ATLAS campaigns is the so-called *dataset*, which is a set of files that logically belong together. This grouping information is known at the highest application layers, but currently not passed down to the storage elements explicitly, but rather as part of the file path. Because the file path is mutable, it is not interpreted semantically by the storage system. Because they are usually requested to be written to tape in approximately the same time period, it may accidentally result in grouping on tape. However, as they are likely flushed alongside other data for the same tape group and because several tape drives may be used for writing, this ordering is frequently lost and they are often distributed over several tapes. Having to retrieve small dataset fragments from many different tapes is highly inefficient, so efforts are being made to implement *smart writing* strategies in order to co-locate datasets written during future data-taking runs. The objective of this paper, on the other hand, is to propose how existing, potentially suboptimally located data on tape can be more efficiently accessed.

2.2 The dCache System and Its Tape Interaction

For their primary storage element, the majority of the WLCG [8] Tier 1 sites use dCache, an open-source distributed storage system for scientific data, which uses a microservices-like architecture to provide location-independent access to data. dCache is easy to integrate with other systems because it can communicate over several authentication and data-access protocols, and supports different qualities of data storage, including tertiary storage support, for which it is able to use disks as a caching layer [9]. The dCache system is able to move

disk-resident data to one or multiple connected Tertiary Storage Systems (TSS) and migrate it back to disk when necessary. It does not operate tape robots or drives itself and assumes that it is connected to an intelligent TSS which does. In most cases, the TSS in use is a tape system or migrating Hierarchical Storage Management system.

There are two different ways to interface dCache with a tape-connected TSS: via a pluggable driver API which is suitable for creating smart TSS connectivity, an example being *ENDIT* [10], a sub-system developed at NeIC, and via an external script that is provided as a reference implementation of the API, which is designed to be simple but versatile. In order to communicate with a TSS, the interface is limited to the operations `PUT`, returning a unique identifier for the saved file that is then stored within dCache, `GET` and `REMOVE`. The system will retry failed operations. When dCache interacts with a tape system, these operations are translated and forwarded to the respective back-end.

When restore requests arrive at a dCache instance, the space needed for storing the requested tape-resident files on disk is reserved before passing them on to the tape system back-end: if the reservable space is bigger than the requested, dCache won't submit all the requests to the TAPE system. Especially in the course of large-scale recall campaigns, this usually results in a backlog of requests because it will likely only be able to pass on a subset of all known requests at any point in time. Because dCache has no knowledge regarding file locations on tape, it currently does no reordering for the purposes of recall optimization and leaves that to the back-end. However, because the latter is only aware of a subset of requests at a time, this reordering is likely to be suboptimal.

3 Tape Storage Access Characteristics

In order to access a tape for reading or writing it needs to be retrieved from its physical storage location and inserted into a tape drive, then wound to the correct position or offset. Data storage and retrieval from automated tape libraries is handled by tape systems, which use hardware-side components that translate high-level instructions to hardware operations, but also provide functionality for high-level request queuing, reordering and an optional range of additional functionalities. There are several tape system software solutions with different approaches and ranges of features, with some of the most important ones in the scientific community being *CTA* [11], *Enstore* [12], *HPSS* [13] and *IBM Spectrum Protect* (TSM) [14].

Despite the sometimes large differences between them, tape systems share some fundamental behavioural characteristics of tape storage.

3.1 Recall Performance Markers

The main read performance bottlenecks of tape systems are the number of mounts as well as the on-tape seek time for required files [15]. Because tapes need to be physically retrieved, inserted into a tape drive and wound to the sought position, the access latency is orders of magnitude slower than for magnetic disk storage, with disk media usually having a 10 ms latency compared to 50 s for tape storage. Likewise, while disks can handle random access, tapes are best accessed sequentially, but can in many cases achieve faster device data rates.

Mount Overhead

The mounting overhead is usually less time-consuming than seeking, but additionally, significantly reduces the lifetime of cartridges, highlighting the fact that the medium is primarily suited for archival storage that is rarely recalled from. As cartridges are increasingly used for both long-term preservation as well as for frequent access, for example in data carousel studies, remounting is desired to be minimized for both reasons.

Seek Overhead

Data on tapes is usually organized on parallel tracks and read sequentially. If several files are recalled from a single tape, the read operations should be ordered so that they can be executed according to this relative physical location on tape. Skipping between different positions leads to longer overall seek times. Additionally, some newer tape drives support so-called *Recommended Access Ordering* (RAO), which allows reducing seek times by moving the read head between tracks in order to minimize the physical skipping between files that are logically far apart but physically less distant. In general, the seek overhead decreases as the volume recalled from a tape increases.

3.2 Optimizing Bring-Online Performance

The first step to implementing changes within dCache is understanding the general potential for optimizing tape recall performance through scheduling strategies by disk systems in front of automated tape storage. In order to avoid unnecessary or potentially harmful changes and to tweak the right parameters, it is necessary to gain some basic understanding of internal tape system behaviour.

The performance of a tape drive reading a certain file may be approximated using the overall percentage of data recalled in the current session as well as the collocation information. To maximize the throughput for a single drive, the seek times have been shown to be minimized by reading over 80 percent of its capacity, ideally the whole tape. The smaller the percentage of data recalled with respect to the maximal capacity of a cartridge (factoring in the compression rate), the smaller the resultant performance will be overall.

Generally, all the tape systems under discussion will cluster requests per tape and do basic reordering of file requests based on their physical or logical location, potentially even making use of RAO capabilities. Therefore, optimizing the tape recall performance by choosing the dispatch sequence of read requests becomes less fine-grained. A tape-system-fronting storage element such as dCache should ideally collect and cluster requests by tape, keeping track of the relative recall volumes so as to approximate performance were it to be mounted and recalled. It should decide which requests for which tape to send to the tape system based on factors such as the number of maximally available tape drives, which limit from how many tapes data could be recalled at once, and the tape system request queue size, which is used for reordering based on on-tape position. Even though the fact that tapes usually remain mounted for a certain period of time without triggering a remount can be taken advantage of in order to refill the queue with requests while that tape is still being processed, the ordering based on on-tape position will be disturbed more frequently, leading to more seeking behaviour. The smaller this tape request queue, the more seeking will likely be necessary, so the queue length in general needs to be treated as one of the relevant limiting factors.

Finally, it should be noted that the relation between the number and sequence of request targets and their location on tapes is the most important element in determining to what degree recall optimization is possible. Reordering requests becomes more effective the larger the number of known requests is, and file recall generally more efficient the larger the file sizes are.

4 Simulating Basic Tape Recall Scheduling

The primary objective of simulating basic tape system behaviour is to evaluate the recall efficiencies of different request scheduling approaches and the effect of the tape system request queue length. Neither of which can realistically be evaluated using a physical tape

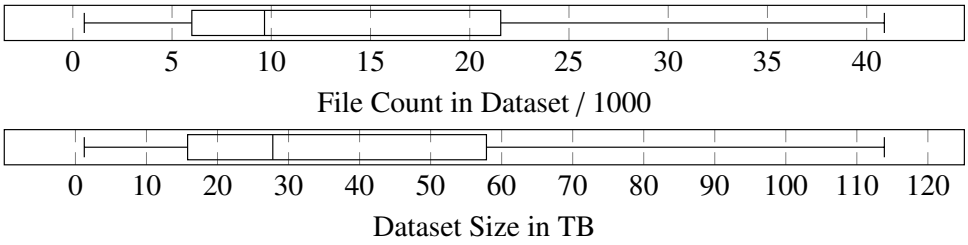
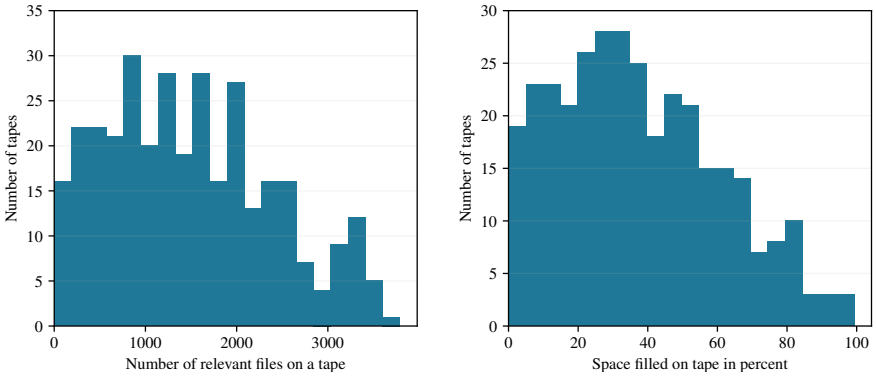


Figure 1: Dataset file count and size distribution(ATLAS reprocessing campaign Jan. 2020)



(a) Number of files recalled per tape

(b) Recall data volume per tape

Figure 2: Tape count histogram for the number of files recalled per tape as well as the associated recall volume in the ATLAS reprocessing campaign Jan. 2020 on 332 KIT tapes

infrastructure for larger numbers of requests, tapes and data volumes, in both cases due to lack of the necessary disposable resources and the time it would take to conduct the tests. The process-based *Discrete Event Simulation* Python framework *SimPy* [16] was chosen for the implementation because it offers the required functionality for process interaction via Python's generator functions and management of several types of resources to simulate capacity congestion points.

One core problem of tape recall optimization endeavours is that infrastructure differs from site to site, both in terms of the physical hardware but also the highly critical distribution of the requested data over and on tapes. For this reason the simulation will be abstracted, idealized and optimized to focus only on certain aspects of relevance that it is able to reproduce as accurately as necessary to answer the questions under investigation. It is not designed to emulate any specific existing tape system and hardware infrastructure. In order to be able to assess the approximate accuracy of the model and resulting behaviour, the German Tier 1 site *KIT* has kindly provided data on their infrastructure as well as data distribution and request patterns for a reprocessing period in 2020, making it an optimal case study.

4.1 Case Study KIT

In the course of the ATLAS reprocessing campaign in Spring 2020, as part of the data carousel experiments, 35 datasets were recalled from KIT, which amount to a volume of 1.1 PB in

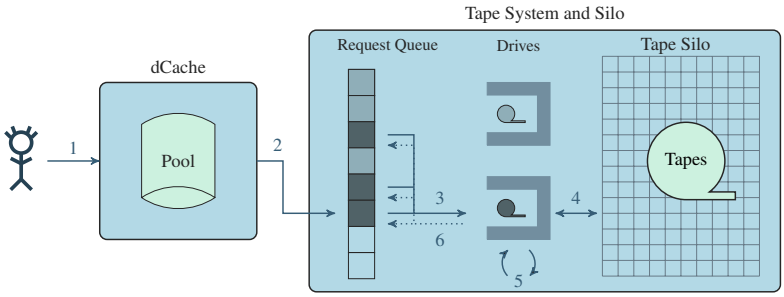


Figure 3: Tape system simulation behaviour overview. (1) Client submits requests to dCache, (2) subset is sent to the tape system request queue if there is space left. (3) A drive is assigned the next most relevant cartridge and associated requests. The cartridge needs to be mounted in the drive. (4). The requested files are read (5) and finally removed from the queue (6).

495,049 files written in the years between 2016 and 2018. The boxplots in Figure 1 show the distribution of file counts per dataset and dataset sizes.

The requested data resided on 332 8.5 TB cartridges. Figure 2a visualizes the number of files that are recalled per tape, which are approximately 1500 on average. Figure 2b shows the percentage to which the 332 used tapes are filled by currently relevant ATLAS data. The y-Axis of both plots shows the number of tapes. Recalling everything that is desired for the reprocessing campaign would in most cases mean reading less than 60 % of each tape's capacity, which limits the maximum possible recall speed under the assumption of optimal request ordering. Additionally, most datasets are themselves spread over 10 to 15 cartridges, up to more than 40, and while most tapes contain one to three, some contain six different dataset parts, which increases the difficulty of request grouping.

4.2 Simulation Setup and Scenarios

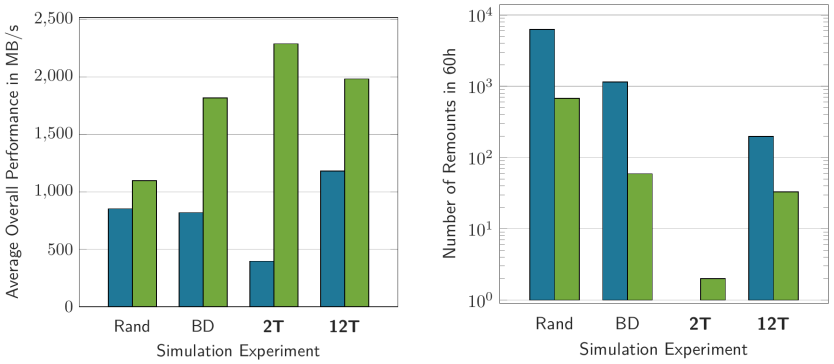
The simulation is dominated by the behaviour of the requester, in this case dCache, which reorders requests before passing them on to the tape system where they are put into the request queue, and the tape drives, which are the primary active component in the simulation and responsible for mounting and dismounting cartridges, fetching and reordering requests for the tape they should mount, calculating read performances, reading the data and removing the requests from the tape system queue upon finishing (see Figure 3).

To better approximate the behaviour of a tape system in the data carousel experiment, the data provided by KIT was used to configure the simulator. During this exercise, 12 drives of type T10000D [17] were used, each having a nominal performance of 252 MB s^{-1} on uncompressed data. Similarly, the values for average rewind times (97 s), tape load times (13 s) and unload durations (23 s) were incorporated as well as the data distribution information. In the *by dataset* scheduling approach, files are requested in the order that the datasets were requested from KIT, and random scheduling is used as a benchmark for comparison. Since clustering requests by tape and sending all requests for one after the other is expected to be highly suboptimal if more than one drive is used, the additional *2T* and *12T* approaches reflect for how many tapes requests are sent in parallel.

In previous exercises, the site had been using a queue size of 2,000, which they were able to increase to 30,000 at the beginning of 2020 by changing to a different dCache-TSM connector. Both queue sizes are used in the analyses to represent a small and a compara-

Queue Size	Sim Mode	Avg Overall Throughput MB/s	Avg Time for Staging in min	Mounts (Distinct)	Avg Capacity Recalled per Mount in %	Avg Mount Duration in min
2000	Random	855	88	6585 (329)	0.3	11
2000	By Dataset	836	78	1221 (77)	1.8	45
2000	By 2 Tapes	393	178	27 (27)	38.0	611
2000	By 12 Tapes	1104	76	269 (71)	8.5	153
30000	Random	1090	783	1004 (330)	2.4	40
30000	By Dataset	1779	536	192 (133)	22.0	225
30000	By 2 Tapes	2263	509	146 (144)	33.0	280
30000	By 12 Tapes	1958	518	168 (135)	26.0	244

Table 1: Summary of simulation results



(a) Tape recall performance

(b) Number of tape remounts

Figure 4: Simulation results: Average recall performance and number of remounts per tape for different queue sizes (blue: 2k, green: 30k) and scheduling patterns (Rand: random; BD: by dataset, the way they arrive; 2T, 12T: pass on requests for 2 or 12 tapes in parallel)

tively large queue size that is able to contain 0.4 % and 6 % of the overall request file count, respectively. The improvements seen in the KIT experiments were similarly observed in the simulation results.

Each simulation run, defined by a combination of queue size and scheduling strategy, took place over the virtual duration of 60 h, so that, depending on efficiency of the respective strategy, the recalled data volumes differ and the number of distinct tape mounts may vary.

4.3 Results

The most relevant metrics differentiating the values of different request scheduling patterns are collected in Table 1 with the respective values for each simulation mode and queue size being summarized visually in Figure 4. These are primarily the averaged overall throughput, the average time for staging a file from the time it has arrived in the queue as well as the number of (distinct) mounts. Additionally, the average percentage of capacities recalled per mount and average duration that a tape remained mounted are included for the sake of context.

Overall, the simulation results show clearly that a larger queue size is beneficial independent of the request pattern used. The concrete results pertaining to the KIT setup at the time, most importantly their use of twelve tape drives, are the following. With random request selection, the overall performance was improved by 27 %, the number of remounts were reduced by 89 %. The idealized *by dataset* request pattern results in a similar overall performance as the random request pattern for the queue size of 2000, which shows the impact of such a highly suboptimal queue size, leading to 0.3 % and 1.8 % of averaged recall capacity, respectfully. Despite the large difference in the number of mounts, the large seeking overhead clearly dominates. In the case of the small request queue, which is not able to contain more requests than need to be retrieved for a single cartridge on average in the current KIT data distribution (usually between 500 and 2500), selecting equal requests for as many tapes as there are drives was confirmed to be the optimal approach. In the case of the large request queue of 30000, which is usually able to contain all requests for the twelve drives used in the analysed reprocessing campaign, scheduling requests for two cartridges in parallel was observed to be better than sending one or as many in parallel as there are drives. This is because the trickle of requests that trigger reordering to minimize seeking becomes more relevant.

The evaluation for improving tape restore request scheduling in dCache by means of simulation is described in more detail in the referenced thesis [18].

5 Implementing Bring-Online Request Scheduling in dCache

The simulation experiments described in the previous section indicate that significant improvements can be made with regard to the recall performance and number of tape remounts if a requester in front of automated tape storage clusters requests by tape. Additionally, it seems to be beneficial to base the tape selection sequence and parallelization on the number of available drives and tape queue size. The larger the tape system request queue, the better, which in the context of dCache also means having a correspondingly large disk area for space reservations and is thus not arbitrarily scalable.

In order to be able to make use of these promising strategies, a proof-of-concept recall request scheduler was added to dCache in the SRM component, which is the current de facto standard for bulk recalls from tape within the system.

5.1 Storage Resource Manager, Status Quo

The *Storage Resource Manager* (SRM) is is a middleware component that provides file management and dynamic space allocation on shared storage components while supporting protocol negotiation and replication mechanisms. The SRM specification [19] standardizes the interface, thus allowing for uniform access to heterogeneous storage elements. It is currently used as the primary bulk tape interface in the dCache ecosystem.

Within dCache, the SRM service is split between a front-end `SRM` and back-end `SrmManager` component for scalability. When requesting an exclusively tape-resident file via SRM (*bring-online* request), the request is passed through `SRM` to the `SrmManager` where it is persisted in a database to survive restarts and sorted into a request-type-specific scheduling component which currently supports *FIFO*, *LIFO* and fair-share request scheduling based on *uid* or *gid*.

When a request is activated, it is passed to the `PinManager` component responsible for persisting file replicas on disk for a defined period of time, which then requests additional metadata from the *namespace*, asks the `PoolManager` for the best data server (`pool`) candidate and triggers the staging on a tape-connected `pool`. When the file has successfully arrived on disk, a confirmation message is sent back to the `SrmManager`.

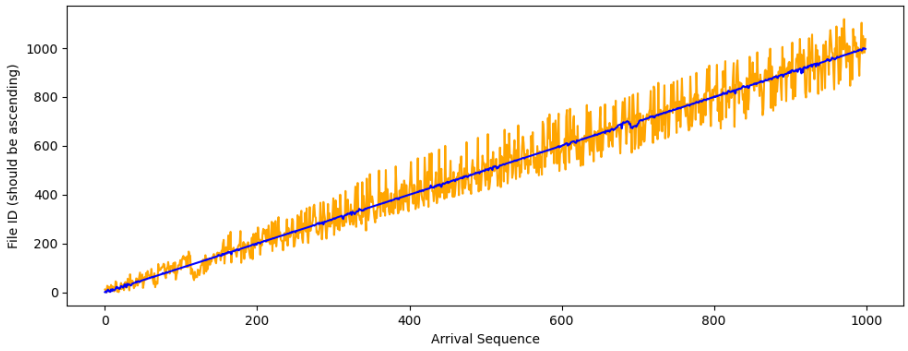


Figure 5: Files are submitted by the `SrmManager`'s bring-online scheduler according to their file ID (blue) but arrive at the `PinManager` more dispersed (orange)

5.2 Clustering Logic

In the newly introduced proof-of-concept bring-online scheduler, requests are first aggregated and successively updated with tape location information, if such is available. Based on configurable hard or soft criteria, tape-associated or individual requests are put into an immediate queue before being submitted in sequence.

The tape information which needs to be provided consists of file size and tape name for each file identifier, and the overall capacity as well as used space for each tape that data is being recalled from. When requests cannot be augmented with tape location information for their target, they will eventually 'expire' their scheduling lifetime and be passed on without being grouped. Otherwise, requests are clustered by tape. A configurable number of tapes may be active at any point in time for dispatching their associated requests in parallel, which corresponds to the *by x tapes* scheduling strategies evaluated in the simulation and should be configured according to the number of available drives and tape system queue size as indicated.

When such a tape slot is free, a new tape may be selected if it has expired requests, or else if it is the tape for which the most data is to be recalled currently and it has either surpassed a certain recall tape volume percentage (possibly relative to the used space) or number of requests, both of which may be configured. Finally, a tape is never selected when the last request for that tape has not arrived prior to a certain time window (except if expired requests are targeting it), so that a tape is not selected when new requests are still incoming at a relatively high rate, increasing the potential recall volume.

Because dCache is based on microservices and messaging between different components happens asynchronously, it is important to note that requests dispatched from the `SrmManager` to the `PinManager` may arrive out of order. An analysis has shown that up to 200 consecutive requests may intermingle (see Figure 5), meaning that it is inadvisable to request fewer than that number of files per tape. If that is not possible, then the maximum allowed number of active tapes should be reduced in order to benefit from scheduling capabilities. For the general optimization context of large-scale bulk recalls, enough files will usually be requested.

6 Summary and Outlook

This paper highlighted the main bottlenecks for bring-online recall scheduling by disk systems, dCache in particular, in front of automated tape store. It presented analyses regarding the optimization potential both with regard to maximizing recall performance and minimizing the number of tape mounts. High-level grouping and scheduling strategies were evaluated in a simulated environment. On account of the promising results obtained, a proof-of-concept bring-online scheduler implementation within the SRM component of dCache was introduced and presented.

After initial tests have been successful and the nature of some limitations of the scheduler is understood, the next steps include deploying and testing the new component at KIT in the near future in order to evaluate its impact in a realistic environment.

References

[1] T.M. Ruwart, techreport, Sandia National Lab. (SNL-NM), Albuquerque, NM (United States) (2018), https://prod-ng.sandia.gov/techlib-noauth/access-control.cgi/2018/181519r.pdf

[2] www.dCache.org, *dCache Website*, https://www.dcache.org, accessed: 2021-02-10

[3] L. Evans, P. Bryant, Journal of Instrumentation **3** (2008)

[4] J. Albrecht, A.A. Alves, G. Amadio, G. Andronico, N. Anh-Ky, L. Aphecetche, J. Apostolakis, M. Asai, L. Atzori, M. Babik et al., Computing and Software for Big Science **3**, 7 (2019)

[5] ATLAS, *Estimated Disk Resource Needs for 2018 to 2032*, https://twiki.cern.ch/twiki/pub/AtlasPublic/ComputingandSoftwarePublicResults/diskHLLHC_18.png (2017), accessed: 2021-02-10

[6] Barisits, Martin, Borodin, Mikhail, Di Girolamo, Alessandro, Elmsheuser, Johannes, Golubkov, Dmitry, Klimentov, Alexei, Lassnig, Mario, Maeno, Tadashi, Walker, Rodney, Zhao, Xin, EPJ Web Conf. **245**, 04035 (2020)

[7] G. Aad et al., JINST **3** (2008)

[8] J. Shiers, Computer Physics Communications **177**, 219 (2007), proceedings of the Conference on Computational Physics 2006

[9] P.A. Millar, O. Adeyemi, G. Behrmann, P. Fuhrmann, V. Garonne, D. Litvinsev, T. Mkrtchyan, A. Rossi, M. Sahakyan, J. Starek, 2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP) pp. 651–657 (2018)

[10] NeIC, *ENDIT code repository*, https://github.com/neicnordic/endit, accessed: 2021-02-10

[11] Davis, Michael C., Bahyl, Vladímir, Cancio, Germán, Cano, Eric, Leduc, Julien, Murray, Steven, EPJ Web Conf. **214**, 04 (2019)

[12] D. Litvintsev, A. Moibenko, G. Oleynik, M. Zalokar, Journal of Physics: Conference Series **396** (2012)

[13] R.A. Coyne, H. Hulen, R. Watson, *The High Performance Storage System*, in *Supercomputing '93:Proceedings of the 1993 ACM/IEEE Conference on Supercomputing* (1993), pp. 83–92

[14] M. Kaczmarski, T. Jiang, D.A. Pease, IBM Systems Journal **42**, 322 (2003)

[15] T. Johnson, E.L. Miller, *Benchmarking tape system performance*, in *Proc. of Joint NASA/IEEE Mass Storage Systems Symposium* (1998)

[16] SimPy, *SimPy 3.0.11 Documentation – Overview*, https://simpy.readthedocs.io/en/latest/, accessed: 2021-02-10

[17] O. Storagetek, *Storagetek T10000D Tape Drive Data Sheet*, https://www.oracle.com/technetwork/topics/security/140sp2254-2298100.pdf, accessed: 2021-02-10

[18] L. Morschel, Bachelor thesis, FH Wedel (2020), University of Applied Sciences Wedel, 2020, https://bib-pubdb1.desy.de/record/437255

[19] *The Storage Resource Manager Interface Specification Version 2.2*, https://sdm.lbl.gov/srm-wg/doc/SRM.v2.2.html, accessed: 2021-02-22