Department of Computer Science

Bachelor Thesis

Improving Tape Restore Request Scheduling in the Storage System dCache

Evaluation of Pre-Scheduling Strategies by Disk Systems in Front of Automated Tape Storage in the Context of the ATLAS Experiment

Handed in on:

February 24, 2020

Author:
Lea Morschel
Bahnhofstr. 32
22926 Ahrensburg
Email: lea.morschel@desy.de

Supervisor (University):
Prof. Dr. Dennis Säring
University of Applied Sciences Wedel
Feldstraße 143
22880 Wedel
Email: dsg@fh-wedel.de

Supervisor (Company):
Tigran Mkrtchyan
Deutsches Elektronen-Synchrotron DESY
Notkestraße 85
D-22607 Hamburg
Email: tigran.mkrtchyan@desy.de

Abstract

This thesis aims to improve the data recall efficiency from magnetic tape to disk by optimizing the request scheduling within the scientific disk storage system dCache. It is written in the context of the future data management challenges by the ATLAS experiment, where the high luminosity configuration will require an order of magnitude more storage capacity than can be provided by the current funding and usage model. A concept called data carousel is being investigated by the ATLAS collaboration, which aims to realize the active yet efficient usage of tape storage. Established components in the data management chain such as dCache are aimed to be adapted to efficiently support this altered workflow.

In the context of this thesis, the current request scheduling within dCache is evaluated, the problem formalized and analyzed with regard to finding an optimal solution. A solution is proposed and discussed. Due to the complexity of the system chain and resulting behavior, which is not easily subjected to experimentation, a tape system simulator is developed, dCache modeled as a requestor thereof. The proposed dCache request scheduling algorithms are then tested in this environment according to defined criteria, the results evaluated. It is shown that the proposed solution is capable of significantly improving the metrics of interest in different setups.

Contents

List of Figures

List of Tables

List of Abbreviations

ALICE	A Large Ion Collider Experiment
AOD	Analysis Object Data
API	Application Programming Interface
ATLAS	A Toroidal LHC Apparatus
CASTOR	Cern Advanced Storage Manager
CERN	European Organization for Nuclear Research
CMS	Compact Muon Solenoid
DAOD	Derived Analysis Object Data
DDM	Distributed Data Management
DES	Discrete Event Simulation
DESY	Deutsches Elektronen-Synchrotron
EGEE	Enabling Grids for E-sciencE
Fermilab	Fermi National Accelerator Laboratory
FIFO	First In First Out
FTS	File Transfer Service
HDD	Hard Disk Drive
HEP	High Energy Physics
HPSS	High Performance Storage System
HSM	Hierarchical Storage Management
KIT	Karlsruhe Institute of Technology
LHC	Large Hadron Collider
LHCb	LHC-beauty
LRU	Least Recently Used
LTFS	Linear Tape File System
LTO	Linear Tape-Open
RAO	Recommended Access Order
TSM	Tivoli Storage Manager
TSS	Tertiary Storage System
VO	Virtual Organization
VTL	Virtual Tape Library
WLCG	Worldwide LHC Computing Grid
WORM	Write-Once-Read-Many

1. Introduction

In times of big data and abundantly emerging machine learning applications, the volumes and variety of data that need to be stored and processed are increasing towards the exascale level. Hierarchical storage systems are needed to fulfill the requirements at minimal cost, which combine different storage technologies of varying characteristics, such as hard disk drives and magnetic tape, and move data between locations based on current access requirements (see Section 2.2).

In the field of *High Energy Physics* (HEP), the major exascale challenge is approaching with the planned upgrade of the *Large Hadron Collider* (LHC) at the *European Organization for Nuclear Research* (CERN) to the high luminosity configuration (HL-LHC) in the second half of the 2020s. 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 available resources assuming a flat-budget model and the requirements for storage and computing power (see Section 3.1). Storage space is even more problematic than the difference in computational power.

The experiment collaboration around the *A Toroidal LHC Apparatus* (ATLAS) detector (see Section 2.3), is working on solving the storage resource discrepancies by implementing and optimizing a so-called *data carousel* model that aims to make active use of tape storage, largely instead of less affordable disk capacity. The goal is to enable large-scale recalls of tape-resident data without relevant loss in performance compared to disk recalls, which is to be expected due to the robotic access and inherently sequential nature of tape storage. The data carousel motivation and concept is described in more detail in Section 3.1. The analysis workflows for the LHC experiments involve a globally connected network of research institutes providing computational power, disk and tape resources, which is referred to as *Worldwide LHC Computing Grid* (WLCG). Recalling data from tape therefore involves both global data management and transfer protocols as well as site-local disk and tape storage systems. The majority of the largest WLCG sites, called Tier 1, use the dCache software as their primary storage system (see Section 2.5). It is distributed, highly scalable and accessible via a variety of protocols as well as capable of migrating data to and from a connected tape storage system. The ATLAS file requests are sent to a dCache instance, which queues and schedules them before passing them on to the tape backend in order to optimize the reading and writing efficiency. It is hoped that this mechanism may be improved in order to better serve the tape carousel purpose.

The highly configurable nature of dCache leads to vastly different installations which are additionally connected to complex software layers before varyingly intelligent tape backends. Such a system is highly abstracted and treated as a black box by dCache, which restricts its ability to make informed scheduling decisions. The tape system request queue length in combination with the number of tape drives is a crucial bottleneck, which will need to be optimally filled with requests. Understanding and improving this selection and pre-scheduling of read requests by the dCache storage software is the objective of this thesis.

1.1. Related Work

The vast majority of research on the performance of tape systems concerns optimizing the recall efficiencies, frequently taking into account the effect of different writing strategies. Johnson et. al. [21] benchmarked the performance of hardware components of tape systems to better understand the characteristics and optimization potential of tertiary storage devices. The systems under scrutiny differ in several aspects, such as data track layout, directory structure and location, data compression as well as block size. In order to assess the end-to-end performance of a tertiary storage device, they define the most relevant characteristics to be the mount and unmount time, the seek time, as well as the the transfer rate, which can be influenced by aspects such as data compression. The performance characteristics of tape storage are determined to be unusual due to their large access latencies combined with high transfer rates. While mount and unmount times are almost deterministic, the seek times can be characterized by complex profiles. Because of the sequential nature of tape storage, reducing the large seek times is the primary goal of performance optimization research. Such optimization approaches should then be taken into account by mass storage system designers in order to optimize larger storage system chains. In an effort to bridge the general access-gap between storage on tape and disks, Dashti et. al. [12] focus on using intelligent data placement on serpentine tapes at the physical track level, which is described as one of several proposed solutions of doing so. The others concern reducing cache misses at the disk level, interleaving disk and tape operations and applying more intelligent tape scheduling. The approach of synchronizing disk and tape access is followed by Myllymaki et. al. [32], who studied the optimization potential of join operations in database management systems under consideration of tape devices as storage media. These approaches to optimization are often very low-level or hardware-specific and for the most part internal to the tape system which is often addressed by disk systems as a black box, although the results of such studies may be used in more high-level approaches.

At such a level of abstraction, Yu et. al. [49] describe ways of efficiently accessing large amounts of tape-resident data using the scheduling softwares *ERADAT* (*Efficient Retrieval and Access to Data Archived on Tape*) and *DataCarousel* at the BNL Tier 1. The DataCarousel is a policy-driven framework to allow fair-share resource assignment in multi-use and multi-group setups, while ERADAT is a lower-level queuing and scheduling system that sits between DataCarousel and interfaces with the HPSS tape management software and reorders requests so that the number of mounts is reduced, the overall resource usage and read performance is increased. ERADAT may be configured to sort requests by time and tape ID, which allows for a good compromise between fairness, time expectations and performance optimization. In [48], where ERADAT is compared to the new HPSS feature of *Tape Order Recall*(TOR), the general conclusion of optimizing tape recall performances by bulk-staging 50 % or more per tape mount is drawn, which is independent of the other scheduling approaches.

The performance of a system such as ERADAT is maximized when the known requests therein are maximized. The BNL Tier 1 site, which uses a dCache storage system, has been able to optimize their tape recall performance significantly by using the DataCarousel and ERADAT softwares before the HPSS tape system. However, a dCache instance insists on reserving space for all files which are requested from a connected tertiary storage system. Due to the limited disk capacity, a mere subset of the known requests needs to be selected before being passed on, reducing the optimization efficiency of common tape systems or enhanced schedulers like ERADAT. Therefore, dCache itself needs to more intelligently pre-select subsets of requests that it passes on, so that the overall performance may be optimized more efficiently, which will be beneficial for both simple and smart connected tape systems.

1.2. Thesis Outline

This thesis is organized in several chapters. After the introductory part, Chapter 2 introduces important concepts, ranging from data management and organization, to technologies such as magnetic tape storage and software like the dCache system.

Chapter 3 analyzes the data carousel project, past and present experiments and their results. It investigates factors impacting the performance at different levels and specifically focuses on the resulting optimization potential within dCache. Current mechanisms and problems are described and investigated.

In the following, Chapter 4 focuses on optimizing the dCache interaction with a tape system by analyzing the problem of optimal read request scheduling for tape systems with a certain number of drives and a limited request queue size. It proposes different scheduling mechanisms within dCache, estimating their added benefits and shortcomings. Acquiring and making use of additional information is considered. Afterwards, scheduling criteria and approaches for evaluating the proposed scheduling mechanisms are defined. Finally, the case study of the KIT Tier 1 site is introduced, which will be used in the evaluation, their setup and data distribution described and the optimization potential discussed.

The simulation developed in order to evaluate different request scheduling strategies to a simplified tape system is described in Chapter 5. The created model is introduced, its design principles described and the applicability as well as limitations discussed. Finally, the configuration and different setups for later analysis are introduced.

The results of evaluating different scheduling strategies are illustrated in Chapter 6. These results are summarized in Chapter 7, which concludes with an outlook.

2. Background

The ATLAS data management challenges are hoped to be improved with the realization of the data carousel concept. In the following, the relevant technologies and concepts are introduced. Section 2.1 presents an overview over the distinct characteristics of scientific data and its management, while Section 2.2 introduces hierarchical storage management. Afterwards, Section 2.3 describes the LHC ATLAS experiment, addresses computing models and challenges. Section 2.4 is dedicated to magnetic tape storage, explaining its history, different data storage technologies and formats. It also introduces tape management concepts such as tape libraries and dedicated storage management software. Section 2.5 introduces the dCache storage system, its characteristics and tape interaction capabilities while Section 2.6 briefly takes a look at other ATLAS software that is relevant for the data carousel.

2.1. Scientific Data Challenges

Data that is the result of scientific experiments needs to be stored for documentation, reproducibility and reprocessing. The characteristics of scientific data, its acquisition and usage lead to distinct challenges and resultant requirements of its organization, management and access that need to be addressed [39].

The output of scientific experiments that needs to be stored and accessed for processing is generally of large volume. It ranges from requests of hundreds of gigabytes to storage of dozens of petabytes and is increasingly growing to the exabyte level. The ingest is usually fast and frequently occurs in bursts. Building an infrastructure that is able to satisfy all of these requirements is challenging. Persistence and long term archival of data are of significant importance, especially with regard to raw data and parameter sets used for simulation that cannot be reproduced upon loss. In a similar notion, the integrity and protection of data needs to be ensured. Because the outputs of experiments should not be changeable, data is also ideally immutable. Even derived or generated data should not be modifiable but rather recreated with different parameters as a new set of results if desired.

The access to scientific data is often chaotic, the requestors heterogeneous and distributed around the globe. Data is accessed for interactive analysis or batch processing, frequently also initially for wide area transfers in order to be used at another location. Access control and data ownership need to be balanced with ways of securely sharing data in order to allow for safe but collaborative research. This thesis is primarily concerned with large-scale bulk reprocessing of large amounts of archived experiment data, which is distributed over several sites.

2.2. Hierarchical Storage Management

Different requirements for storing large amounts of data, primarily concerning fast ingestion and access as well as persistency, are difficult to balance. In order to satisfy them, different technologies are combined to avoid the high cost of using exclusively high-speed devices. In this

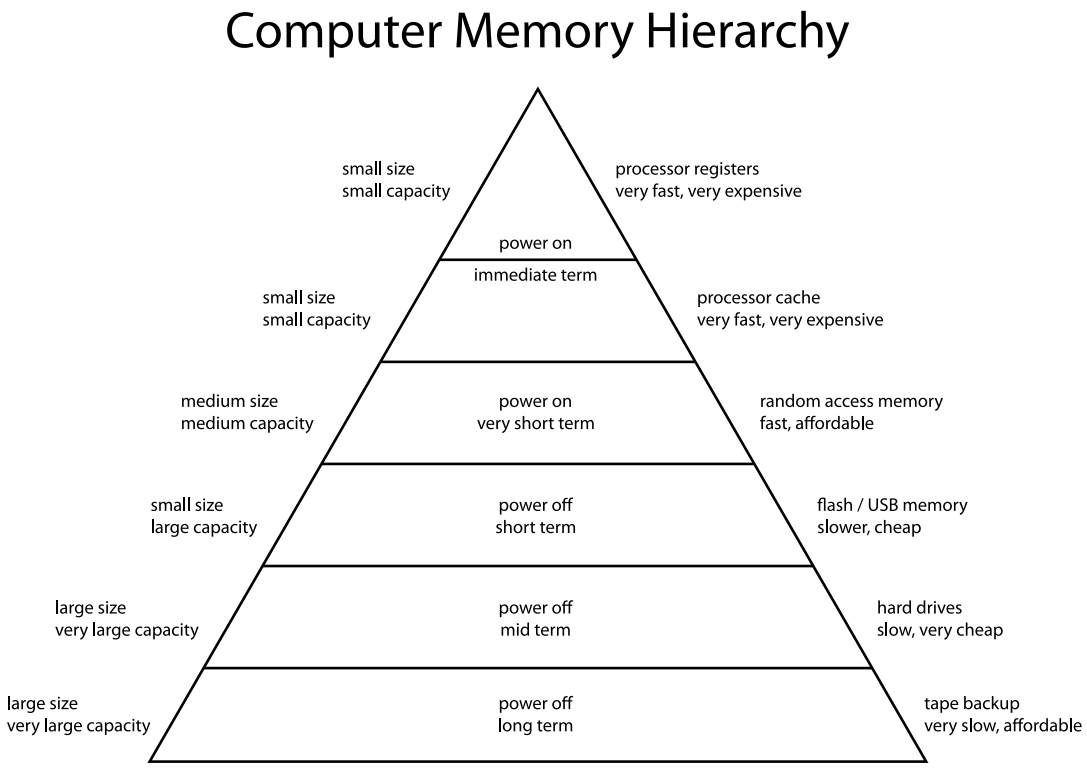


Figure 2.1.: From [36]: Computer Memory Hierarchy. The price for a certain capacity increases along with the performance from lower to upper layers while the persistence decreases.

model, storage technologies are conceptualized as forming a hierarchy that includes fast but expensive devices of therefore low affordable storage capacity on the upper end and slow, but cheap devices of large affordable capacities on the lower end. Such a hierarchy can bee seen in Figure 2.1. It explicitly highlights that the fastest devices such as caches and random access memory require power for running, increasing the operational cost and linking data persistence to power supply, whereas the lowest layers, featuring hard drives and magnetic tape, merely require power for access. The main reasons for using tape as a storage medium today and its persistent relevance are this inexpensiveness and reliability.

Such a storage setup is often referred to as tiered storage or *Hierarchical Storage Management* (HSM). In many cases, such systems manage two technologies, disk and magnetic tape media. Below a certain I/O latency threshold, storage media are referred to as *nearline*, which is usually the case for tape. In HSMs, data is moved between storage layers to fulfill different requirements. These may be based on abstract criteria such as access latency and likelihood of data loss (retention policy), which are ideally separate from the concrete implementation. For example, a low likelihood of data loss could be realized as a tape copy or as multiple disk copies in different locations. This concept is referred to as the *quality of storage service*.

In HSM systems, faster media are used as caches for slower storage elements. In so-called automated tiered storage, a form of HSM, data is moved between different layers in an automated way. Often, the complexity is then hidden from the user. They do not know where data is stored and ideally do not notice excessive data access latencies. It is similar to the usage of memory caches in a regular computer. Hierarchical Storage Systems may be connected in series to each interface with a lower-level HSM.

2.3. The LHC ATLAS Experiment

The *Large Hadron Collider* (LHC) [16] is the largest and most powerful particle accelerator in the world, located in a 26.7-kilometre ring beneath the France-Switzerland border. It was built by CERN over a period of ten years in collaboration with different laboratories and universities from over 100 countries around the world.

It started operation on September 10, 2008 and has since been alternating between three to four years of operation and *long shutdowns* of two years for maintenance and upgrades. It has been built to test predictions made by different theories in the field of particle physics, to solve unanswered questions and measure known parameters more precisely. The perhaps most notable result during the LHC runs so far was the confirmation of the existence of the Higgs boson in 2012 [3], which is associated with a mechanism that explains why particles have mass. This discovery lead to the awarding of the Nobel price to Peter Higgs in 2013, who is responsible for the theoretical prediction.

The particle beams inside the accelerator are made to collide at four locations around the LHC ring, where the four main particle detectors reside. *ATLAS* [1] and *CMS* (Compact Muon Solenoid) are the largest, general-purpose detectors, which took part in the discovery of the Higgs boson. *ALICE* (A Large Ion Collider Experiment) and *LHCb* (LHC-beauty) are more specialized regarding the aims and processes of investigation.

Computing and Data Management Inside the LHC two beams of particles travel in opposite directions. The bunches cross inside the ATLAS detector with a rate of 40 MHz, which is reduced to 400 Hz by multi-level hardware and software triggers selecting the events of interest [2]. The detector output that has been pre-selected by the hardware-level trigger is immediately transferred to an online storage system that is able to handle the high data rate. Software triggers are then executed and the selected data is stored in a large buffer before being transferred to the mass storage facility.

Currently, it is expected that the LHC produces 50-70 PB of data yearly, which cannot be processed, stored and made accessible efficiently by CERN alone. Therefore, the *Worldwide LHC Computing Grid* (WLCG) project [38], short *Grid*, was created. It is a global collaboration of about 170 computing centers from 42 countries that connect the different local resources such as processing power and data storage capacity. Overall, this distributed computing infrastructure contains about one million computer cores and an exabyte of storage. The 12,000 LHC physicists can submit jobs to the system via one of the many access points which then make use of the resources provided for the experiments without the user having to manage where they are coming from. In LHC Run 2 (2014-2018), CERN stored data at rates of about 8 GB/s while global transfers were often at over 60 GB/s.

WLCG is arranged in tiers ranging from 0 to 3, which is illustrated in Figure 2.2. The CERN Data Center is the Tier 0 and displayed in the center. It is responsible for storing one tape copy of all raw data and generating first reconstructions. It also distributes raw and reconstructed data to the Tier 1s. In down-times of the LHC, it takes part in the reprocessing of old data. The Tier 0 provides around 20 % of the total computing capacity. The Tier 1s are thirteen large computer centers (see Section 3.2). They each store a proportion of the raw and reconstructed data on tape as a second copy, engage in large-scale reprocessing of data and store the outputs. The Tier 1 sites distribute data to the Tier 2s and safe-keep a portion of the data simulated there. The German Tier 1 centre is called *Grid Computing Centre Karlsruhe* (GridKa), which is located at the *Karlsruhe Institute of Technology* (KIT). Taking another step down the Tier hierarchy, there are about 160 Tier 2 sites which each have an associated primary Tier 1. They mainly

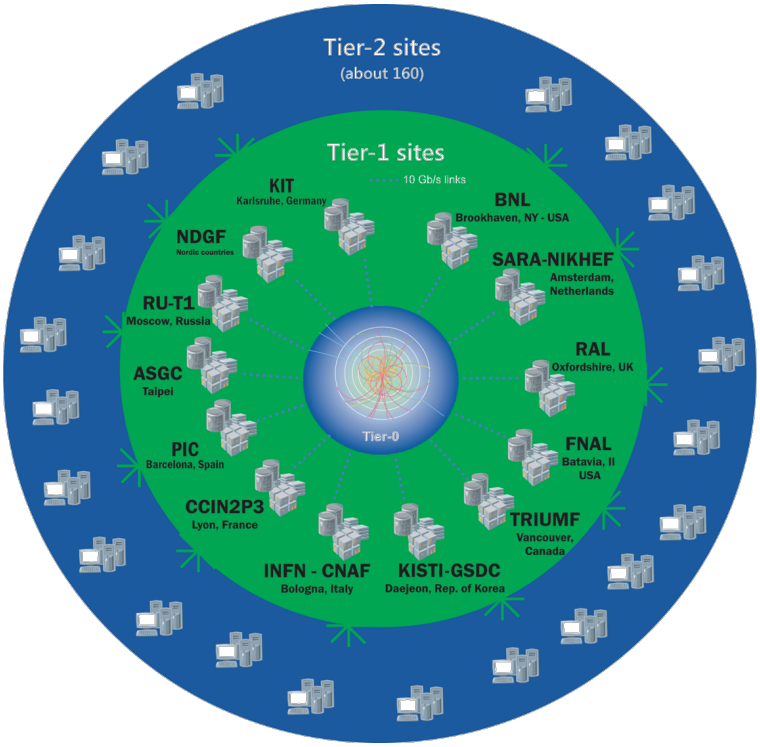


Figure 2.2.: From [45]: WLCG Tiers. The Tier 0 at CERN is shown in the middle, the Tier 1s are surrounding it. The outer circle represents the Tier 2s.

consist of universities and other scientific institutes, that can store certain amounts of data and provide computing resources for specific analyses such as production of simulated events. They do not store raw data nor, unlike Tiers 0 and 1, provide tape infrastructure for WLCG. The *Deutsches Elektronen-Synchrotron* (DESY) research institute is one example of a Tier 2. While Tier 1s and 2s have strict *Sevice Level Agreements* regarding maximally permissible downtimes, Tier 3s do not. They mostly consist of individual scientists or small local clusters and are not under formal obligations regarding the Grid. More details regarding the computing models of the WLCG and the large experiments can be found in the latest WLCG update report from 2014 [9].

The ATLAS collaboration has established specific software stacks, process chains and workflows to optimize their computing efficiencies. It manages around 270 PB of storage as well as 350,000 cores on average distributed over 150 WLCG sites, which is a subset of the general WLCG resources. Over 250,000 to 300,000 jobs are routinely running in parallel, analyzing detector and simulation data. Figure 2.3 shows the main components of the ATLAS distributed computing system and their responsibilities. The top layer shows different actors submitting jobs, such as individual users as well as *ProdSys*, which is a workflow management system translating scientific requests into production jobs. The *PanDA* system (see Section 2.6) on the left is the main data-driven workload management software. It places jobs into global job queues and is responsible for scheduling and assigning resources. A service interacts with the main data management framework *Rucio* (see Section 2.6) shown on the right, which handles data distribution based on rules, current requirements by jobs or general access patterns as well as associated performance and locality requirements. Finally, a job should be processed as efficiently as possible on assigned resources, accessing ideally local data with low latency. Details on the ATLAS distributed computing system may be read up in the respective overview paper [15].

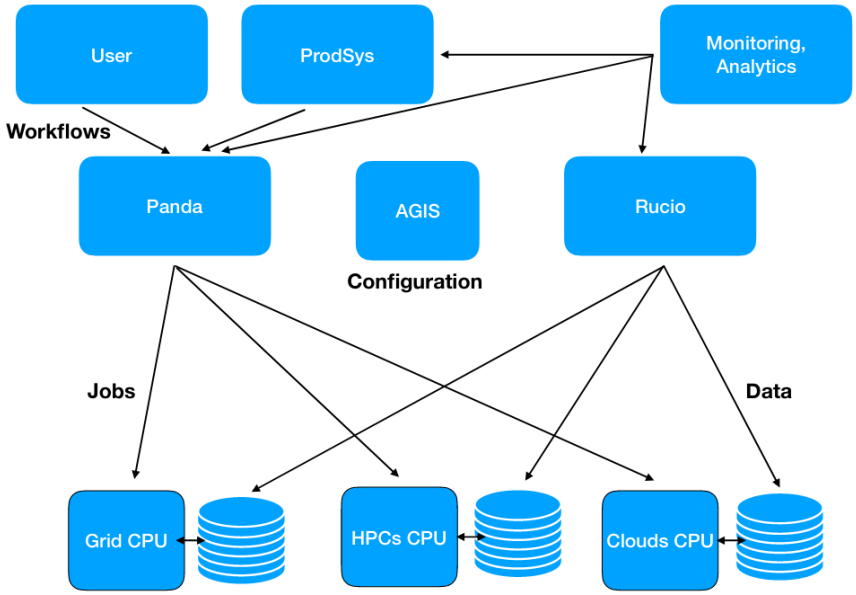


Figure 2.3.: From [15]: The ATLAS Distributed Computing System Schema. Jobs are submitted by users or via *ProdSys*, which are then being scheduled by *PanDA* to be executed on certain resources while *Rucio* coordinates the data placement.

There are different types and formats of data that need to be stored. 60 % of the total space for ATLAS are dominated by *Analysis Object Data* (AOD), which contains the summary of a reconstructed event and is sufficient for common physics analyses, and *Derived AOD* files (DAOD), which is used in statistical analyses of many events [5]. AOD and DAOD data are stored both for the actual experiment data as well as simulated and reconstructed results obtained using *Monte Carlo* methods [29]. The ATLAS computing system is used to run continuous Monte Carlo simulations and reconstructions, user analyses and a large overall reprocessing campaign once a year at Tier 1s and Tier 2s. The different types of data differ in volume and file size distribution, which is important for making informed storage decisions.

2.4. Magnetic Tape Storage

Storing information on magnetic tape, as is done for archiving the scientific data for the LHC experiments, has a long tradition. The technology is based on magnetic wire recording used for audio storage, which was invented in the beginning of the twentieth century. Although the technology has come a long way since that time, the basic principle remains the same. Magnetic tape is a long, flat band consisting of a polyester-based carrier substrate that is on one side coated with a layer of oxide. The combination is only a few micrometers thick. The oxide coat can be magnetically manipulated to store information. Most commonly tapes are 12.7 mm wide, but other sizes exist to accommodate different needs. In order to read, write or delete data, magnetic heads are passed over the tape medium by winding it from one reel to another by using a so-called tape drive device.

Generally, tape belongs either in the category of offline storage technologies, which cannot be automatically accessed, or *tertiary storage systems* (TSS), which is operated robotically. For comparison, secondary storage is defined as not being directly accessible by CPUs, an example being hard disk drives (HDDs), whereas primary storage includes processor registers and memory. Tape technology does not require power for persistence, merely for operating the drive while

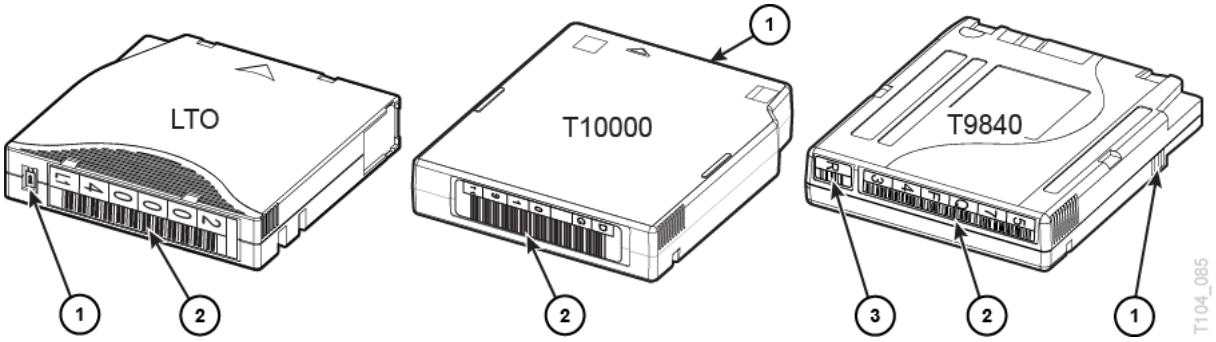


Figure 2.4.: From [35]: Examples of tape cartridge formats. (1) Write-Protect Switch, (2) Cartridge Label, (3) Separate Media ID for specific types. The actual tape inside is not displayed.

accessing a tape. It allows to store data for many years up to a few decades. It is overall more robust than hard disks. However, tapes are not intended for frequent access, which could significantly reduce their lifetime. The medium is optimal for archival storage, in which data is written once and then only rarely accessed again. When data that is written cannot be modified, it is referred to as Write-Once-Read-Many (WORM), which is frequently desired in the context of tape storage. 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 magnetic disk storage, with disk media usually having a 10 ms latency and tape 50 s. Likewise, while disks can handle random access, tapes are best accessed sequentially, but can in many cases achieve faster device data rates.

In an overarching storage perspective, tape is primarily attributed with being slow, secure and cost-efficient.

2.4.1. History and Future of Tape Storage

The first usage of magnetic tape for computer data storage was in 1951 with the *UNISERVO* tape drive as the primary I/O device for the *UNIVAC I* computer [26]. It could write 7.200 characters/s on 8 tracks, which was 56 times faster than the rate of punch cards. Up to 224 kbit could be stored on a 1200 ft tape.

Since that time, tape storage and its usage has improved significantly. In October of 1974 IBM announced the *IBM 3850 Mass Storage System* (MSS) tape library as one of the first examples of nearline storage, which describes a system that resides between online, meaning immediately available, and offline storage, requiring human intervention for access. The MSS consisted of a robot that could fetch cylindrical tape cartridges of 50 MB from a library and insert them into a device for reading and writing. It was used to automatically off-load infrequently used data to tape, referred to as destaging, and automatically transfer it back to disk, called staging. In 1984 IBM introduced the *IBM 3480* tape drive that operated on a single tape reel that was located within a rectangular cartridge, capable of storing 200 MB. The usage and format of this cartridge is still a standard up to this date. Three different modern cartridge examples can be seen in Figure 2.4.

In 1998, the *LTO Consortium* was founded by IBM, Hewlett Packard Enterprise and Quantum (then Certance) in order to create an open tape specification. The resultant *Linear Tape-Open* (LTO) technology has been widely adopted and is now one of the leading formats in use. Due to rising demands for capacity, so-called supertapes were introduced, such as the *Super Advanced*

Intelligent Tape (SAIT) in 2003 and the *Super Digital Linear Tape* (SDLT) in 2007. They peaked at capacities of around 800 GB. All of them are discontinued as of now. Modern tape cartridges allow for about 10 TB native data capacity.

Currently, due to dropped prices for HDDs, the initial investment for tape technology is often higher than that for disks. However, the low maintenance cost, high durability, good backward compatibility and easy expandability of tape still make it the cheapest solution for long-term storage of large data volumes [8]. Native features such as on-the-fly encryption and the inherent air-gap combined with comparatively slow accessibility provide an extra layer of favorable security in times of increasing numbers of cyberattacks. It may even offer some extra protection against software errors. Surprisingly, the scaling up of capacity is expected to keep increasing at a rate of 33 percent a year, whereas disk technology is slowing down, making tape a relevant medium even for the foreseeable future.

Even though the technology outlook for tape storage is very positive, there are problems concerning practical aspects. Observably, the I/O speed is not increasing at the rate of capacity, even though it is still able to provide higher I/O rates than typical disk systems can handle, and price declines of tape media are slightly stagnating. More importantly, however, the current market for tape is problematic. Oracle withdrew from research and development of tape drives in 2017, leaving IBM as the sole provider. Additionally, the two main manufacturers of tape media, Sony and Fujifilm, are locked in a patent war over LTO technology, which has significantly affected the production and availability of LTO-8 cartridges throughout 2019 [28].

In recent years there has also been a trend towards using *Virtual Tape Libraries* (VTLs), which is a disk-based technology emulating tape. It is therefore generally easy to integrate into existing backup environments, is significantly faster than physical tape media and may take care of tasks such as data deduplication and compression. Finally, the VTL data can itself be selectively saved to tape for archiving purposes or serve as a disk buffer for tape caching.

Still, magnetic tape storage is the most cost-efficient technology for storing vast amounts of data and is expected to remain an integral part of scientific data management. A 2018 report [37] discusses the challenges and requirements for exa-scale storage of scientific data, analyzes different technologies, products and data management techniques. It explicitly addresses the current and future importance of tape storage in this context, focusing on its cost-effectiveness, the continually increasing cartridge capacities, longer media lifetimes and reduced bit error rates corresponding to the growing amounts of data that need to be stored persistently. Tape is said to continue expanding beyond its historical role of backup technology to an archival storage offering cost-efficient access to large quantities of data in different fields.

The 2018 report of the *Computing Resources Scrutiny Group* at CERN [5] refers to tape storage as an essential resource that has increased by an average of 23 % annually, resulting in a total of 575 PB in 2018, of which the *ATLAS* and *CMS* (see Section 2.3) experiments each use approximately 40 %. The Group criticizes that tape storage is currently underutilized by some experiments relative to the space pledged, ATLAS being one of them. For this experiment, a tape capacity increase of around 50 % is expected by the year 2021, factoring in efforts to reduce certain data footprints and aims to improve tape access efficiencies via workflows based on the so-called *data carousel* model.

2.4.2. Magnetic Tape Data Storage and Formats

There are different ways for data to be organized on tape. It can be recorded either in different variations of a linear approach by writing data in parallel tracks along the length of the tape, or a scanning fashion, writing data along its width. The speed and data storage capacity will in most

Tape Drive	Developer	Release Date	Capacity Native (Compressed)	Max. Speed Native (Compressed)
T10000C	Oracle	2011	5.0 TB	240 MB/s (360 MB/s)
T10000D	Oracle	2013	8.5 TB	252 MB/s (800 MB/s)
LTO-6	LTO Consortium	2012	2.5 TB (6.25 TB)	160 MB/s (400 MB/s)
LTO-7	LTO Consortium	2015	6.0 TB (15 TB)	300 MB/s (750 MB/s)
LTO-8	LTO Consortium	2017	12 TB (30 TB)	360 MB/s (900 MB/s)
TS1155	IBM	2017	15 TB	360 MB/s (800 MB/s)
TS1160	IBM	2018	20 TB (60 TB)	400 MB/s (900 MB/s)

Table 2.1.: Characteristics of Relevant Tape Formats. [41, 42, 19, 20, 33, 34]

cases be affected by the choice of approach. For example, the liner-serpentine method enables much higher space utilization that the purely linear approach, by shifting all read/write heads slightly after having each written a single track and then moving in the opposite direction. This is the most common way to layout data on tape today. Few systems still use the helical-scan pattern, whereby short tracks are written diagonally across the tape width.

Data is usually written to tape in blocks separated by gaps, which may be encrypted, compressed or buffered before ending on tape. New data is always appended. Blocks of modified or released files are marked as unavailable and cannot be made use of, unless the whole tape is reformatted, which makes it distinctly different from random-access disk technology.

Because magnetic tape storage has existed for a long time, many different formats have been in use over the years. Today, there are a small number of leading formats that dominate the market, each of which consist of a combination of a tape drive along with usable tape media devices. Many of the most relevant tape drives and associated maximal cartridge specifications today are displayed in Table 2.1. The lifetime of different tapes is another relevant factor, that is differently specified per vendor. Most have an archival lifetime of 30 years, which is reduced by the number of accesses. Sometimes the number of maximal mounts/unmounts or, alternatively, end-to-end passes is given, above which the probability of error rises above a certain threshold. In the case of the cartridges T10000 and T10000 T2 for the drive T10000D, for example, the number of mounts/remounts should not exceed 15,000 and 25,000, respectively [10]. LTO-8 cartridges, on the other hand, are expected to safely endure 20,000 end-to-end passes.

StorageTek Txxxxx The *Txxxxx* StorageTek formats are one of these. They are now owned by the Oracle Corporation. The newest *T10000D* variant features a 8.5 TB non-compressed tape media capacity and speeds of up to 252 MB/s.

IBM TSxxxx The *IBM 3592* series features *TSxxxx* drives, of which the newest model *TS1160* of generation 6 offers native storage capacities of 20 TB at speeds of up to 400 MB/s. These formats are especially known for being highly interchangeable and backwards compatible.

LTO The best-selling super tape formats are part of the Linear Tape-Open technology referred to as *Ultrium*. LTO is an open tape storage specification by the LTO Consortium and developed by Hewlett Packard Enterprise, IBM and Quantum. Consequently, there are several manufacturers for any of the defined formats. With few exceptions, an LTO tape drive is capable of reading data from tape of its own as well as the two previous generations and writing to tape of its own

and the previous generation. The newest format, *LTO-8*, features 12 TB native data capacity and uncompressed speeds of up to 360 MB/s. The roadmap for LTO Ultrium formats is displayed in Figure 2.5, showing the aim to double native capacities with each generation.

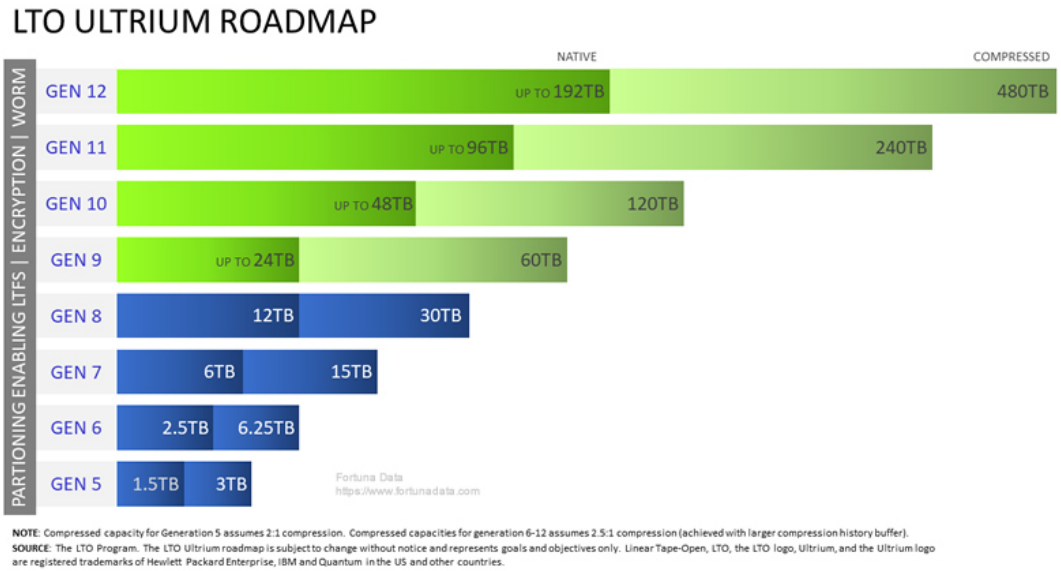


Figure 2.5.: From [46]: LTO Roadmap

Historical tape storage does not hold file metadata, such as file names or timestamps, which is therefore commonly managed in external databases, making the media largely incomprehensible without this information. This lack of filesystem characteristics lead to the development of the self-describing *Linear Tape File System* (LTFS) specification by IBM, which became part of the LTO Consortium in 2010. By storing metadata on a reserved section of tape and having a corresponding software interpret the format accordingly, the media can be presented as if the data were residing on a mounted disk. This especially allows for greater ease in exchanging tape-resident data.

2.4.3. Tape Management

In order to access a tape for reading, also called staging, or writing, referred to as flushing, it needs to be retrieved from its physical storage location and inserted into a tape drive. Especially for large storage environments, this process is required to be automatized. Today, most tape systems are automated tape libraries.

Tape Library

A tape library is a collection of tapes in cartridges labelled with barcodes that are packed in racks. It includes one or several tape drives that allow reading and writing data and corresponding slots for inserting cartridges. A tape library with a single tape drive is commonly referred to as an autoloader. A robot with a barcode reader identifies, fetches and loads requested tapes. Usually, a library has multiple robots working in parallel and may contain thousands of tapes, accommodating capacities ranging to a few exabytes. Access to data can take several seconds up to a few minutes.

Most modern tape libraries are designed to be incrementally scalable up to a vendor-specific set limit. Although terminology differs, a combination of units is generally referred to as a library complex. Systems from different vendors have unique selling points, focusing on features such as high tape densities or small seek times. Some of the most relevant and powerful modern examples of tape library complexes are the *IBM TS4500*, the *Oracle StorageTek SL8500* and the *Spectralogic TFinity ExaScale*.

Tape Storage Management Software

Data storage and retrieval from tape libraries is handled by tape management systems. They maintain lists of tape volumes as well as file locations, translate high-level instructions to hardware operations and thereby enable locating or tracking media. When large data volumes are being archived and accessed, a dedicated HSM or archiving system is frequently used to specifically handle the tape backend and pass on this information to the disk *Storage Element* (SE). Because these applications define the interface presented to the rest of the storage hierarchy and handle the scheduling and optimization of large amounts of higher-level requests, they will in the following simply be referred to as tape systems.

There are many different such approaches and software solutions that help to integrate tape into a storage hierarchy. Some of the most important ones will be introduced in the following.

HPSS The *High Performance Storage System* (HPSS) is a proprietary HSM system [11]. It is being developed by the HPSS collaboration consisting of IBM and five US Department of Energy laboratories. HPSS is designed to incrementally scale with the provided resources, offer high availability and durability by ensuring data integrity. The presentation of a single namespace eases the management of large setups. The performance is optimized by transferring large files in parallel, ordering read and write requests and collocating related files, for example on the same tape or tape set. HPSS aims to improve hardware utilization and maximize performance. It can be configured through policies that define automated processes.

TSM *IBM Spectrum Protect*, previously called *Tivoli Storage Manager* (TSM) is a proprietary backup and recovery platform offering highly scalable, centralized and automated data protection functionality [22]. The system consists of client applications which send data to server components that save corresponding information in a database and manage transferring the client data to storage devices such as cloud services, hard disk drives, disk arrays, tape drives or libraries. TSM offers complete and incremental backups, compression, replication, deduplication and cost-effective storage by enabling data-lifecycle automation. It also provides ransomware detection.

CASTOR and CTA The *Cern Advanced Storage Manager* (CASTOR) was developed by CERN and is the currently deployed system at the institute for managing transfers from disk to tape storage. Due to hardware evolution and increasing demands its efficiency has become limited. Therefore, a successor called *CERN Tape Archive* (CTA) has recently been developed, which makes use of the core tape server component of CASTOR but aims to redesign the previous architecture [13]. In a more simplistic approach, CTA will not manage disk, tape and the staging in between, but limit itself to function as the tape back-end for the EOS disk system, which is also being developed by CERN and is already the de-facto storage system on site. The software works around a queueing system implemented as an object store as well as a metadata database that can both be updated by tape server and frontend agents to reorder requests so that tape access efficiencies can be maximized. Currently, dCache is not able to use CTA as a tape backend.

Enstore The open-source mass storage system *Enstore* [25] is being developed by the *Fermi National Accelerator Laboratory* (Fermilab), a US-American Tier 1 site located near Chicago. The project started in 1998 in order to satisfy requirements for Run 2 of the local Tevatron collider experiments. It provides scalable and fault-tolerant management as well as distributed access to tape-resident data. In order to provide a unified, hierarchical namespace view of stored files, it uses the Chimera namespace provider, which is developed at DESY as a central component of the dCache storage software, with which Enstore is optimized to work with. The system ensures data integrity, manages resources, provides tape allocation accounting for example per storage group or file family, is able to perform migrations from older to newer media types and can aggregate small files to increase performance.

Tapeguy The Canadian Tier-1 centre at *TRIUMF* developed the non-proprietary, high-performance HSM system *Tapeguy* in 2010 for efficient tertiary storage operations, especially in the context of the data-intensive computing model of the ATLAS experiment [14]. It is optimized to work with the dCache storage system and is successfully deployed at TRIUMF. Tapeguy is designed to be scalable, manages metadata that enables providing a common file namespace and allows controlling dataset writing in order to group and collocate files that belong together. It manages the dynamic allocation of resources, handles tape drive load balancing, orders requests to minimize tape loading and guarantees timely file delivery to clients by implementation of request prioritization.

OSM The *Open Storage Manager* (OSM) software provides automated hierarchical storage management functionality to administer different TSS systems and tape libraries. It was originally developed by Legent Corp. and offered as a proprietary solution. Eventually, the rights were sold to CA. The DESY research center later acquired a license with access to the source code, and has since then maintained its own modified version. Because the rights still belong to CA, the software cannot be made open-source or shared beyond DESY, which lead to the development of Enstore, which is similar in many aspects. It is also deployed there and used to interface with the local tape installations to manage the long term data storage for several experiments, primarily in the area of photon sciences.

2.5. The dCache Storage System

The *dCache* software is an open-source distributed storage system that is used at most Tier 1 sites to manage their main storage element (see Section 3.2). It is written in *Java* and uses a microservices-like architecture to provide location-independent access to data. It is designed to support a wide range of use cases, from high-throughput data ingest, being (dynamically) scalable to hundreds of petabytes, as well as deployable in heterogeneous systems and on commodity hardware. It is easy to integrate with other systems, because it can communicate over several protocols for accessing data and enabling authentication. It also supports different qualities of data storage, including tertiary storage support, for which it is able to use disks as a caching layer [30]. The name reflects its original purpose: to function as a disk cache for tape. As a scientific data storage system, dCache is designed as a WORM system, but has in recent years acquired noWORM capabilities in order to more easily be usable as a general-purpose file system [31].

2.5.1. Main Components and their Interactions

Within the dCache architecture, a microservice is termed *cell* and represents dCache's most fundamental executable building block. Each cell fulfills a specific task and can be grouped into a certain type category, for example *pool* as a storage component or *door* for enabling access to data over a specific protocol, such as *NFS*, which then gets translated into internal instructions. A *domain* is a container for cells that runs in a single *Java Virtual Machine* (JVM) instance. The included cells, each running as a separate thread therein, can be arbitrarily chosen but are usually combined in a way that is deemed useful to achieving a certain objective together. A domain's name is required to be unique within a dCache installation.

A minimal dCache instance consists of a single domain housing specific cells. A door and a pool type are necessary for accessing and storing data. Two management components are required: a *pnfsManager* that supervises the metadata namespace, and the heart of the dCache system, a *poolmanager* that manages all the pools in a system and regulates data dispatch. The namespace manager has the prefix *pnfs*, because dCache internally identifies files by unique identifiers called *pnfs-ID* due to to the original *Pretty Normal File System* (pnfs) component. As an external dependency, an *Apache Zookeeper* [47] instance is also necessary as a distributed directory and coordination service. New cells can easily be added to an instance. Growing the storage capacity, which often also improves the aggregated throughput of an instance, just requires the addition of new pool nodes. In order to provide resilience of an instance to node failures, dCache supports running in *high availability* (HA) mode, wherein redundant cells for a service can be run in parallel, such as having multiple pnfsManagers.

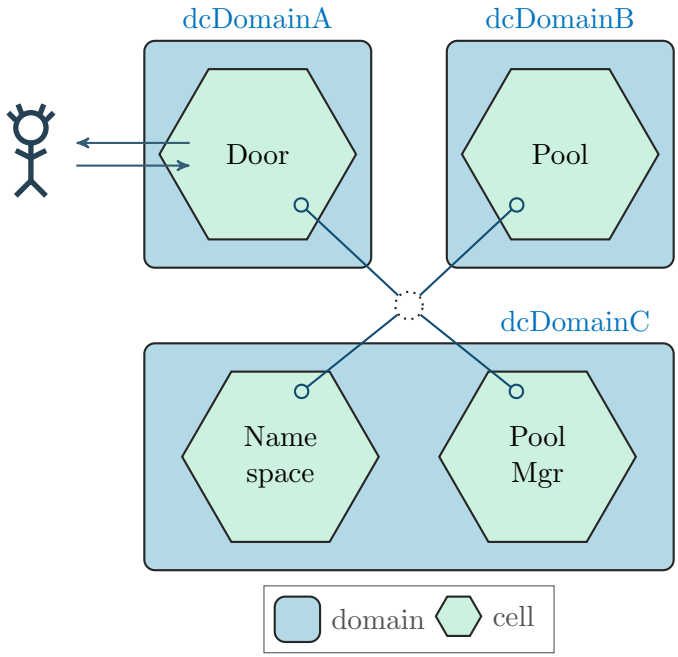


Figure 2.6.: Visualization of a dCache instance with the main components

When a request arrives, for example via an HTTP door, the requestor is authorized via the dCache access management component *Grid-aware PLuggable AuthoriZation Management* (gPlazma). In order to access an existing file, the door requests metadata from the pnfsManager and then asks the PoolManager to select a pool that can provide the respective file. If the file resides on disk, the client is redirected to communicate with one of the pools that stores a copy directly.

2.5.2. The dCache Tape Interaction

The dCache system is able to move disk-resident data to one or multiple connected TSS systems and migrate it back to disk when necessary. It does not operate tape robots or drives itself and assumes to be connected to an intelligent TSS which does. In most cases, the TSS in use is a tape system or migrating HSM system (see Subsection 2.4.3). This is illustrated in Figure 2.7.

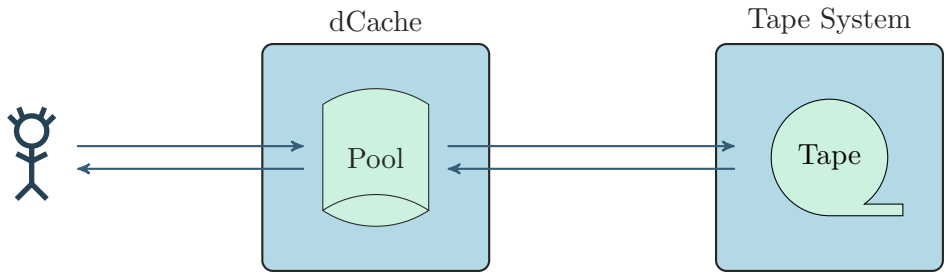


Figure 2.7.: The dCache system transparently interacting with a tape backend

There are two different ways to interface with a tape-connected HSM: a pluggable driver *Application Programming Interface* (API) which is suitable to create smart HSM connectivity, an example being ENDIT, and an external script, that is provided as a reference implementation of the API, which is designed to be stupid and simple yet brilliant. In order to communicate with a TSS, the script abstracts the interface to the operations `PUT`, returning a unique identifier that is stored, `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 backend by the script. The syntax of the standard executable can be seen in Listing 2.1, where `put`ting returns a `storage-uri` to identify the passed file on tape which then needs to be included in the respective `get` and `remove` calls.

```
put <pnfsID> <filename> -si=<storage-information> [<other-options>...]

get <pnfsID> <filename> -si=<storage-information> -uri=<storage-uri>
    [<other-options>...]

remove -uri=<storage-uri> [<other-options>...]

-------------------------------------------------------------------------

WHERE:
<pnfsID>: internal identifier (i-node) of the file within dCache
<filename>: is the full path of the local file to be copied to the TSS
<storage-information>: the storage information of the file (sClass, size, etc.)
<storage-uri>: the returned URI after the file was written to tertiary storage
<other-options>: -<key>=<value> pairs from TSS conf. cmds of pool setup file
```

Listing 2.1: Syntax for the dCache TSS-support executable

All pools in a dCache instance that should be connected to as TSS need to be provided with a HSM script, but not all of them need to be. Many sites have set up their system so that TSS-connected pools are not accessible by the end-user. Therein, staging requests trigger transfers of files from a TSS to certain connected disk pools, which are not accessible to users, from where these files are then moved via pool-to-pool transfers to accessible ones. This is done in order to relieve the strain from the TSS-connected pools.

2.6. Other ATLAS Distributed Computing Software

The ATLAS experiment group is distributed all over the world. Over time, different tools have been created for efficient distributed computing. This includes job workflow management as well as access to data and resources at remote sites while hiding some of the complexity from end users.

Rucio The *Distributed Data Management* (DDM) project was created to take care of data placement, deletion and access of the distributed ATLAS experiment data. The latest version, called *Don Quijote 2* [18], has in recent years been largely replaced by *Rucio* [17], which is a distributed system designed to overcome the previous scalability limits of Don Quijote 2 [7]. While Rucio has initially been created in the context of the ATLAS experiment, it has soon expanded to the wider area of *High Energy Physics* (HEP) and other scientific communities.

Rucio manages data access by means of accounts that are mapped to single or multiple individuals. Files are the smallest unit of operation, which can be grouped into datasets or containers. Metadata is stored using attributes. The physical files are stored in indexed endpoints called *Rucio storage elements* (RSE), which may for example be a *dCache* installation. Information on an SE contains its support for different protocols, an individual threshold for deletion and a weight value for data distribution. Replica management and file grouping in Rucio is based on rules defined on file sets, where a wide distribution over many RSEs enables the utilization of more CPU resources for processing.

FTS The high-level Rucio application uses the lower-level *File Transfer Service* (FTS) [23] for scalable, flexible and reliable bulk data transfers between heterogeneous sites. It is being developed and funded by the European Commission project *Enabling Grids for E-sciencE* (EGEE) in the context of the CERN ecosystem. FTS takes into account the policies of individual Grid sites and allows them to control the respective network resource usage. The system places importance on efficiency, security, fair share mechanisms, auditing and accounting while balancing different, possibly conflicting requirements from resource owners. Robustness is of high importance, because jobs need to be able to run reliably over long periods of time without being supervised.

PanDA The ATLAS experiment is developing the *PanDA* [27] system as its main high-level workflow and workload manager for distributed production and analysis. It is highly automated and tightly integrated with DDM/Rucio for transparent job submission to the Grid.

3. Problem Analysis

This chapter introduces the problem that is intended to be solved in the context of this thesis, starting with the future ATLAS data management challenges along with the currently investigated *data carousel* workflow to more efficiently make use of the available resources in section 3.1. The setups of the Tier 1 sites will be briefly examined in Section 3.2, followed by the description and analysis of the previous data carousel experiments and their results in Section 3.3. Many different factors, layers and components impact the data carousel performance, which will be explored in Section 3.4. Finally, the potential for optimization within the dCache storage system deployed at many Tier 1 sites is investigated in Section 3.5.

3.1. The Data Carousel Motivation and Concept

Although the resources available to the ATLAS experiment described in Section 2.3 are sufficient for the current situation, planned developments lead to increased requirements that need to be addressed. According to current plans, Run 3 of the *LHC* will last from May 2021 to the end of 2024 and is expected to be manageable with current trends in infrastructure and software improvements. In the following long shutdown that will end in mid-2027 the accelerator and detectors are going to be upgraded to be able to produce more data. For Run 4 it will then be in so-called high luminosity configuration (*HL-LHC*). Luminosity is a measure describing the efficiency of a particle accelerator producing collision events. The HL-LHC is predicted to require approximately ten times the resources that are currently needed. This is causing much effort to be spent on trying to plan and create the needed infrastructure, software and data management policies [4].

Figure 3.1 from 2017 illustrates the estimated disk resource needs for ATLAS up to the year 2032. As can be seen, the amount available with the flat funding scenario, which assumes an increase of 15 % per year under current technology trends, is far from sufficient to cover the expected requirements for Run 4. The baseline model already assumes a reduction of AOD/DAOD sizes by 30 % compared to Run 2. The reduced storage model was created under the assumption of even smaller sizes as well as the storage of only one DAOD version of a common format and the complete removal of data from previous years from disk.

In order to cope with this discrepancy, work is being done in different areas to more efficiently make use of the existing resources. In addition to efforts addressing the reduction of AOD and DAOD footprints, more efficient access to data is investigated that might reduce the need to have multiple replicas on disk [5]. ATLAS also manages a *Research and Development* (R&D) activity called data carousel, which investigates the performance of different storage technologies in I/O intensive workflows. Its main goal is to more efficiently conduct large reprocessing campaigns. The primary focus is placed on magnetic tapes, which are significantly less expensive than disks (see Section 2.4) and already used for archival storage at the Tier 0 and Tier 1 sites. However, tape is also much less performant and suitable to random access than disk storage and cannot simply replace disks in the established workflows without significant efforts.

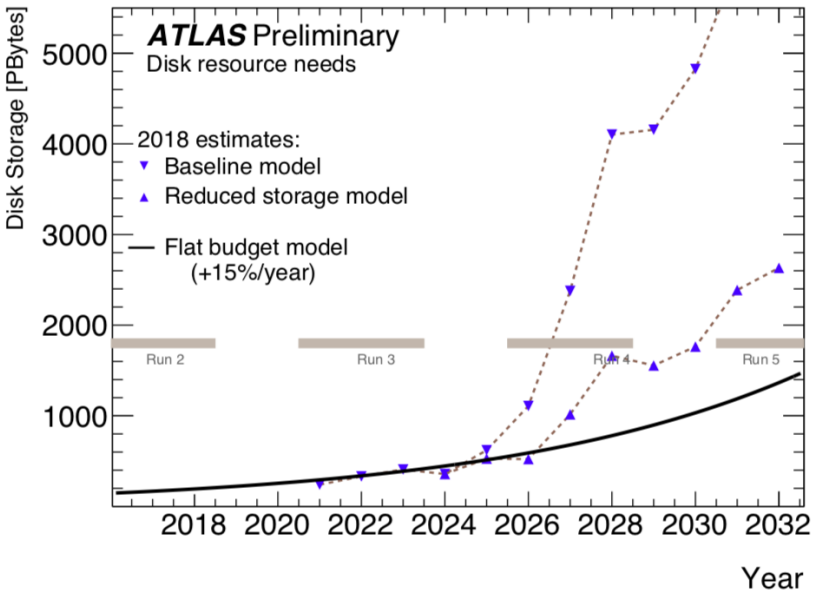


Figure 3.1.: From [6]: ATLAS Expected Disk Resource Needs. The schedule for the different future Runs has since been amended slightly.

The data carousel concept aims at actively using tape as a nearline storage media behind a smaller layer of disk space that is serving requests. The goal is to enable large-scale recalls of tape-resident data without relevant loss in performance due to the robotic access and inherently sequential nature of tape storage. The idea is to no longer allow randomly requesting files from tape, but staging and processing exclusively a sliding window of a fixed percentage of the total data volume. Afterwards, the data on disk is discarded and the next chunk is cycled in from tape. This approach aims at minimizing the tape access overhead by maximally making use of the disk space and thereon currently available data. In order to improve the efficiency and throughput of the tape systems to the end-user, the various components need to be orchestrated over the whole data management system stack, ranging from *Rucio* over *FTS*, the storage endpoints such as dCache and the tape systems themselves. It involves investigating inter- and intra-component bottlenecks, schedulers on different layers, as well as smart writing strategies to optimize reading from tape.

Most sites have historically set up their tape architecture for archival storage and not to optimally support active, I/O intensive usage. It is therefore important to first of all understand the current setups of sites, the mechanisms involved and the currently achievable throughput. Only then the optimization potential can be assessed and improvements made by identifying and correcting problems and using opportunities. The goal is to identify and realize the best scenario at all layers for optimal resource usage. One of them is the storage element layer, which for most Tier 1 sites consists of a dCache installation in front of their tape storage. In coordination with the other layers there might be room for improvement to allow better throughput from the lower-level tape system to the higher-level file transfer service used by the Rucio management layer. By better understanding the current and ideal conditions at these bordering layers, the range of potential within dCache may be assessed and exploited.

Site	Country	Disk Resources (TB (%))	Tape Resources (TB (%))	Storage Element	Tape System
CERN	Switzerland	27,000 (100)	94,000 (100)	EOS	CTA
BNL	USA	20,200 (23)	51,000 (23)	dCache	HPSS
RAL	UK	13,024 (15)	32,708 (15)	Echo	CTA
KIT	Germany	11,000 (13)	27,625 (13)	dCache	TSM -> HPSS
CCIN2P3	France	11,100 (13)	28,700 (13)	dCache	HPSS
TRIUMF	Canada	8,800 (10)	22,100 (10)	dCache	Tapeguy
INFN	Italy	7,920 (9)	19,890 (9)	StoRM	TSM
NL-T1	Netherlands	6,538 (7)	16,076 (7)	dCache	DMF
NDGF	Nordic	5,380 (6)	15,520 (6)	dCache	TSM
PIC	Spain	3,520 (4)	8,840 (4)	dCache	Enstore
RRC	Russia	4,500 (5)	5,700 (3)		

Table 3.1.: WLCG storage resource pledges for ATLAS of the Tier 0 and 1 sites providing tape from January 2020 [44]

3.2. Tier 1 Storage Setups

Out of the thirteen Tier 1 sites, only ten provide tape resources for ATLAS. The resource pledges of the these Tier 1s and the Tier 0 are shown in Table 3.1, which also contains the storage software each site runs to manage their main disk area (Storage Element) as well as the tape system in use. While CERN is able to provide 100 % of the necessary disk and tape space, the Tier 1 sites range from 23 % by BNL in the US to 4 % by PIC in Spain. Seven of these sites use dCache as the main storage element, but the tape system choices are much more diverse. KIT is planning to move away from TSM to HPSS, making the latter the most used software with three installations that would then handle almost 50 % of the Tier 1 tape resources. It is then followed by TSM but also CTA, which is not used by any of the dCache sites (see Section 2.4.3). Tapeguy, DMF and Enstore are only used by one site, each.

Table 3.2 shows, among others, the tape drives available at different sites. They are not necessarily all used at any point in time. It is important to note, that the origin of the numbers also differs. The drives at CERN/KIT/RAL/CCIN2P3/INFN are usable by all logical Virtual Organizations (VO), which are groups of institutions or individuals in the grid computing context, that share resources for a common purpose, such as serving different LHC experiments. At the same time, the numbers for NL-T1/PIC are those reserved for ATLAS usage alone while the sites BNL and TRIUMF are dedicated ATLAS sites that do not provide resources for other LHC experiments. It is, however, noticeable, that most sites use enterprise drives by Oracle and some have LTO technology (see Section 2.4.2 and Table 2.1).

3.3. Data Carousel Experiments

Large reprocessing campaigns which are aimed to be improved with the data carousel concept request the recalls of large amounts of data from tape, primarily from the Tier 1 sites with the Tier 0 as a fallback option. This data will be copied to Tier 2 sites and processed. Afterwards, the data is deleted again from disk.

First experiments started in mid-2018, when a reprocessing campaign of raw data was conducted. Most Tier 1 sites that provide tape resources for ATLAS participated in order to assess their current tape recall performance. The average throughput over the entire storage chain based on

Site	Tape Drives (max. reading)	P1 avg. Thrpt	P1 stable Thrpt	P2R2 avg Thrpt (100% staged)	P2R2 avg Thrpt (90% staged)
CERN	(expect 60 GB/s)	2 GB/s	2 GB/s	267 MB/s	1.2 GB/s
BNL	22 LTO7, 17 LTO6	545 MB/s	866 MB/s	900 MB/s	1.4 GB/s
KIT	30 T10KD	286 MB/s	300 MB/s	316 MB/s	324 MB/s
RAL	21 T10KD	1.6 GB/s	2 GB/s	850 MB/s	1.3 GB/s
CCIN2P3	56 T10KD	2.1 GB/s	3 GB/s	401 MB/s	524 MB/s
TRIUMF	20 LTO8, 12 LTO7	700 MB/s	1 GB/s	366 MB/s	330 MB/s
INFN	10 T10KD, 19 TS1600	255 MB/s	300 MB/s	N/A	N/A
NL-T1	8 T10KC, 2 T10KD, 10 LTO8	630 MB/s	640 MB/s	626 MB/s	630 MB/s
NDGF	N/A	300 MB/s	500 MB/s	214 MB/s	371 MB/s
PIC	4-6 T10KD	400 MB/s	380 MB/s	179 MB/s	170 MB/s

Table 3.2.: ATLAS Data Carousel Experiment Results until Autumn 2019

Rucio data is shown in Table 3.2 as P2R2, both for the time until 90 % as well as 100 % of the data is staged to the destination. Usually, there is a discrepancy between the 90 % and 100 % throughput rate, which is especially striking for sites like BNL or NDGF. It takes a long time to retrieve the last ten percent, which might be due to scheduling problems or unfavorable file distributions on tape.

The next large tape carousel tests were conducted in August of 2019, which lead to the discovery of several problems that are not site-related. One of these concerned problematic behavior of the FTS scheduler, which could not handle a large amount of requests and started a throttling feedback loop with Rucio and sites. Because FTS could not move files from the disk area to which they had been staged at sites to their destination fast enough, they were removed from the buffer and had to be re-staged again later on. Therefore, these results from the first run are also mainly useful for reference purposes and are shown summarized in Table 3.2 with the values average and stable throughput. In most cases, the performances achieved has improved dramatically between P2R2 and P1. Most importantly, the experiments have so far lead to the discovery of problematic system behavior that has in many cases been improved or resolved.

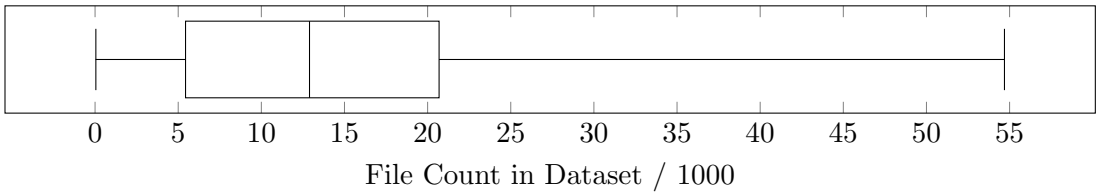


Figure 3.2.: File count per dataset (ATLAS reprocessing campaign January 2020)

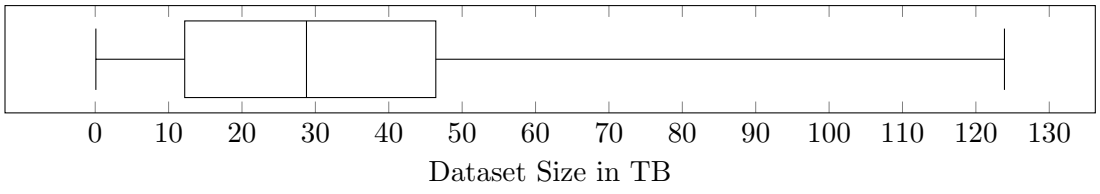


Figure 3.3.: Size per dataset (ATLAS reprocessing campaign January 2020)

Starting in January of 2020, the next reprocessing campaign is conducted on 18 PB of raw data. In order to better prepare sites for the upcoming requests, a list of contained datasets was created.

Overall, they include 8,115,426 files in 596 datasets. On average, a dataset contains 13,616 files and is 30.658 TB large. The average file is therefore 2.251 GB large. The distribution of file counts per datasets is displayed in Figure 3.2 as a box-whisker plot, which shows that the majority of datasets consists of about 6,000 to 21,000 files, but there are large outliers at the upper end up to a about 55,000 constituents. Similarly, the dataset size distribution is shown in Figure 3.3, with percentiles 25 to 75 spanning the range between approximately 12 TB to 47 TB, but a large whisker on the upper end of the scale until 125 TB.

3.4. Factors Impacting the Data Carousel Performance

The system chain that is used to introduce and experiment with the data carousel concept has grown over many years, wherein tape storage was used for archival purposes rather than fast I/O heavy access. Therefore, different factors impact trying to use the available resources in this way, which are aimed to be minimized.

First of all, the hardware setup of tape storage determines the maximally available performance. This includes not just the type of hardware but also the number of robots, amount of drives relative to the tape count, the disk buffer size and network connection to as well as performance of the destination system.

Tape systems or individual drives are often capable of reordering received lists of requests for files. They will be clustered by tape or tape set in order to maximize the sequential reading from each tape. Newer drives are able to calculate the *Recommended Access Order* (RAO) of files, which serves to minimize the number of remounts as well as the on-tape seek times, which are the main performance bottlenecks. Drives usually have a small buffer for reading ahead of requests and absorb performance drops by seek times. Tape systems usually accept a certain number of requests at a time and can answer only as many in parallel as there are tape drives available. For write requests the data itself needs to be buffered, which is again dependent on the size of the disk buffer. Therefore, requests from a larger pool should ideally already be ordered before being sent to the tape system so that the clustering can be more efficient. Because user requests are not deterministic or completely known at any point in time, making this an online problem, this is only possible to a certain degree. Decisions need to be made while they come in. Trickling file requests to a single tape may cause it to be mounted and unmounted continually when new ones arrive, instead of being able to accumulate and work on them in one go. Some sites impose a maximum time before remounts in order to reduce stress on tapes and make mounting them more efficient. Small files are especially impractical due to the inherent I/O overhead, but very large ones may also be difficult to handle. Additionally, spreading a set of requested files over several tapes may be beneficial due to being able to read these back in parallel by using multiple drives as opposed to just a single one, but would require multiple tape mounts in advance. Requesting collocated file sets may be more efficient than requesting individual files. Finally, knowledge of metadata, metrics concerning file popularity or general request patterns could significantly improve scheduling and caching strategies for optimal performance. All of this is further complicated because tape systems are often shared between different communities with separate storage systems, access patterns, requirements and priorities.

The responsibilities tape systems can assume differs vastly. Some merely pass on requests sequentially, some calculate the access sequence of tapes and for the layout on each one in complicated scheduling and caching processes. In all cases, the client, in this context primarily a higher-level disk system, which is aware of more requests than a tape system can accept at once, can improve tape I/O performance by smart selection and scheduling of subsets of requests that are passed on. However, due to the different levels of tape system intelligence and the fact that

the client views the tape system as a black box, redundant, canceling or oscillating behaviors by scheduling pre-scheduled inputs may become a problem that should be carefully assessed. This behavior is a problem in general for the interaction of the different layers. For example, in the last reprocessing campaign to test the recall efficiency of data from tape, the FTS scheduler had problems with throughput and influenced Rucio, which led to a feedback loop of throttling behavior.

The way data is placed and organized on tape is an important determinant for the efficiency of data access. Although there are storage groups that may be a defined set of tapes for a certain experiment for example, there is an even smaller grouping called *dataset* which is the basic unit of analysis and therefore request by ATLAS campaigns. This information is known at the highest layers, but not passed down to the storage elements. If this could be done and data collocated on tape accordingly, it is assumed that the data access performance could be improved dramatically. However, there are still open questions and concerns regarding the expectations and foreseen activities from the ATLAS experiment in general, which are important for the Tier 1 sites in order to adequately adapt their setups. It would be beneficial to know concrete numbers on the expected average and peak data rates, both for reading and for writing. Similarly, information regarding dataset structure, size, number and request patterns is not always clear.

3.5. Optimization Potential within dCache

The dCache system is deployed as the main storage element at most Tier 1 sites as described in Section 3.2. It receives requests via FTS/Rucio for writing files to tape or staging them back. These requests are queued and scheduled before being passed on to the tape system. As a relevant part in the ATLAS storage management chain, the way dCache handles these requests has a great influence on its performance.

Because dCache is highly scalable and can accept large numbers of requests, it may act as a buffer for data and requests to a connected tape system. When a user wants to read a file, it may either be cached on a pool and served directly, or fetched from tape storage. Because these backends usually have a more limited request queue length and dCache will not attempt to stage more files than there is space available on disk, requests accumulate within dCache and are drained over time. This has the advantage of having a much larger window of requests that may be reordered before being sent in chunks to the tape system. Instead of passing on 500 read requests for 100 different tapes, one may select 500 different requests that can be recalled from a single tape. In an ideal case, all requests would be known and sent to the tape system, which has the necessary information to optimally reorder them for maximal throughput. Because this is not possible, dCache is forced to pre-schedule smaller chunks of requests. Unfortunately, as a logical layer of abstraction, it does not know about the physical locations for files on tape for optimal recall. Organized writing, however, may be enforced by defining and associating storage groups with logical file system structures. These allow collocating large sets of files that belong together on one tape or a small set. The larger these storage groups are, the less optimal the recall efficiency will be. Mapping smaller logical units of files such as datasets to storage groups would lead to a suboptimal tape usage, however. Additionally, this detailed dataset grouping information is currently not propagated down the ATLAS data management chain and therefore not yet available to dCache.

Tape connectivity is an essential part of the original dCache design, where disk space may be used as a caching layer for a connected tape infrastructure. As described in Subsection 2.5.2, tape management is hidden from the end-user via write-back and read-through like behavior. This includes that data is always initially written to the disk layer and then migrated to tape based on

flush policies. The latter entails that a requested file that is not present on disk is automatically fetched from tape and cached on disk before being provided to the user. Because access to tape is expensive, dCache offers a feature called *stage protection*, which allows to control access to tape-resident data based on VO, user or protocol that may be defined using a corresponding rule.

As a distributed system, dCache manages multiple file servers called pools, as introduced in Section 2.5, which may reside on different physical hosts. Each pool is an independent unit that does not know about the others in a dCache installation. It has knowledge about the files residing in its own space and whether it has a connection to an HSM instance. A system administrator can add or remove pools, configure their size and HSM connection. Each pool can be connected to none, one or multiple of such instances. Usually, a dCache installation has a single tape HSM backend shared by all pools.

When a file is written to dCache, the PoolManager selects a pool where it will be placed. Files in dCache may be *precious*, meaning they are the sole copy, which must not be deleted and is waiting to be transferred to tape, or *cached*, where the file may be deleted if the disk space is needed for other data, which may for example be the case when there is already a copy on tape, a second copy on disk or when the instance is exclusively used for caching. A file may be *pinned* for a certain, possibly unlimited time, in which it can not be deleted from disk. Precious files are pinned until they are flushed to tape, just like files that have been staged are pinned until they are no longer directly needed. Files are never actively removed from disk by dCache, but may be displaced when they are not pinned and newer data requires space, which is usually done according to the *Least Recently Used* (LRU) algorithm.

Flushing and restoring requests are treated differently within dCache. Files that are written to dCache and should be transferred to tape have an attribute called *storage class*. Each pool groups files in the flush queue by this attribute. Certain classes are then selected based on the maximum time they have been waiting, if the queue surpasses a certain summarized size or contains more than a defined number of files. Storage classes may be flushed concurrently, if configured. Based on the number of allowed concurrent flushes defined in the HSM configuration, a certain subsets of each selected storage class flush queue is passed on to the connected tape system, where the amount of tape drives then limits the number of concurrent transfers. This is illustrated in Figure 3.4 for a single pool. Because this is done by and for each pool individually, the selection is not coordinated, so that each pool may decide to flush different storage classes, leading to a mixed result on the tape system side, defeating the original purpose of ordering files before flushing. However, this works well under the assumption that there are not many storage classes or at least not attempted to be written concurrently.

The dCache system assumes that tape storage is primarily used for archival purposes and reading data back from the archive is only done in extreme situations. Therefore, while writing to tape is coordinated, reading requests are sent to the tape system directly without prior organization as shown in Figure 3.5. The to-be recalled files are not ordered by storage group, because it is assumed that only a single storage group is requested at a time as is usually the case and that requests arrive sequentially. With the tape carousel concept, active, batched recalls are desired, which may concern specific datasets from potentially different storage groups. Currently, dCache does not support the dataset concept, but even if it did, the staging process would need to be enhanced to make use of the information. Optimally, requests for files belonging to a dataset would be grouped and sent to the tape system, where the ideally well collocated files could be retrieved with minimal tape mount and seek overhead.

The way a dCache system is set up will also substantially influence the performance of accessing the connected HSM as well. Among the relevant factors are the number of stage and the number

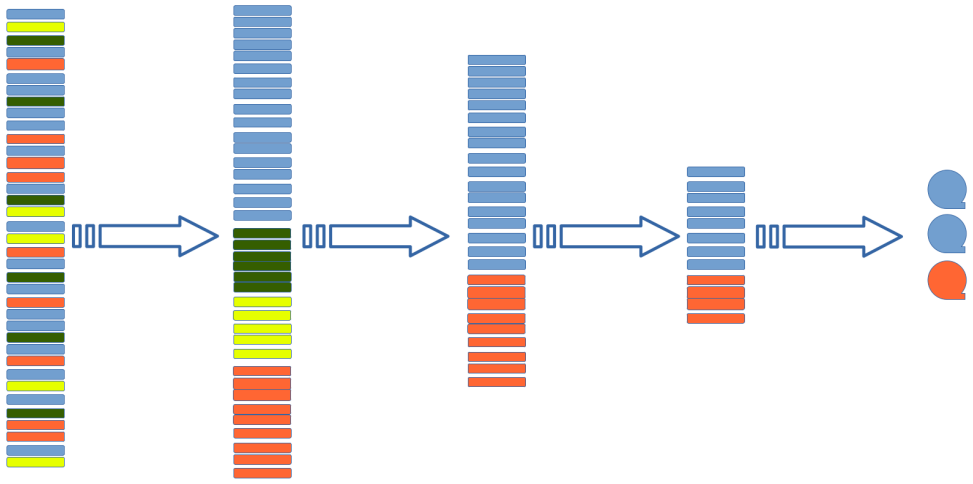


Figure 3.4.: The dCache flush queue. A pool sorts its queue by storage class, selects some by policy and sends the number of allowed concurrent flushes to the tape system.

Figure 3.5.: The dCache restore queue. Restore requests are sent directly to the tape system without prior sorting, files are staged accordingly.

of flush pools, their connection to the rest of the system, the physical separation and sizes of pools as well as the maximum number of parallel operations allowed. The number of flush or stage pools scale the number of possible parallel operations, but may further diminish the storage group organization and selection by each pool because they all make an isolated decision based on the local data. A central component may be imagined, which could coordinate the flushing and potentially also the restoring operations passed to the tape backed. Alternatively, a single, large pool may be responsible for HSM connectivity and coordinate the processes on its own. And at last, it needs to be considered that a tape system is a shared resource that may be used by several dCache instances or other services. It might otherwise be considered whether it is possible to optimize tape access by by-passing the internal scheduling done by the tape system that is currently treated as a black box by sending on individual requests instead of large lists. Therefore, it may even be more useful for the general case to have a system between storage element and tape system that can buffer and order requests more intelligently based on a larger, broader window and global knowledge. The storage elements would have to pass on everything immediately or the most relevant selection at any point in time. This is information that dCache currently does not have.

4. Optimizing the dCache Interaction with a Tape System

As analyzed in the previous section, the way that dCache interacts with a tape backend may be improved to allow for better restore performance. There are different approaches and aspects of relevance, most prominently the integration of the concept and associated knowledge of data sets, which are the main unit of processing and as such the primary concept underlying the data carousel request patterns. Likewise, the way that requests are scheduled and buffered within dCache needs to be addressed, while considering the complexity of connected tape systems. Even though the tape carousel aims to improve the recall efficiency from tape, smart writing will likely initially take up more time, but may later on lead to immense recall benefits. Therefore, the relation between reading and writing needs to be considered, weighed and priorities decided. The most efficient request patterns to retrieve data when it is optimally distributed on tape may differ, but can likely be transferred in parts from approaches to retrieving files under the current, more unstructured data distribution. The latter is of more acute importance and addresses a more general problem that does not make assumptions that would imply minimal recall request scheduling overhead for optimal performance.

4.1. Characterizing the Optimal Storage Chain

To isolate the dCache behavior, several of the problems affecting the tape system performance as described in Section 3.4, that are outside of its control, need to be reduced or ignored. In an ideal case, the tape system would provide data with the maximum throughput possible to dCache, while the upper layers would transport the staged files away as soon as they arrive, freeing staging space almost immediately. This assumes that files are collocated ideally on tape according to the order in which they are requested so that performance is maximized by minimizing the number of remounts, maximizing the amount of sequentially read data while making use of parallelization through reading from different tapes mounted in several drives. The logical grouping of files that underlie the recall request sequence, such as dataset affiliation, is propagated to the relevant layers. The tape system is assumed to be reserved exclusively for the dCache instance under investigation. In such a situation, dCache may be assessed as to the difference between currently provided and optimal performance, which would minimize the overhead of scheduling and management of requests assuming maximal data ingest rates from tape. Due to the caching nature of the storage element and redundant requests, this performance may even be better than the isolated tape system throughput.

In the example of the KIT Tier 1, of the 30 T10000D tape drives documented in Table 3.2, only a maximum of 12 are being used for ATLAS in the current reprocessing campaign. Assuming usually enabled compression, this leads to an overall performance of

$$12 \times 800\,\mathrm{MB/s} = 9600\,\mathrm{MB/s} = 9.6\,\mathrm{GB/s}$$

according to the tape drive characteristics shown in Table 2.1. This maximum performance is only possible in a period of sequentially reading from a mounted cartridge and excludes mount and seek as well as rewind times, which are realistically unavoidable. It requires that the stage requests are optimally ordered by the file collocation on tape and the network bandwidth is sufficient to immediately transfer the staged files from the tape system buffer to the destination so that the buffer may be reused. This maximum performance will never be reached in practice over longer periods.

Once the files have been transferred to dCache, the respective disk space is occupied, which is much smaller than the overall capacity provided by the tape system. If there are dedicated stage pools, the files are usually transferred to different pools on arrival to limit this bottleneck. Because these files have been staged in order to be reprocessed, they need to be kept on disk for a certain period of time, which is usually achieved by pinning them for a defined duration of a few hours, days or weeks. When the data is processed locally, this time needs to be larger than if it is intended to be done at another site, usually a Tier 2 or even 3. The latter is commonly the case, in which Rucio needs to realize that files have been staged and transport them to their destination via FTS as soon as possible so that the space may be freed again.

The optimal storage chain from the perspective of the dCache disk storage system, therefore, includes being able to buffer large numbers of requests that arrive in bursts rather than a trickling fashion so that subsets from a much larger list of read requests may be selected without relevant delays due to waiting periods. Furthermore, the tape system is assumed to be reserved for the dCache instance alone, from which just recall activities for the data carousel are conducted, no writing activities. It is expected to run without failing or malfunctioning hardware or software components and optimally schedule read requests according to the tape and on-tape location. The network between dCache and the tape system is considered to be infinitely fast and that Rucio/FTS will remove files from disk as soon as they are staged to dCache, removing the pins and allowing the data to be displaced. In this way, the dCache performance with respect to its employed scheduling strategies is isolated, influences of the other cooperating systems are minimized.

4.2. Smart Reading

The focus of the current data carousel campaigns is optimizing the recall efficiency of data that is already located on tape systems. The way it is organized there differs from site to site and dataset to dataset. Sites usually had no information on datasets and associated files at the time of writing. Because data is first written to the CERN local T0 instance and only afterwards copied to Tier 1s, the write requests arrive in bursts of few datasets which theoretically allow implicitly collocating the files on tape. However, the ATLAS data management organizes all data that should be written to tape in a single storage group, which in practice leads to sometimes wide distributions over large numbers of cartridges within that group. Organizing the read requests by storage group, as dCache does for writing per pool, possibly then coordinating all such restore pool choices, would therefore not improve the recall efficiency in the current situation.

To enable the tape system to maximally schedule tape read processes, its queue should be largely filled at any point in time when there are pending requests in dCache. The larger the known file read queue is with respect to the amount the tape system can accept, the selection criteria for subsets that are passed in which order on becomes increasingly relevant. To optimally do so, it is necessary to first analyze and make assumptions regarding the tape system scheduling and queuing to assess how to optimally access tape-resident data and discover which information is

of most relevance while easily acquirable. Finally, different scheduling options are proposed and formalized.

4.2.1. Tape System Queuing and Scheduling Assumptions

Sections 3.4 and 4.1 mentioned that the main read performance bottlenecks of tape systems are the number of cartridge remounts as well as the on-tape seek time for required files. The number of remounts is usually less time-consuming, but additionally significantly reduces the lifetime of cartridges, which are primarily intended to be used for archival storage that is rarely recalled. As cartridges are intended to be used by ATLAS for both long-term preservation as well as eventually active usage in data carousel endeavors, remounting is desired to be minimized for both reasons.

Figure 4.1 shows an exemplary queue of five read requests for two tapes. In the first case, they are read in the order they arrive (*First In First Out* – FIFO), which leads to five mounts and five unmounts, resulting in a total of fifteen operations. In the second case the requested files are grouped by tape and recalled by these groups. This strategy results in two mounts as well as two dismounts and 9 operations overall. Seek times on each tape are disregarded in these examples, but due to the small recall size of a single file the FIFO case is especially problematic, because a randomly positioned file on average requires winding the tape to half its length and back.

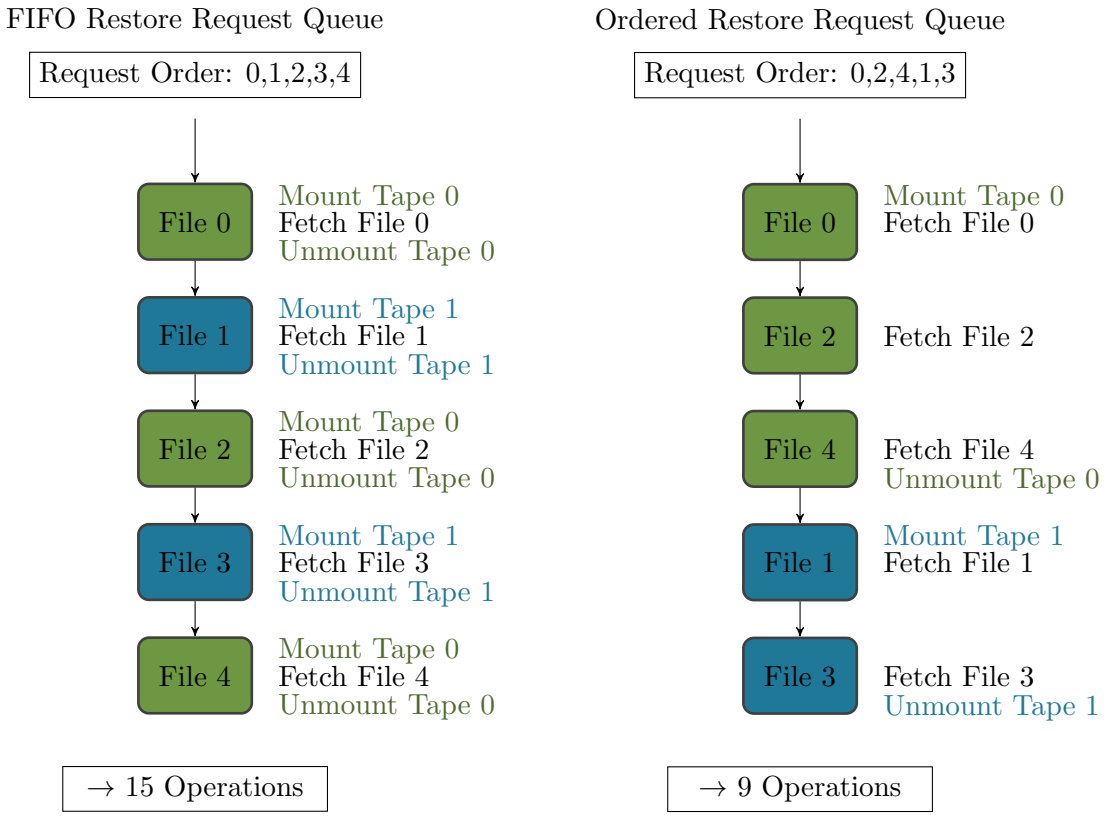


Figure 4.1.: Minimizing mounts and unmounts by read request ordering ([24])

If several files are recalled from a single tape, they should be ordered so that they can be read sequentially. Skipping between different positions, forwards and backwards, leads to longer overall seek times. Figure 4.2 illustrates the example of five files to be recalled from a single tape on which they are collocated. In the first case, they are accessed and read in the order that the requests arrive (FIFO), which leads to five seeks for the beginning of the next file. In contrast,

the second queue illustrates that only one seek for the first file is necessary when the requests are ordered accordingly before being retrieved. It is important to note that this ordering is not necessarily equivalent to the logical sequence of files on tape. Two files that are logically far apart may physically reside next to each other on parallel tracks and it may be more time-efficient to access them by shifting the read heads rather than in the logical order. Additionally, systems like TSM transparently aggregate files and chunk them into definable sizes before writing these to tape. This serves to avoid performance and data placement problems associated with small or large files. Although these chunks are usually collocated, this also makes it more difficult to make assumptions regarding data placement and optimal retrieval. The representation in figure 4.2 continues to be a good approximation, but is not to be mistaken for an accurate representation of seek optimization mechanisms in every tape system.

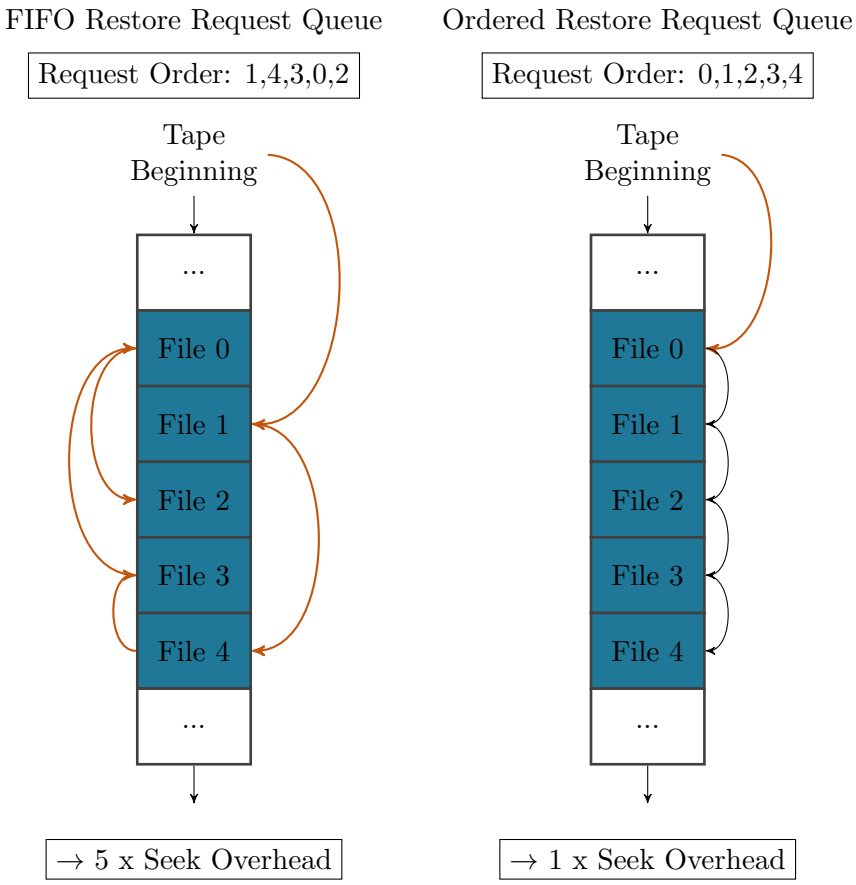


Figure 4.2.: Minimizing tape seek times by read request ordering

The way that tape systems organize and schedule their request queue differs widely. TSM, for example, only reorders requests within a single session by one client. To maximize the reordering done by the tape system, the requests should therefore not be sent in parallel via several dCache pools but be passed on by a single one, which then becomes a bottleneck in its own way. The OSM instance at DESY has separate queues per tape type, in this case different generations of LTO. Each queue includes read as well as write requests and has limits on the maximal length as well as physical size mainly relating to the files that should be written. A queue is sorted by tapes upon arrival of new requests and processed in a FIFO manner, looking ahead in the queue by a certain amount of time. Each tape has a maximum mount time in order to avoid infinite mounting and starvation of unpopular requests. In general, it may be assumed that tape systems aim to maximize the volume read from a single tape while minimizing idle times and request age.

It is therefore expected that a tape system is able to both sort by and on tape in a way that is optimal.

4.2.2. Optimally Accessing Tape-Resident Data

Empirical evidence collected at DESY indicates that 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 size of the particular file alongside 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. Within this general performance window for a single batch recall, the performance for recalling an individual file will again vary according to its size. The overall performance for reading a file f with a drive d will therefore be approximated with a linear equation of the form

$$\mathrm{p_d(pf)} = \begin{cases} (\mathrm{ap_{max}} - \mathrm{ap_{min}}) \cdot \mathrm{pf} + \mathrm{ap_{min}}, & \text{if } 0 \leq \mathrm{pf} \leq 1 \\ \mathrm{ap_{max}}, & \text{otherwise,} \end{cases} \tag{4.1}$$

where

$$\begin{aligned} \mathrm{ap_{max}} &= \mathrm{p_{dmax}} \\ \mathrm{ap_{min}} &= \mathrm{p_{dmax}} \cdot (1 - \mathrm{plc})\,. \end{aligned}$$

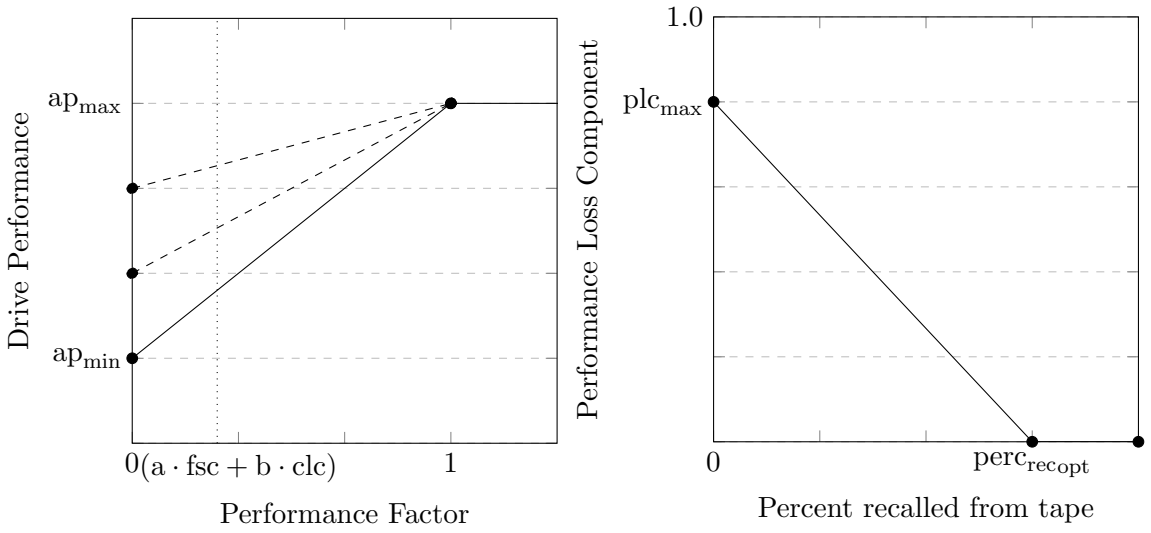
As is also illustrated in Figure 4.3a, the performance of a certain drive when reading a file f is $\mathrm{p_f}$, the sum of the adjusted minimal tape drive performance $\mathrm{ap_{min}}$ and the product of the performance component pc with the difference between the adjusted maximal and minimal drive performance. The adjusted maximal performance remains the maximal overall drive performance $\mathrm{p_{dmax}}$, whereas the minimal performance is a fraction thereof, calculated by subtracting a percentage of the maximal performance that will be referred to as performance loss component plc and is correlated to the percentage of data recalled from a cartridge. The plc may be approximated by a static linear equation as follows:

$$\mathrm{plc(perc_{rec})} = \begin{cases} -\mathrm{plc_{max}} \cdot \dfrac{\mathrm{perc_{rec}}}{\mathrm{perc_{recopt}}} + \mathrm{plc_{max}}, & \text{if } \mathrm{perc_{rec}} \leq \mathrm{perc_{recopt}} \\ 0, & \text{otherwise,} \end{cases} \tag{4.2}$$

with

$$\mathrm{perc_{rec}} = \frac{\sum_{i=0}^{|rec|-1} \mathrm{s_{f_i}}}{\mathrm{s_t}},$$

where $\mathrm{perc_{rec}}$ is the sum of all recalled files in the current session divided by the overall tape capacity and $\mathrm{perc_{recopt}}$ is the minimal percentage for optimal drive performance. The plc function



(a) Tape drive performance

(b) Performance loss component

Figure 4.3.: Tape drive performance approximation and performance loss component

is visualized in Figure 4.3b. The larger the percentage of data that is recalled from the overall compressed tape capacity, the smaller the plc will be, reaching 0 when the recall percentage perc_{rec} is at least as large as the optimal recall percentage $\text{perc}_{\text{recopt}}$. This in turn reduces the minimally possible drive performance from the maximum possible value and changes the associated gradient, as can be seen in Figure 4.3a. Recalling over 80 % renders the seek times and general overhead negligible and results in a plc of 0 and therefore causes the performance by files of every size to be the maximum a dive can offer. In order to never have an overall performance of zero, the maximum plc value, plc_{max}, must remain below 1, for example 0.9. The optimal performance of a drive for a particular file, therefore, is either possible if at least 80 % of the capacity is recalled from a tape, regardless of the file size, or for a particular file if the performance component *pl* is 1.

The performance factor pf is a weighted combination of the file size component fsc and the collocation component clc:

$$\begin{aligned} \text{pf} &= a \cdot \text{fsc} + b \cdot \text{clc} \\ 1 &= a + b \\ a &\ll b \end{aligned}$$

The constant factors a and b sum up to one, where is significantly larger than a. The factors fsc and clc are both valued between 0 and 1, a higher value indicating more optimal file size and collocation of the recalled list of files with regard to the resultant drive performance. The fsc may be calculated using the formula

$$\text{fsc} = \begin{cases} \frac{\text{s}_\text{f}}{\text{fs}_\text{opt}}, & \text{if } 0 \leq \text{s}_\text{f} \leq \text{fs}_\text{opt} \\ 1, & \text{otherwise,} \end{cases} \tag{4.3}$$

where s_f is the size of the file in question and fs_{opt} is the minimal file size required for optimal performance. The optimal file size is usually at about 10 % or less of the overall cartridge capacity, oftentimes already beginning at 10 GB.

The collocation component clc is more difficult to approximate. When file locations are only known by counted position, the physical distance between two non-neighboring files is not known. The clc may then be calculated by calculating the ratio of collocated files in the list over the total number, resulting in an optimal value of 1 when all files are adjacently located on tape. This may be described by

$$\begin{aligned} \mathrm{clc} &= \frac{c_{req_t}}{|req_t|} \\ c_{req_t} &= 1 + \sum_{n=1}^{|req|-1} \begin{cases} 1, & \text{if collocated}(f_n, f_{n-1}) \\ 0, & \text{otherwise,} \end{cases} \end{aligned} \tag{4.4}$$

where c_{req_t} is the number of collocated files and $|req_t|$ the overall file count.

Optimizing the performance of a single tape drive when given a list of requests for a single cartridge under the condition that the request queue is maximally filled results in choosing a subset out of the overall number of requests. Because the read performance cannot be evaluated for a single file alone but always depends on the entire chosen list, this optimization problem may be specified in the following way:

$$\begin{aligned} \text{Maximize } p_{1d}(x) &= \sum_{i=0}^{2^{|req|}-1} p_{s_i} \cdot x_i \\ \text{subject to } &\sum_{i=0}^{2^{|req|}-1} |s_i| \cdot x_i \leq |q| \\ \text{and } &\sum_{i=0}^{2^{|req|}-1} x_i \leq 1, \end{aligned} \tag{4.5}$$

where $|req|$ is the number of requests in the overall request list, p_{s_i} is the performance of reading all files in subset i of the overall request list and $|s_i|$ is the number of requests in the subset i. The x_i parameter is the inclusion factor, which may have values in the range [0..1]. The optimization problem consists of selecting at most one subset of requests out of the larger set of requests, so that the overall performance of reading the included files is maximized and the number of included requests is at most as large as the queue size $|q|$. The performance is maximized when the volume and collocation of the files in the selected subset is maximized. The number of subsets within a set of size n is 2^n. This number scales rapidly with an increase in the size of n, but may in practice be reduced by discarding sets that alone exceed the queue size limit and possibly those which have too small collocation values c_{req} by means of heuristics.

Most automated tape infrastructures make use of several drives in parallel, however. The performance of a multi-drive system is the sum of the performances of the individual drives, which may each work on a single cartridge at a time:

$$\mathrm{p_{n\_drives}} = \sum_{i=1}^{n} \mathrm{p_{d_i}(x)}. \tag{4.6}$$

Optimizing this overall performance may be formalized in the following way:

$$\begin{aligned} \text{Maximize } \mathrm{p_{nd}(x)} &= \sum_{i=0}^{|\mathrm{t}|-1} \sum_{j=0}^{2^{|\mathrm{req_i}|}-1} \mathrm{p_{s}}_{ij} \cdot \mathrm{x}_{ij} \\ \text{subject to } \sum_{i=0}^{|\mathrm{t}|-1} \sum_{j=0}^{2^{|\mathrm{req_i}|}-1} |\mathrm{s}_i| \cdot \mathrm{x}_{ij} &\leq |\mathrm{q}| \quad , \\ \text{and } \mathrm{n} \geq \sum_{i=0}^{|\mathrm{t}|-1} &\begin{cases} 1, & \text{if } \exists j : \mathrm{x}_{ij} = 1 \\ 0, & \text{otherwise,} \end{cases} \end{aligned} \tag{4.7}$$

where $|\mathrm{t}|$ is the number of cartridges for which requests exist, $\mathrm{req_i}$ the size of the request list for cartridge i and $\mathrm{p_{s}}_{ij}$ the performance for reading subset j on tape i. The x_i parameter is the inclusion factor, which may have values in the range [0..1]. The last line from Equation 4.7 states that at most one subset may be selected for each cartridge, the line above that the sum of requests within all selected subsets needs to remain below the tape system queue length $|\mathrm{q}|$.

This is a variation of the well-known 0-1 *knapsack problem*, which describes an optimization problem where a number of items i, each with a weight w_i and value v_i, need to be selected so that the value is maximized and the weight remains below a certain limit W. In the 0-1 version, each object may at most be included a single time. It is formally defined in the following way:

$$\begin{aligned} \text{Maximize } & \sum_{i=1}^{n} \mathrm{v}_i \cdot \mathrm{x}_i \\ \text{subject to} & \sum_{i=1}^{n} \mathrm{w}_i \cdot \mathrm{x}_i \leq \mathrm{W} \text{ and } \mathrm{x}_i \in \{0, 1\}. \end{aligned} \tag{4.8}$$

In the variation at hand, the items are the subsets, their value is the performance of reading the files requested in the subset, their weight the number of the contained items and the weight limit W is the queue size $|\mathrm{q}|$. This problem imposes the additional condition, that the subsets are clustered in larger sets according to the cartridges they target, that at most one subset per cartridge set may be selected and overall not more than there are tape drives, ideally exactly as many. The decision form of the knapsack problem, which consists of deciding whether there is a selection of items so that a certain overall value V may be reached, has been shown to be NP-complete, so that a solution is verifiable in polynomial time but finding a solution is not. The optimization problem itself is NP-hard, meaning that no algorithm exists, that allows finding or verifying a solution in polynomial time. However, using dynamic programming, pseudo-polynomial time algorithms exists, which allow finding an optimal solution for small problem instances [43].

The variation under consideration requires choosing the optimal combination of at most $|\mathrm{q}|$ subsets from $|\mathrm{t}|$ groups and selecting at most a single set from each group. A brute-force approach

would require the consideration and evaluation of all possible combinations. This problem is still NP-hard. Considering that the creation of subsets within groups as the basic elements that are then needed to be selected from is already complicated, the effort to find an optimal solution gains a further level of computational complexity.

Reducing Complexity This problem is very complex and does not lend itself to be solved optimally, especially when dealing with large lists of tens to hundreds of thousands of requests that need to be selected from and ordered, as is the case in this context. Therefore, approximate approaches need to be considered, that result in good absolute performance characteristics and offer distinct improvements over the current approaches. By considering the concrete circumstances of request arrival, storage and current sorting, assumptions and simplifications may be made that could lead to significant improvements.

Unfortunately, read requests are not sorted by some HSM systems and passed on based on criteria that rarely positively affect the recall performance, such as request age, resulting in even smaller percentages being recalled per tape and drastically increased numbers of tape remounts. The dCache system does not currently sort requests with the aim of improving the recall performance. In order to mitigate the number of remounts, some setups enforce a minimal time span such as 2 h before a tape may be mounted again. This may in turn adversely affect the queue processing if it is too large.

In the ATLAS reprocessing campaigns, files are recalled by dataset. The contents of each of which have been requested to be written to tape in approximately the same time period, but potentially alongside other data and may not have arrived at the destination site all at once. Even though writing is usually prioritized over reading, they are often distributed over several cartridges. Thirty or more datasets might be recalled at the same time, each of which containing about 13,000, but up to 55,000 files (see Figure 3.2). Because a dataset is on average 30 TB large and most used cartridges offer compressed capacities of 10 - 15 TB, datasets are spread over cartridges by necessity, even in the optimal case. If datasets were maximally collocated, they could be retrieved with maximal performance for the respective drives. However, because the tape system queue length is limited, requesting this way would result in recalls of maximal performance by a few drives and idle time by others, which might be less in combination than if each would recall small percentages but be constantly busy. This gain in performance will be at the cost of increased numbers of remounts. In this way, sub-optimally collocated datasets might inherently keep more drives occupied, but the percentages recalled likely vary largely, which might have an adverse effect. When requesting a subset of the data required from a single cartridge, this should ideally be collocated rather than spread over the entire tape.

The queue length in general needs to be treated as the limiting factor and its effect on different scheduling approaches should be investigated accordingly. In order to avoid request starvation, a certain FIFO adherence should possibly underly the new scheduling approach. Additionally, because it is assumed that requests arrive roughly clustered by dataset, this approach will likely ensure completely staging individual datasets as soon as possible, so that they can be used in analyses and removed from disk again as fast as possible, allowing new data to be staged. By sorting all currently known requests by tape and offset thereon, the offset could potentially be neglected and the mere summed volume and number of requests per tape considered. Then one may simplify the above performance approximation formula to

$$\mathrm{p_{df}}(\mathrm{s_f}) = \begin{cases} (\mathrm{ap_{max}} - \mathrm{ap_{min}}) \cdot \frac{\mathrm{s_f}}{\mathrm{fs_{opt}}} + \mathrm{ap_{min}}, & \text{if } 0 \leq \mathrm{s_f} \leq \mathrm{fs_{opt}} \\ \mathrm{ap_{max}}, & \text{otherwise,} \end{cases}, \tag{4.9}$$

which only depends on the overall recall volume and the size of the file in question. It simplifies the optimization formula for a single drive to selecting the first |q| files from the cartridge that is targeted by the oldest request, where |q| is the queue size (see Equation 4.5). If a large percentage of a tape's content is requested, it might alternatively be beneficial to incrementally submit all requests before starting to access another cartridge. Transferring this approach to a multi-drive setup, the question how many requests to submit for each cartridge remains difficult. Assuming there are n drives and requests targeting at least n cartridges, requests for at least n cartridges should be in the queue at any point in time in order to not let any drive be idle. Especially in small queues an equal distribution seems useful, possibly even trying to request similar volumes, because differences in requested volume will be unlikely to lead to beneficial performance peaks in certain drives that compensate the decrease in others. When the tape system request queue is large enough, however that all of the requests for a significant number of the selected |q| cartridges would fit in the queue with enough room for additional requests, they should potentially be submitted entirely, the unclaimed request slots filled equally be requests for the remaining cartridges.

Concretely, when the queue may on average contain all requests for as many cartridges as there are drives, requesting a single cartridge in parallel may be optimal. When the queue may on average contain one or less, requesting equally distributed according to the number of drives is likely optimal. Any queue length in between may be filled with equal selections of requests for the number of drives tapes, which would be at the cost of increased numbers of remounts. Because it is more performant to read a certain volume from a single cartridge than distributed over several cartridges and thus drives, the number may be reduced. An iterative approach is likely useful, wherein the overall performance that may be achieved by reading the maximum number of cartridges that may fit into the queue is considered, the performances calculated according to the simplified formula in Equation 4.9, the remaining queue slots filled for the idle drives and their performances added as well. Then the number of drives is iteratively reduced and the selection resulting in the maximum overall performance is selected.

4.2.3. Acquiring Information on Data Distribution on Tape

The information regarding which requests target files on the same cartridge and their offset of tape is of crucial importance to making any form of scheduling optimization. As a further layer of abstraction, this information is hidden from the HSM storage before the tape system, which requests files by the file identifier, called *bitfile ID* (bfid), returned by the tape system upon writing. Because it manages the mapping of bfid to physical location, it is possible to query this information from most tape systems, although the format and type of information may vary. It is undesirable to store a copy of this information in dCache, duplicating the required storage space and management overhead. An alternative, if it is sufficiently performant, is querying the tape system for the relevant information before scheduling and submitting requests. This might take the form of an external component, that accepts a list of file identifiers by dCache and returns this list enriched with the corresponding tape ID and offset thereon for each entry in a unified format. It would need to be able to interact with the API of different tape systems and convert the returned information. dCache might then cache this information for files in yet unsubmitted requests and make use of it when making scheduling decisions. The number of allocated drives should be known as well, in order to know how many different tapes should ideally be requested concurrently, which may be difficult due to opportunistic drive usage. However, slight over-committing might not have a large impact on the resultant performance and number of remounts, which should be investigated.

4.2.4. Scheduling Criteria and Evaluation Approaches

The read requests sent to a site are grouped by dataset. At the destination these are usually not known in advance and may vary in volume, file count and number of concurrent datasets. Because the requests arrive in a dCache instance over time and have stochastic components both in time and regarding the order of arrival due to inherent data transport unreliabilities, the online problem of bulk request scheduling for the tape system cannot be statically optimized.

Evaluation Criteria The goal of evaluating different scheduling strategies is to minimize the number of remounts and request wait times while maximizing the overall read performance. The most important metrics for assessing the value of an algorithm are therefore the following:

- Throughput per drive
- Throughput overall
- Stage request answer times per requested file
- Accumulated number of tape remounts in period

The performance is of primary importance along with the answer time per request, while the number of tape remounts is primarily an important consideration factor in deciding on a long-term viable strategy that will not damage the storage media by excessive usage. Additionally, it may be of interest to know the distribution of recalled percentages of the overall compressed cartridge capacity

$$\mathrm{perc}_{\mathrm{rec}} = \frac{\mathrm{s}_{\mathrm{rec}}}{\mathrm{s}_{\mathrm{t}}},$$

where $\mathrm{s}_{\mathrm{rec}}$ is the size of the recalled data and s_{t} the size of the tape capacity factoring in the compression level, as well as the distribution of cartridge mount durations in order to better understand different behaviors.

Scheduling Approaches to Evaluate Any scheduling approach that is evaluated should perform better in most categories than submitting random requests. As such, random request scheduling should be performed in every setting in order to establish a baseline. We define *ordered requests* to mean a sequence of complete datasets the way they are initially requested. It is reasonable to assume that any additional entropy in the requests such as reordering during transfer and resultant unclear separation of datasets negatively influences the results of subsequent scheduling and submission endeavors due to incomplete information. This factor may not be influenced at the level of storage systems. In evaluating different scheduling approaches, an optimal request order will be assumed to isolate the achievable improvements under optimal conditions.

The access patterns to be evaluated are therefore the following:

- *Random*
- *By dataset*; FIFO of ordered request list
- *By tape*, sequentially by tape occurrence in ordered request queue
- *By X tapes in parallel*, where X may vary to find the optimal number

By tape means iteratively submitting requests for all files to be recalled from a single cartridge to the tape system, then moving on to the next cartridge. Therefore, the relevant files from as many tapes are fully requested, as the queue allows entries. When requests for fewer cartridges are submitted than there are tape drives, the superfluous drives will likely be idle, unless another drive has to unmount a cartridge due to maximal mount time violation. *By X tapes in parallel* denotes submitting requests for the first X tapes *by tape* in parallel, each requestor moving to the next cartridge when all requests for the current one have been submitted. In contrast to the first approach, if all requests for these X tapes do not fit in the tape system request queue, it is filled to equal parts with requests for each cartridge and additional ones, where all requests for one tape were submitted while there was still space in the queue. If X is equal to the number of drives, it is assured that no drive will be idle, but the performance may be low for all drives depending on the size to be recalled per tape. The smaller the request queue, the more difficult it will be to achieve a good overall performance. Depending on the amount of drives, it may be ideal to submit all requests for a few tapes completely while adding partial lists in order for the other drives to no be idle or completely distribute the number of requests over all drives, which is to be investigated.

Evaluation Settings Evaluating different scheduling strategies is dependent on the tape system behavior, as well as certain configurable properties therein. Most notably, the number and performance of tape drives as well as the request queue size are of fundamental importance. Because these parameters are related, it is reasonable to assume one of them to be static and evaluate the influence of different sizes of the other. According to the assumptions made in Segment 4.2.2, optimal performance may be reached if a large percentage of a tape is recalled. Therefore, the optimal request list size also depends on the capacity of cartridges, the assumed compression level of contained data and the subset that needs to be read, as well as the spread of data to be recalled over different cartridges. It is therefore more sensible to vary the queue size and keep the number of drives at a reasonable amount. Consequently, the different scheduling strategies should be evaluated according to the selected evaluation criteria in the context of different tape system queue lengths to assess the varying benefits and make predictions regarding the optimal combination of setup and scheduling as well as limitations under sub-optimal conditions.

4.3. Introducing the Case Study KIT

The currently running data carousel exercise, which started on the 21.01.2020, is linked to a real reprocessing campaign that seeks to recall and analyze the entire data from the LHC ATLAS Run 2, which encompasses 18 PB that were written between 2015 until 2018. Every site will be requested to stage a share of this data relative to their overall pledges, taking into account the distribution of datasets over sites and that the Tier 0 CTA instance is included with a share of about 4 PB.

KIT is the third largest Tier 1 site along with CCIN2P3, which each manage 13 % of the global disk and tape resources for the ATLAS collaboration as shown in Table 3.1. KIT has agreed to cooperate with the dCache development team to help analyze the system's performance in the context of the data carousel exercise. Their tape connection setup is illustrated in Figure 4.4. They have one primary and one secondary store and likewise restore dCache pool for ATLAS with 130 TB for the primary and 100 TB for the secondary ones, each. This is a relatively simple setup from the perspective of dCache, which on the one hand introduces the bottleneck of using a single machine for staging operations, but at the same time allows for easier scheduling of the

requests passing through, without having to coordinate the behavior of several restore pools. The KIT setup has recently moved from using a common HSM script with dCache before TSM to the *Efficient Nordic DCache Interface to TSM* (ENDIT) system developed by the *Nordic e-Infrastructure Collaboration* (NeIC). It was created to efficiently use a TSM controlled tape system with dCache and has been used by the NDGF (NeIC) Tier 1 center for over a decade. It can parallelize reading by opening a separate stream per cartridge based on cached tape system knowledge and throttle request submission for trickling reads and prevent tapes from being remounted more than every two hours. The dCache plugin system is much more scalable than the HSM script interface allows and ENDIT has successfully been tested with over 100,000 outstanding read requests and reached peak performances of up to 1 GB/s. However, dCache insists on reserving space for all files that are requested from tape storage in order to be able to store them upon arrival, which limits the number of restores to a small subset of the there known requests. Nevertheless, this change allowed KIT to increase their restore queue size from a small length of 2,000 to 30,000 at the beginning of this year. In the KIT setup, the dCache provider plugin for ENDIT shares an IBM Spectrum Scale (GPFS) file system with the ENDIT TSS system, which communicates with a TSM storage agent that manages storing and restoring from the physical tapes.

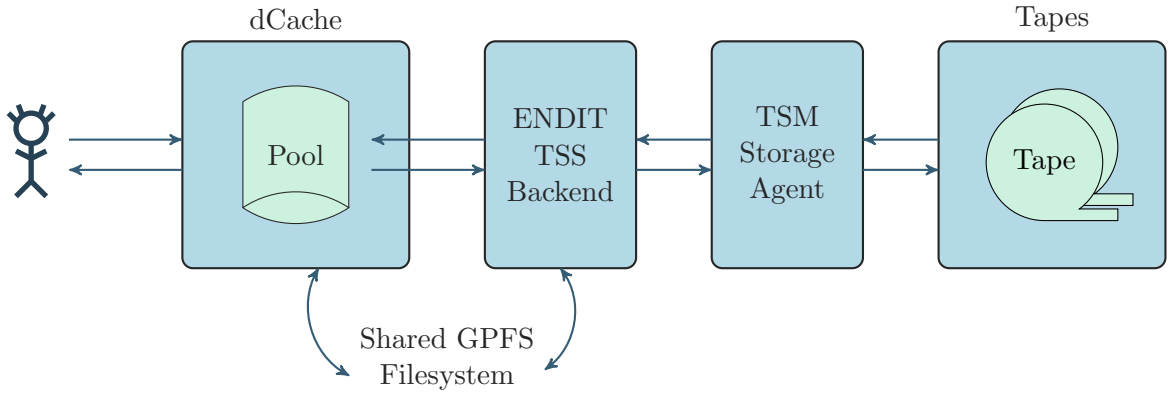


Figure 4.4.: KIT tape setup: dCache pools are located on Spectrum Scale File System (GPFS), it communicates with the ENDIT-based TSS Backend via a corresponding dCache provider plugin, which is in turn connected with the TSM storage agent that addresses the actual tape storage

In order to better understand factors impacting the KIT site performance in the current tape carousel exercise, it is important to consider the characteristics of the data that is recalled. The ATLAS collaboration has declared which datasets would be recalled from each site before the beginning of this exercise along with the included file counts and total sizes. Overall, 35 datasets will be recalled from KIT, which amount to a volume of 1.1 PB in 495,049 files that have been written in the years between 2016 and 2018. The boxplot in Figure 4.5 shows the distribution of file counts per dataset, averaging at about 14,000 but ranging between a few hundred and over 40,000. The latter would not even fit in the newly increased queue size length. Plot 4.6 illustrates the distribution of dataset sizes, which range from 2 TB to over 113 TB and average at 39 TB. The average file size is therefore about 2.2 GB.

Because the dataset name is encoded in the file paths, KIT was able to provide a list of associated files with their respective sizes and locations on tapes. Their tape system uses up to 12 drives of type T10000KD, which offers performances of up to 252 MB/s natively or up to 800 MB/s on compressed data (see Table 2.1). The data that will be requested resides on 330 different corresponding cartridges, which are then able to store 8.5 TB of raw data or more, depending on the compression level. Having calculated the maximal necessary tape capacity for storing the

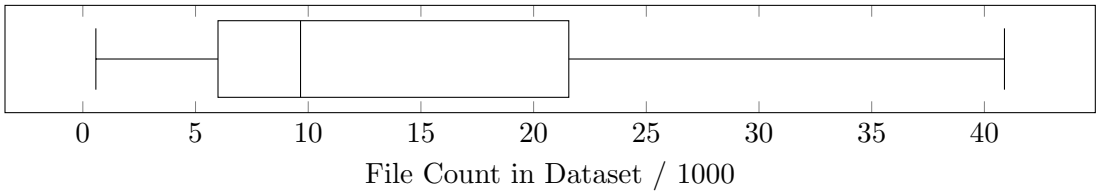


Figure 4.5.: File count per dataset that will be recalled from KIT (ATLAS reprocessing campaign January 2020)

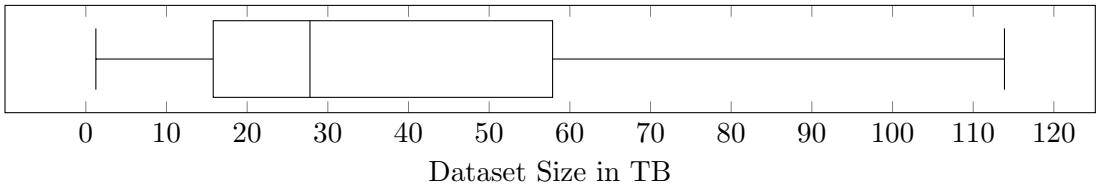


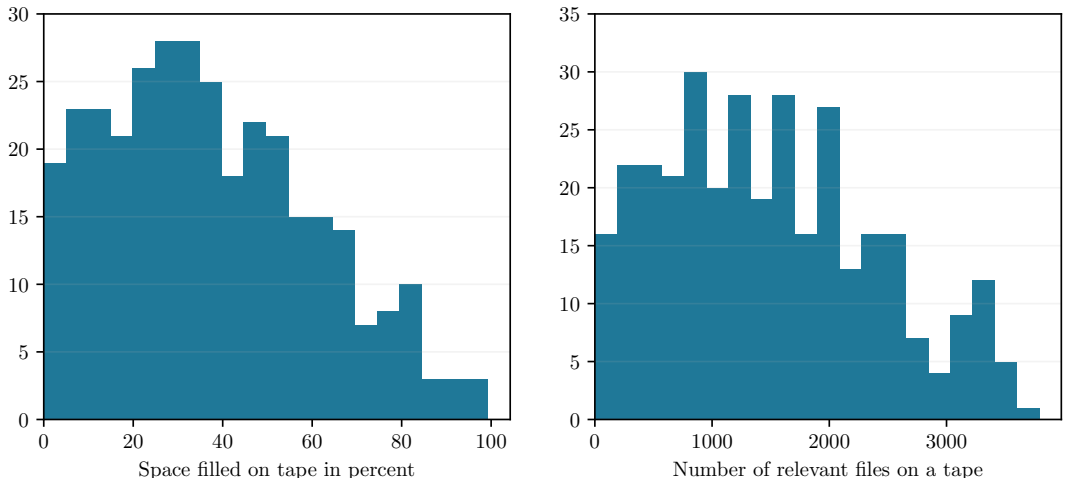
Figure 4.6.: Size per dataset that will be recalled from KIT (ATLAS reprocessing campaign January 2020)

files, the compression must be at about 1.3:1, which is assumed overall, but is likely higher, but might also be lower occasionally. Figure 4.7b visualizes the number of files that are to be recalled per tape, which are between 500 and 2500 for most cartridges, approximately 1500 on average. This is especially relevant for queuing and recalling data in parallel by requesting multiple tapes at the same time. Figure 4.7a shows the percentage distribution to which the 332 used tapes are filled by currently relevant ATLAS data. Recalling everything that is desired for the current reprocessing campaign would in most cases mean reading less than 60 % of each tape volume, assuming higher compression levels even less. The average relevant data volume is 37.5 %. This is less than optimal, enforcing large seek times. Additionally, most datasets are themselves spread over different tapes, as shown in Figure 4.8a. Some of this distribution is by necessity because the respective datasets are too large to fit on a single cartridge. However, even the largest dataset of about 113 TB would fit on

$$\frac{95TB}{8.5TB \cdot 1.3} \quad \approx \quad 8.5 \text{ Tapes},$$

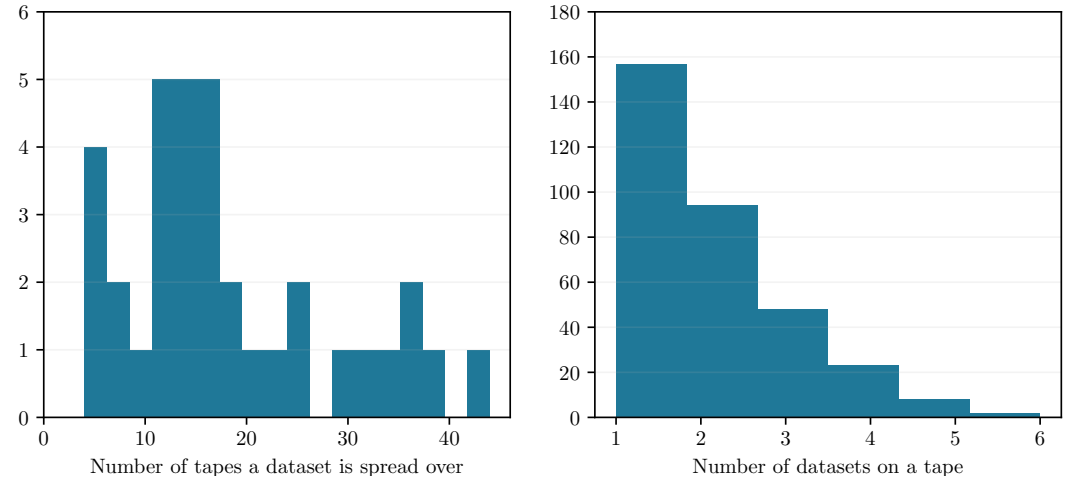
but most are spread over 10 to 15 cartridges, one even over more than 40. This is shown in more detail in Figure 4.9, which breaks down the file count per dataset into their location on different cartridges. Figure 4.8b shows the distribution of the number of datasets that are at least partially on one of the 332 cartridges. While most of them contain one to three, some contain six different ones.

This detailed dataset and file distribution information by the KIT Tier 1 site is an important guidance to understanding performance problems, analyzing the contribution that a suboptimal recall request scheduling method might have and evaluate more appropriate approaches.



(a) Recall data volume per tape (b) Number of files recalled per tape

Figure 4.7.: The volume of ATLAS data for the current reprocessing campaign on 332 KIT tapes, as well as the associated number of files. The overall fill level of the tapes is unknown, but assumed to be maximal. The data storable on the used T10000D cartridges is calculated assuming a compression level of 1.3:1 to be able to fulfill these ATLAS data storage requirements.



(a) ATLAS datasets spread over tapes (b) Dataset count on each of the 332 KIT cartridges

Figure 4.8.: The number of tapes a dataset is spread over as well as the number of datasets per tape

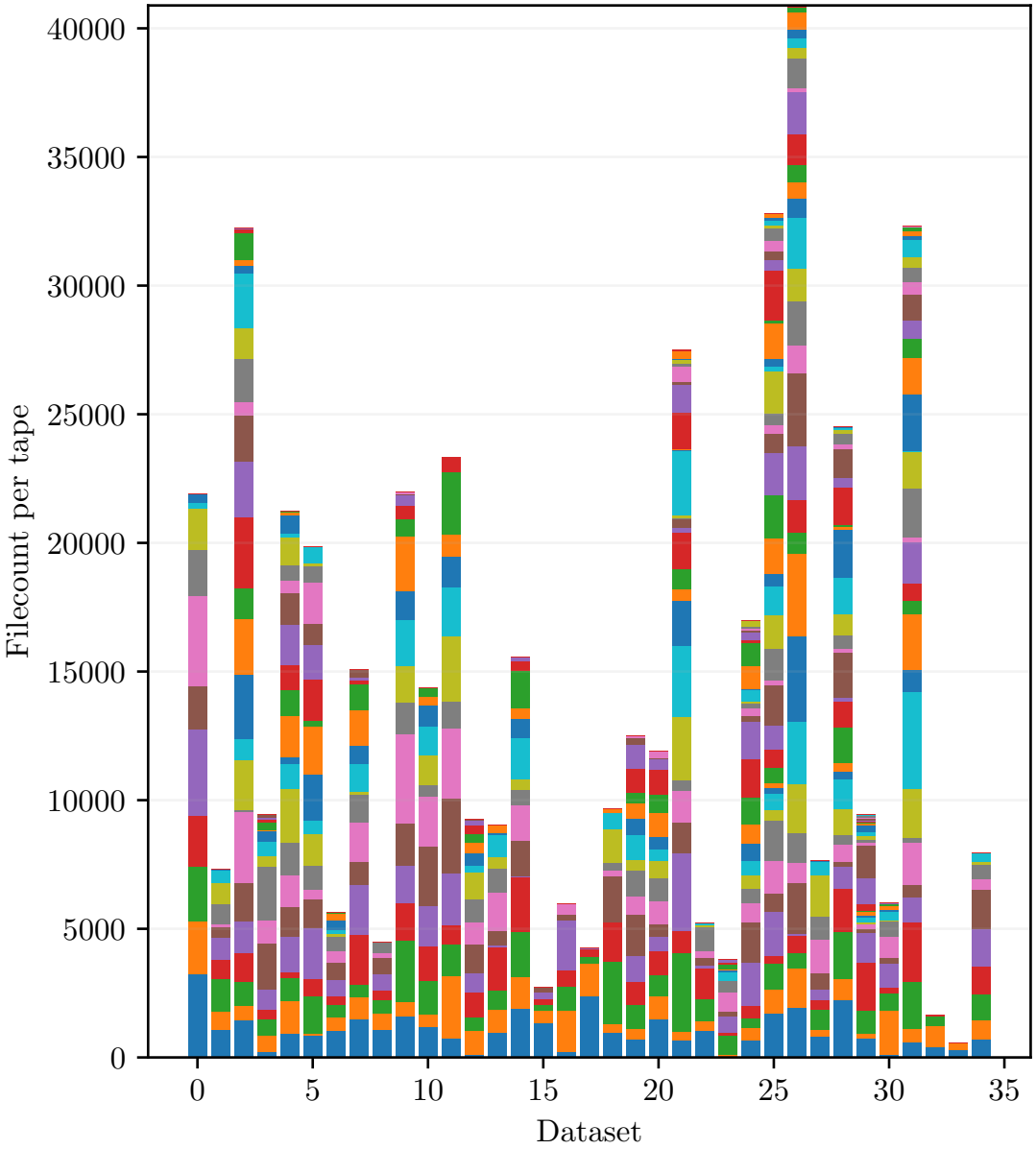


Figure 4.9.: File count per tape for each of the 35 datasets, which are ordered by descending age

5. Simulating the dCache Tape Interaction

Because of the complexity of this problem, an optimal scheduling solution cannot be purely devised on paper. Different strategies can also not realistically be evaluated using a physical tape infrastructure. One or two drives may be reserved along with a few cartridges, but analyzing how an approach scales with increased resources and data volumes is not possible. Additionally, testing with real data and hardware requires a lot of time. Therefore, a simulated environment that is capable of sufficiently reproducing the behavior is desirable, which is capable of simulating large infrastructures and data volumes in a short period of time and collect monitoring data on the metrics of interest. Although tape emulators exist in the form of VTLs, which do not fulfill the desired criteria, no tape simulator implementation seems to exist that is publicly available. Therefore, a limited tape simulator was developed for evaluation purposes in the context of this thesis.

5.1. Discrete Event Simulation

Simulation is an approximate imitation of the behavior of a system or operation. It is used in a wide range of fields to understand the behavior of the target under different conditions. In order to create a simulation, a corresponding *model* is created as a set of assumptions regarding the operation of the system. As such, a simulation will be abstracted, idealized and may even be 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. *Emulation* in contrast is used to completely replicate the behavior of a system and model it at a structural level. It is primarily used in the field of computing and electronics, where the processes of a certain hard- or software may be reproduced in a different environment. Like simulation, it may differ in the internal realization compared to the original process, but an emulation needs to strictly adhere to all of the rules and interfaces of the original system. A virtual tape library is an example of an emulator for a tape library.

Discrete Event Simulation (DES) is a form of simulation that models system behavior as a discrete sequence of well-defined events. Each event marks a point in time that is associated with a specific change in system state, which is assumed to not occur between events. Discrete event simulation works at a higher level of abstraction compared to continuous simulation in which the system state is changed continually and is therefore usually less computationally expensive. The *fixed-increment time progression* models time as being broken up into small fixed-sized chunks, whereas *next-event time progression* allows jumping directly to the occurrence time of the next event, which is generally faster because not every time chunk has to be simulated.

5.2. Objectives and Approach

The primary objective of simulating a tape system in this context is evaluating the recall efficiencies of different request scheduling approaches over certain periods of time in the context of the ATLAS experiment data carousel exercises. Different factors affect how a tape system at

a site behaves. First of all, recall requests are submitted to a site by datasets and arrive with varying amounts of entropy at the destination. They are submitted to the local storage system chain, where, after passing through dcache, they are sorted according to the scheduling algorithm to be evaluated. They arrive in chunks in the tape system queue over time, where they are reordered by tape, selected by drives and finally staged to a disk buffer. The tape infrastructure differs from site to site, both in terms of the capacity and performance parameters of the physical hardware, the scheduling mechanisms of the tape system software, the request queue and buffer sizes as well as the network connections. The most important factor that influences the recall efficiency, however, is the distribution of the requested data over and on cartridges. The KIT Tier 1 has kindly provided data on their infrastructure as well as data distribution, which has been introduced in segment 4.3, making it an optimal case study to reproduce and analyze.

Goals Because the goal is limited to evaluating recall efficiencies, the simulator to be developed does not need to universally recreate a tape system, model different hardware and software components or be otherwise fully modular. The main aspects that need to be modeled are the following:

- Set of tapes with tape ID, a certain capacity and reserved space
- List of files with name, size and dataset association
- Mapping of tape to files contained thereon with their respective offsets
- The tape system accepting read requests for files when space is available in the queue, generating timestamped request items
- The tape request queue being sorted upon arrival of new requests
- Tape drives of certain maximal performance selecting requests for the best not yet mounted tape
- Drives mounting/unmounting tapes, calculating read performances and durations, removing served requests from the queue, fetching further requests
- External request generation and submission to the tape system

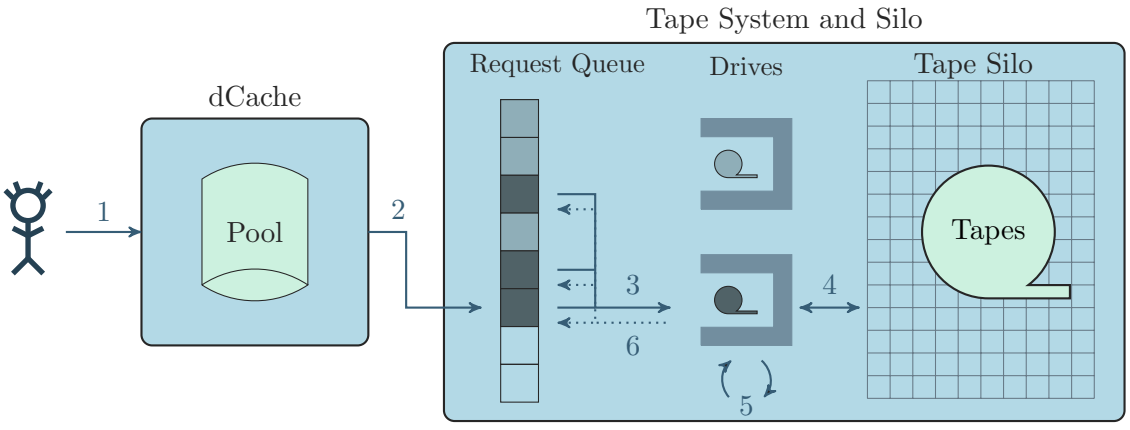


Figure 5.1.: Tape system simulation behavior overview. (1) Client submits requests to dCache, (2) subset gets sent to the tape system request queue if there is space left. (3) A drive is assigned the next most relevant cartridge and associated requests. The cartridge needs to be mounted in the drive. (4). The requested files are read (5) and finally removed from the queue (6).

Figure 5.1 show the path a request takes through the simulated tape system. When a list of files is requested by a user from dCache (1) which is not located on disk, the list and all other remaining requests within dCache get reordered based on one of the scheduling methods to be evaluated and a subset is submitted to the tape system request queue when there are enough free slots available (2). The dCache system does not need to be modeled in any detail in this approach, where the focus is placed on the tape system response to different submission patterns. As a requestor like any other from the perspective of the tape system, only the resultant succession of request submission is of relevance. When requests have been added to the tape system request queue, it gets reordered based on tape location. When one of the defined drives is idle and there are requests in the queue, the next best cartridge that is not yet mounted by another drive is assigned to a drive with the associated requests (3). If another tape is currently mounted, it needs to be replaced (4). The drive then reorders the requested files by location on tape and reads them, calculating the required durations (5). Finally, the processed requests are removed from the request queue. It should be noted, that neither network latency, nor buffer space for read files are taken into account. This model purely takes into account the tape system performance with the understanding that limits on network bandwidth and the client performance when fetching the staged files from the buffer, in this case dCache, will further decrease the resultant performance. But since the goal is to maximize this performance, the implementors need to ensure that the surrounding infrastructure is able to make use of it, which is outside of the scope of this thesis.

Approach and Tools The simulation model includes multiple actors that perform jobs in parallel and access shared resources. They create events and act on others when they occur. A such, the use of discrete event simulation with next-event time progression as introduced in Section 5.1 is useful, because not every time step needs to be simulated when no event occurs. This might for example be the case when no claimable requests are in the tape system queue or all drives are busy, which is frequently the case.

There are many tools that support creating discrete event simulations. Of the open source frameworks, the process-based DES framework *SimPy* [40] was chosen because it is not specific to a certain level of abstraction or field, because of its simple to use syntax, excellent documentation and community support as well as the good maintenance. It is based on the standard Python language and focuses on providing functionality for process interaction via Python's generator functions and the management of several types of resources to simulate capacity congestion points.

5.3. Simulation Model

The simulator consists of active components and passive carriers of state. The active components, such as requestors and tape drives, read and modify information from state objects such as the file index, the tape silo containing cartridges and file locations or the tape system request queue. Certain operations on shared resources need to be atomic. These include read operations such as selecting a cartridge to mount, because different drives should not make and follow through with the same decision, but also write operations such as two different clients adding x requests to the tape system request queue concurrently, when there is only space left for X and not $2 \cdot X$ requests. Therefore, access to resources needs to be guarded. In the following, the most important state objects and active components are introduced, their interactions described.

5.3.1. File Index and Tape Silo

For the goal of simulating read requests, the components file index and tape silo will only be accessed for reading after their initialization. The file index contains a set of files with associated name, size, dataset and storage group association. Due to the current ATLAS data management situation, all tape-resident data is located in the same storage group. The files may also be accessed by dataset or storage group. The tape silo contains a list of tapes with associated identifier, storage group, configurable maximal capacity and remaining available space. A universal compression level is assumed that increases each tape's capacity accordingly. The tape silo also manages the mapping of file identifiers to tape ID and vice versa, which affect the remaining capacities. Additionally, the offset of each file on its containing tape is stored.

Files of different storage groups and their distribution over cartridges may be generated automatically at random, assuming a normal distribution of file sizes. This functionality might be extended to analyze different distributions. Primarily, it is possible to read in the KIT file distribution data, which generates file index entries with file size and dataset association, creates cartridges according to the associated tape IDs, stores a given logical file position offset. KIT uses tape drives. This data allows to calculate a minimally necessary compression level. The initialization of this tape system state takes approximately 5 s.

5.3.2. Tape System

The tape system primarily manages the request queue. It contains a file index and a tape silo instance. Additionally, it keeps track of which tapes are currently mounted.

Request Queue When requests are attempted to be submitted, which is a guarded function that may only be entered by one client at a time, it is ensured that there is enough space left in the queue before accepting. Each request for reading a file is converted into a timestamped internal request object with a generated unique identifier. This in consequence allows multiple requests for the same file to exists without triggering deduplication efforts. This is usually also the case in real tape management software because it cannot differentiate between the origin of requests, which might be different users. However, a dCache installation already deduplicates multiple requests for the same file before passing them on to a tape system instance, so that redundant requests should not occur in these simulations. Upon addition of these requests to the main queue, a second list that is sorted per tape is maintained as well to ease decisions regarding the processing sequence. Only a single process is again allowed to remove requests from the queue concurrently.

Selecting which Cartridge to Mount Usually, request queues in tape systems roughly follow a FIFO order so as to maintain a level of fairness and prevent request starvation. When a drive requests the selection of the next tape for mounting, the candidates are ordered by the age of the oldest request that target them. The one with the oldest request wins, provided that it is not currently mounted in any drive, except for the requesting one, under the condition that the maximal mount time has not been exceeded. Only a single process may access this function concurrently.

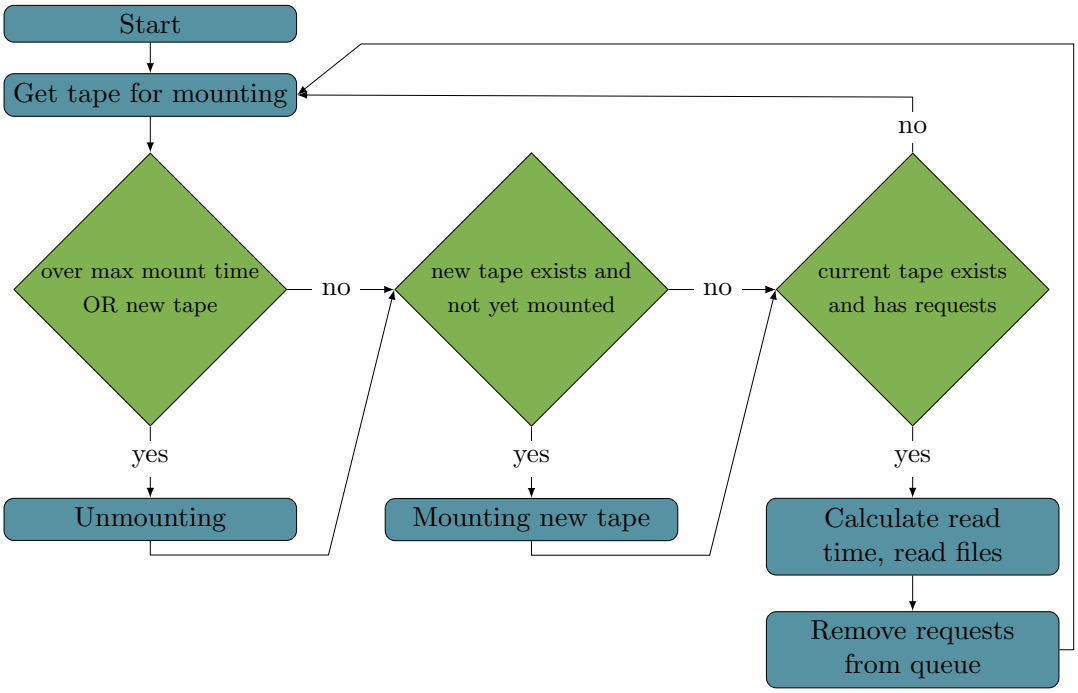


Figure 5.2.: Tape drive workflow

5.3.3. Tape Drive

A tape drive is the primary active component in the simulation. It is 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. The number and maximal performance of drives is configurable.

The general workflow of tape drives is shown in Figure 5.2. Initially, a drive requests getting the next most optimal drive for mounting. This might be the same as is currently mounted or none if no candidates exist. If this next tape is a different one than the previous one or the maximal mount time for the current one has been exceeded, it needs to be unmounted. Then, if a new cartridge has been proposed, it should be mounted. The mounted tape may not exist because it did not exist before and no new one has been mounted, otherwise the requests are sorted, the read duration is calculated and the files read accordingly.

The calculation of the performance of reading a file is most critical for the overall performance of the simulation. The offset on tape of the data that was provided by KIT is the file position as a counter, so that the physical distance between files is not known. Therefore it was implemented according to the reduced complexity formula from Equation 4.9, which assumes collocation of requested files by making the performance solely dependent on the particular file size and overall recall volume. The maximal read performance of drives is the specified maximal performance of a drive on uncompressed data multiplied with the assumed compression level.

The performance of a drive is realized in the simulation time by yielding a time duration event of the calculated read time for the currently read file that takes into account the size of the overall batch volume to be read. Logging performance data and displaying it as a uniformly bucketed time series is achieved by logging the beginning of the current performance along with a timestamp. This is later averaged over buckets, taking into account the varying durations to bridge the gap between the discrete time steps to a continuous understanding of the behavior.

5.3.4. Requestor

In order to simulate different request patterns, active components, called requestors, generate read requests according to different criteria and submit them to the tape system request queue. When there is not enough room for accepting the requests, an incremental back-off routine is performed, by increasing the wait times and stagnating at a reasonable retry rate, falling back to faster submission attempts upon success. Additionally, if mode active waiting is performed, the benefit of next-event time progression is lost and the simulation duration increases significantly, because many more steps need to be calculated.

A requestor usually creates and submits requests in lists of fixed or random length of usually around 50 up to 100 entries. After submission, a small wait period is inserted, which is also subject to a level of randomization. Different modes are currently implemented, one of which randomly requests reading files from the file index, which might then also include duplicates, another requests the files by dataset in ascending order by creation age, the way that they are requested by the ATLAS collaboration. A third mode schedules the files by tapes, selecting request for cartridges according to their first occurrence in the sequence of requests by dataset. It serves to evaluate the optimally possible performance when all requests for a tape are known at a time. The number of parallel requestors by tape may be configured. Finally, an online sorter exists, that reorders and schedules the n known requests that would have arrived by dataset via the higher level systems. It orders requests according to tape location based on the limited window of known requests and tries to optimize the overall performance by maximally making use of all drives.

5.4. Applicability and Limitations of the Model

The introduced simulation model is abstracted to the most relevant parameters and characteristics of tape system behavior. It is explicitly limited to being able to simulate the behavior of a tape system when managing and processing read requests, so that the tape silo is initialized at startup and at runtime only accessed for reading purposes. The parallel operations of reading and writing as well as their prioritization and possible sharing of queue space will usually affect the behavior of real-world systems, but also make it more difficult to analyze the effect of certain changes in request scheduling, which would need to be assessed against a stochastic background of other operations. The model also does not take into account network characteristics and associated throughput limitations regarding request arrival, nor buffering by tape drive memory and the detailed staging behavior of recalled files to a disk system. In this way, the scheduling, mounting, seeking and reading behavior is isolated. While drives may be reserved or opportunistically used in real tape systems, the robot arms for mounting are shared, which is neglected in this design. Finally, details in areas such as track layout, block size and file markers as well as the complex characteristics of specific hardware are only roughly approximated or omitted entirely, both to avoid overspecification and loss of generalizability, as well as the complexity and lack of insight due to proprietary specifications.

Therefore, the implemented simulation should not be mistaken for being able to accurately emulate the behavior of real-world systems in detail, but taken to be an approximation of the behavior characteristics of a generalized tape system that is primarily valuable for being able to simulate relative performance patterns.

5.5. Configuration and Different Setups

In order to assess different scheduling algorithms with respect to their benefit and shortcomings regarding the overall recall performance, the general performance of a tape system should be sufficiently realized in a simulator. To further approximate the behavior of a tape system in the data carousel experiment, the data provided by KIT was used to configure the simulator. First of all, during this exercise a maximum of 12 drives of type T10000D were used, which each allow for a maximum performance of 252 MB on uncompressed data (see Table 2.1). The data sheets specify average rewind times of 97 s, tape load times of 13 s and unload durations of 23 s, which were incorporated. Based on the data distribution information, the almost 500,000 files are located on 332 different cartridges which can hold 8.5 TB of uncompressed data. The given data placement allowed deriving a minimum and thus globally assumed compression level of 1.3:1, thus increasing the capacity to 11.05 TB. The data distribution is analyzed in more detail in Section 4.3.

In previous exercises, the site had been using a queue size of 2,000, which they have been able to increase to 30,000 at the beginning of this year by changing to a different dCache-TSM connector. Both queue sizes are used in the analyses as representatives of a small and a comparatively large queue size, that are able to contain 0.4 % and 6 % of the overall file count, respectively.

6. Evaluation of Simulated Tape Request Scheduling Patterns

In the following, the results of sending requests to the simulated tape system based on different scheduling patterns are evaluated. The patterns were tested and evaluated in two contexts of different tape system queue lengths. Finally, the progression and preliminary results of the current data carousel exercise are discussed in light of the simulation results and derived suggestions.

As described in Segment 5.5, the simulator is configured according to the KIT setup, the data distribution recreated. Most importantly, it includes 12 drives of type T10000D, which, incorporating the compression level of 1.3, offers maximal performances of 327 MB/s. Two queue sizes are used: 2000, which was used by KIT in the previous exercises and may contain up to 0.4 % of all file requests, and 30000, which can hold 6 % and has been used since the beginning of this year. In these two setups, requests were scheduled by different requestor modes according to the evaluation criteria specified in Segment 4.2.4. Random scheduling is used as a benchmark for comparison, by-dataset and different variations of by-tape clustering are simulated to evaluate the impact of different scheduling features on the metrics of interest. The simulations are run over a time period of 60 hours.

6.1. Small Tape Request Queue Length

The small request queue of 2000 slots is of particular difficulty regarding recall optimization goals. It is self-evident that a larger queue, up to a certain point, allows for larger volumes of requests to target the individual drives, increasing their performance. In Section 4.3, the KIT data distribution is analyzed in detail. The average dataset that will be requested to be staged from the KIT tape archive contains seven times as many requests as this queue may contain at once. The number of files that are located on a tape are between 500 and 2500 for most cartridges, so that submitting the sorted requests by tape will likely lead to having a single drive work with the maximal performance that recalling 20 - 60 % of the total cartridge volume can offer, sometimes two, rarely more out of the twelve available drives. It is expected that selecting requests uniformly over cartridges offers better results.

Random Request Pattern Randomly submitting requests is done to establish the baseline metric values for comparison. Figure 6.1 shows that the requestor saturates the queue from hour three, fluctuating slightly. This may be explained by the random request list size, incremental backoff behavior and fluctuating drive speeds. The queue size saturation is similar for all request patterns in this queue size and is therefore not shown for subsequent scheduling patterns.

During the operation of the simulation, the following behavior was observed:

- Number of files recalled: 78,183 (16 %)
- Total volume recalled: 217.328 TB (20 %)

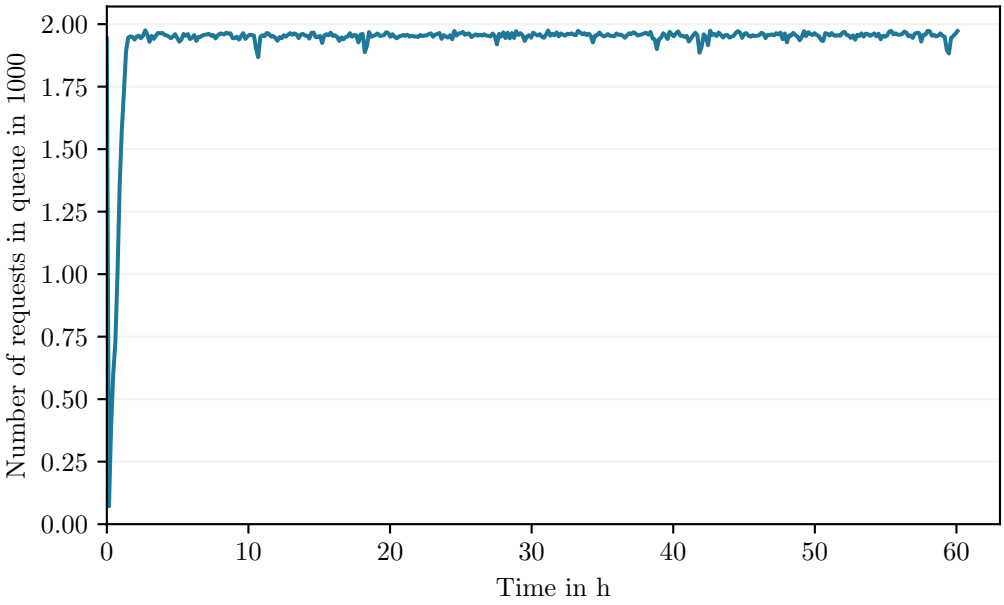


Figure 6.1.: Simulation with random request pattern and a small-sized tape system request queue of 2000: Requests in tape system queue over time

- Number of cartridges mounted in period overall: 6585
- Number of distinct cartridges mounted in period: 329

Figure 6.2 depicts the throughputs per tape drive over time. They fluctuate around 80 MB/s, occasionally dipping down to almost 50 MB, rarely performing better than 100 MB/s. These peaks are especially visible in Figure 6.3, where the throughputs for all drives are displayed stacked over time. The fluctuations result in an overall relatively even performance of about 830 MB/s, with variations of at most 200 MB/s.

Figure 6.4a shows the distribution of recalled tape volume percentages, which are largely below half of a percent, none below two percent. The distribution of the time it took to stage a file from the time of request submission is visualized by the diagram in 6.4b, showing an overall similarly frequent occurrence of the durations between 20 and 160 min, averaging at about 88 min.

The small recall volumes and stage durations are reflected in plot 6.5a, showing a histogram regarding the durations of tape mounts, which are largely between 6 and 18 min, averaging at 11 min. The number of registered accumulated remounts over time is especially remarkable, shown in Figure 6.5b. It is calculated by the difference of overall mounts and unique mounts. The absolute amount rises lineally by about 600 remounts per hour, which equals 10 per minute. After 60 h, over 6000 have taken place, Overall, each cartridge has on average been mounted 19.7 times in 2.5 d using this approach.

Requests by Dataset When requesting files by dataset, starting with the newest ones, the intention is not to try to maximize performance. The results rely on how optimal the dataset distribution on tape is. This may again vary depending on queue size. During the operation of the simulation, the following behavior was observed:

- Number of files recalled: 87,048 (18 %)

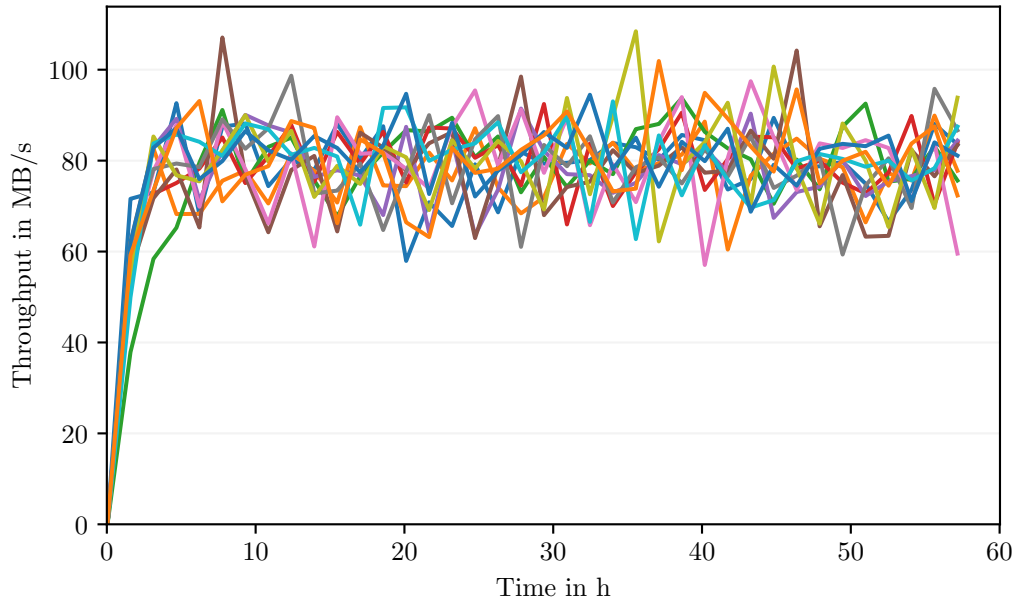


Figure 6.2.: Simulation with random request pattern and a small-sized tape system request queue of 2000: Throughput per drive

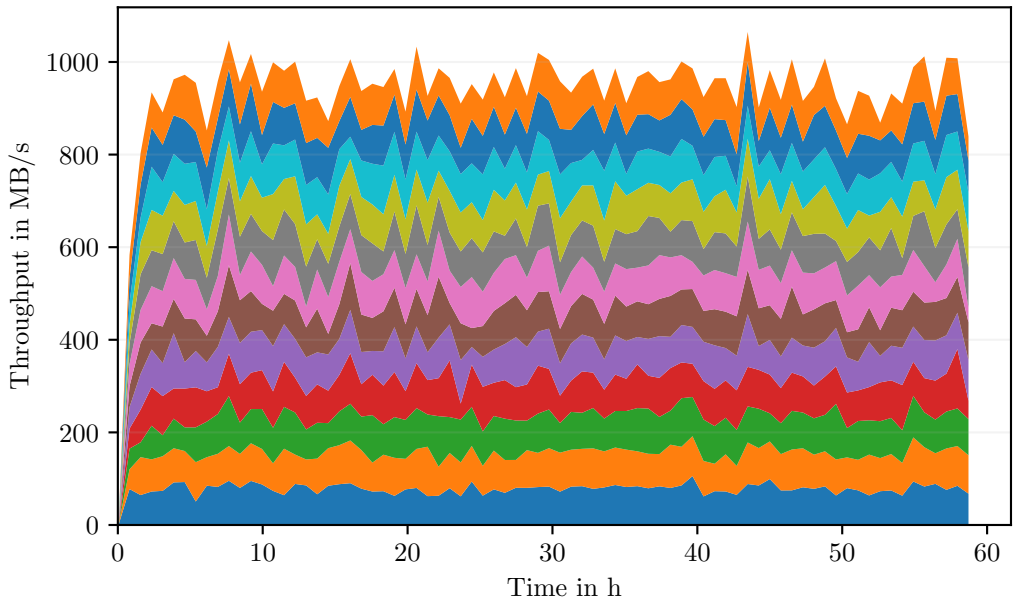
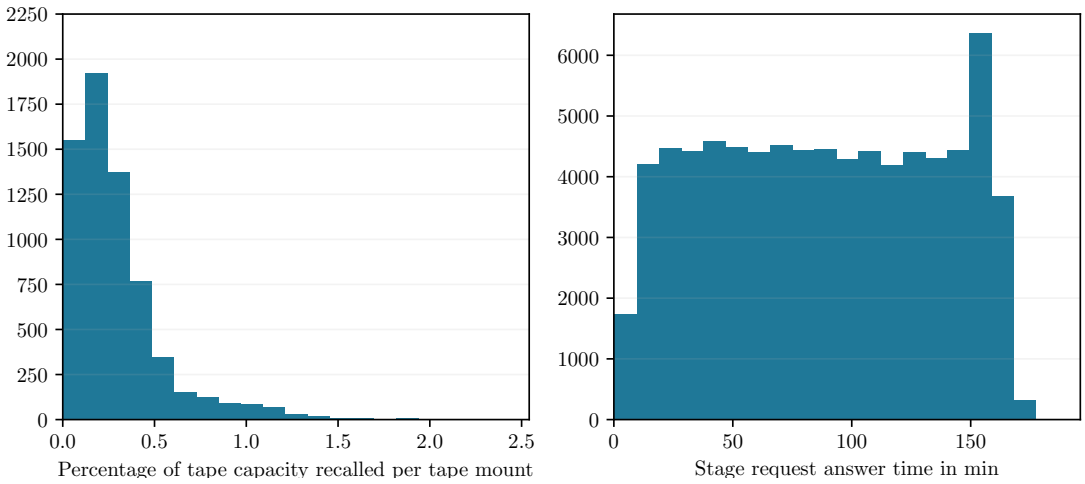
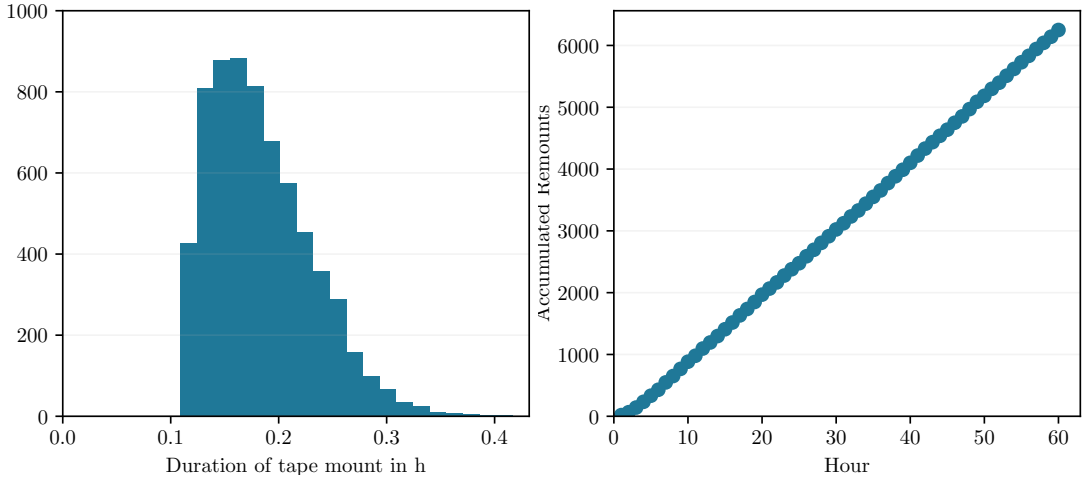


Figure 6.3.: Simulation with random request pattern and a small-sized tape system request queue of 2000: Stacked throughput per drive



(a) Percent of tape capacity recalled per mount (b) Time for staging

Figure 6.4.: Simulation with random request pattern and a small-sized tape system request queue of 2000: Percent of tape capacity recalled per mount and time for staging



(a) Duration of tape mounts (b) Accumulated number of remounts

Figure 6.5.: Simulation with random request pattern and a small-sized tape system request queue of 2000: Duration of mounts and accumulated remounts

- Total volume recalled: 240.829 TB (22 %)
- Number of cartridges mounted in period overall: 1221
- Number of distinct cartridges mounted in period: 77

Figure 6.6 shows the throughput per tape drive in MB/s. Especially during the initial 15 - 20 h and between the hours 44 and 55 the performances are spread very wide, with variations of over 150 MB/s. While some drives are almost idle, others perform at over 150 MB/s, in one instance even over 180 MB/s. These fluctuations are also visible in Figure 6.7, which shows the stacked performances of all drives. The overall performance is visibly lower in periods where the difference between individual drive performances is larger, falling down to below 800 MB/s and peaking at just above 1400 MB/s. The variation may be explained by uneven distributions of requested files over tapes. The initial datasets seems to be distributed over a number of tapes that is less than the number of available drives. Later on, other datasets partially intermingle and are individually spread over more cartridges, again dissolving to fewer cartridges in the later period. Between hours 20 and 40 as well as from 50 onward the performance is rather stable, however, varying around slightly over 1200 MB/s, which indicates that the requested files are more evenly distributed over tapes in this period. These observations correspond to the breakdown of the file counts per tape for each dataset as shown in Figure 4.9 in Section 4.3. The average throughput overall for this simulation is 855 MB/s

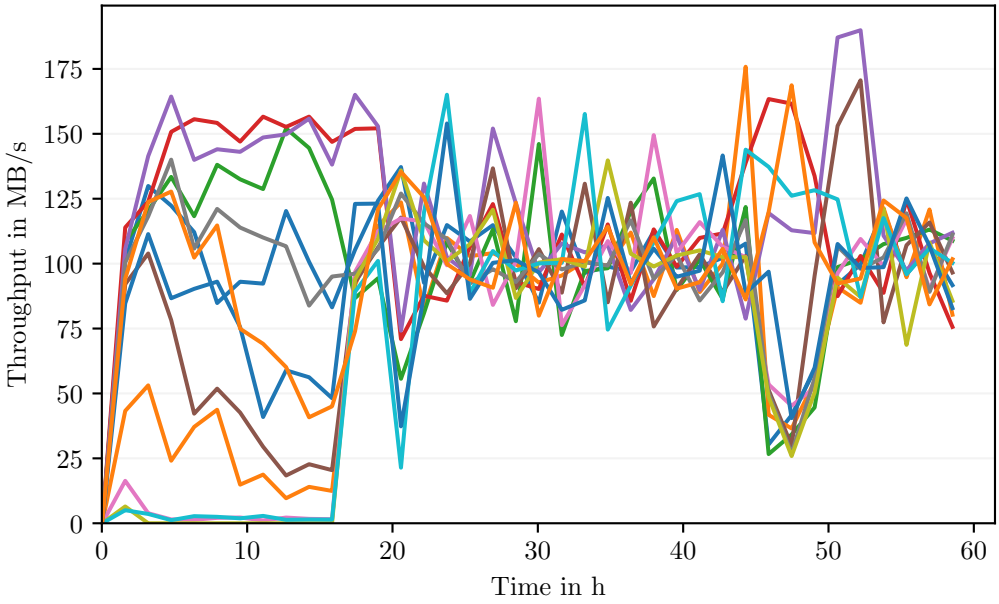


Figure 6.6.: Simulation with requests by dataset and a small-sized tape system request queue of 2000: Throughput per drive

Compared to the random request pattern, the recalled data volume percentage as shown in plot 6.8a is larger, the stage request answer time slightly smaller, in the majority of cases below 150 min and on average taking 78 min. This corresponds to the duration of tape mounts as shown in Figure 6.9a, which is usually 5 h or less, but sometimes also up to 10 h, averaging at 45 min. The number of remounts over time has in comparison improved even more significantly. Figure 6.9b shows a reduction to almost one sixth of the overall number of remounts by the end of the 60 hours. During the first 18 hours, no cartridge had to be remounted at all, followed by a linear increase of about 22 remounts per hour. Interestingly, the slope increases dramatically between

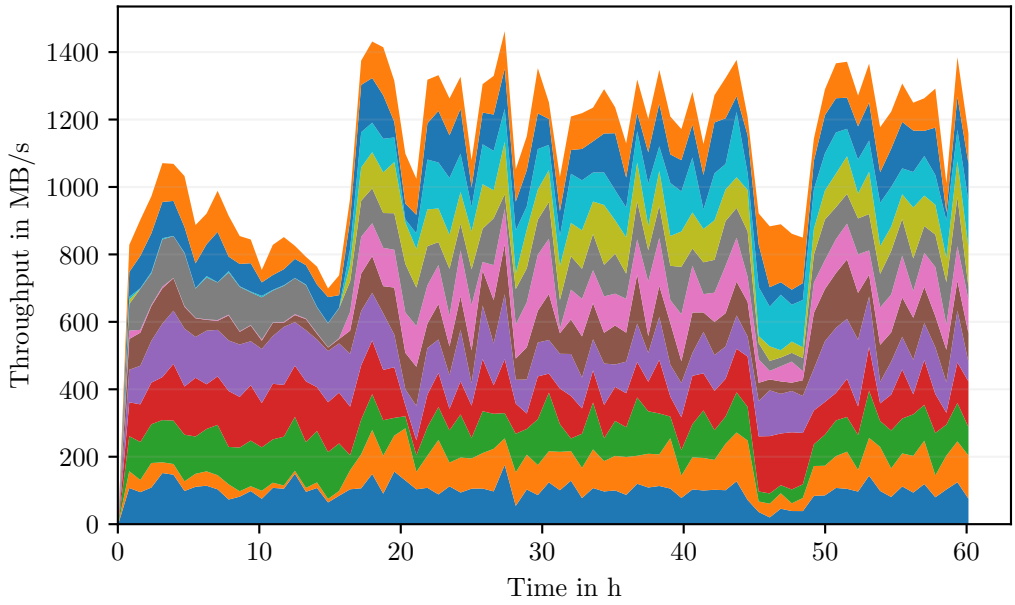


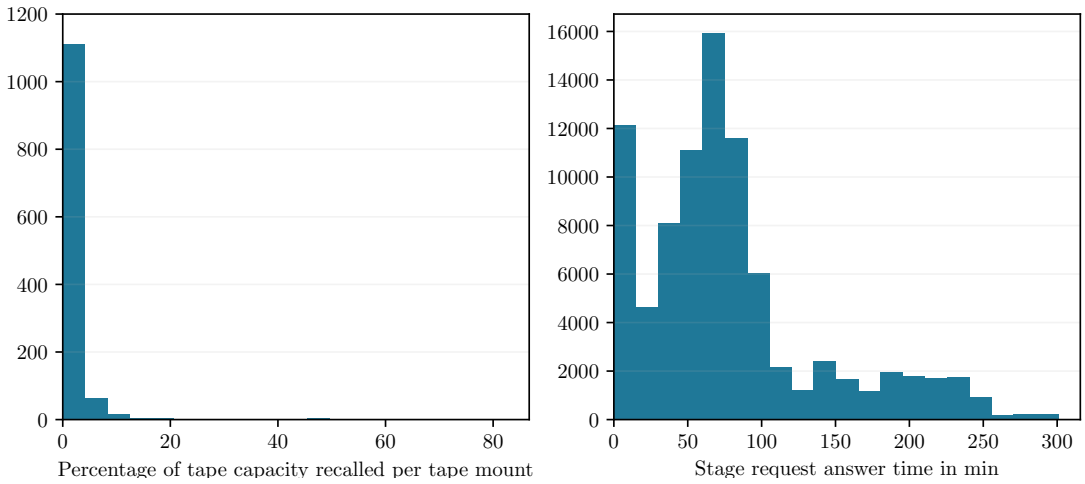
Figure 6.7.: Simulation with requests by dataset and a small-sized tape system request queue of 2000: Stacked throughput per drive

hour 45 and 50, corresponding to the dip in overall performance, which is mostly carried by five drives, which the rest are apparently busy swapping cartridges and reading small number of files from each. Afterwards, the slope steadies again.

Requests for 2 Tapes in Parallel Reading requests by tape aims to analyze the maximum achievable performance, when all files from each cartridge are known. The tapes are requested in the sequence that they first occur in the expected request pattern by dataset, but all requests are assumed to be present for the cartridge in question at the time of requesting. Ordering the requests under these optimal conditions is mainly dependent on the number of parallel cartridges that are requested. When this number is two, the queue is filled with all requests for the first two cartridges in line, then the second two and so on, until the queue is full. During the operation of the simulation, the following behavior was observed:

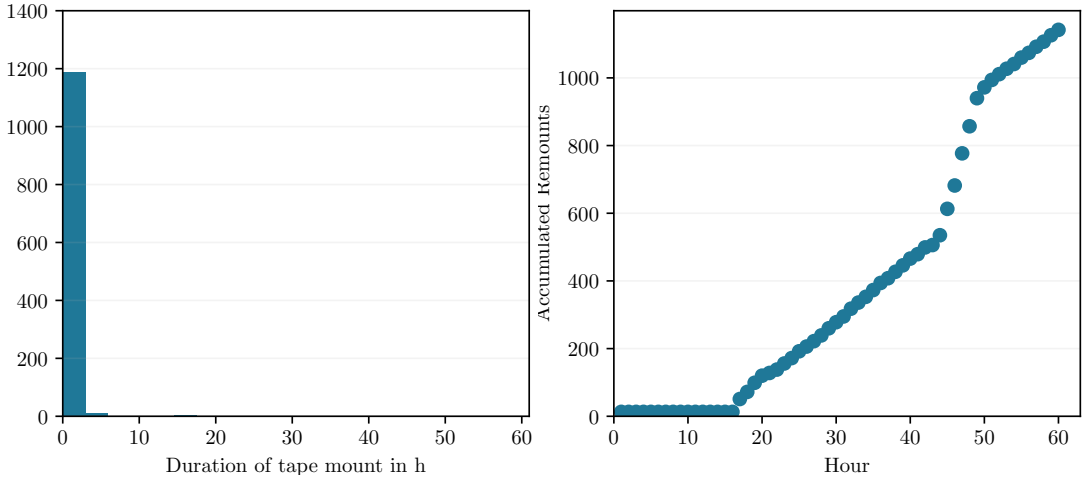
- Number of files recalled: 37,667 (8 %)
- Total volume recalled: 102.736 TB (9 %)
- Number of cartridges mounted in period overall: 27
- Number of distinct cartridges mounted in period: 27

During this period no tape was mounted twice. However, because of the small queue size, the result shown in Figure 6.10 is undesirable as expected. Initially, only two drives are busy recalling data at high individual performances of 150 to 200 MB/s, later on requests intermingle and up to five cartridges are mounted in parallel, which is less than half of what is possible. This leads to average performances of about 400 MB/s and a peak performance of 800 MB/s. It is obvious that this way of scheduling is highly sub-optimal.



(a) Percent of tape capacity recalled per mount (b) Time for staging

Figure 6.8.: Simulation with requests by dataset and a small-sized tape system request queue of 2000: Percent of tape capacity recalled per mount and time for staging



(a) Duration of tape mounts (b) Accumulated number of remounts

Figure 6.9.: Simulation with requests by dataset and a small-sized tape system request queue of 2000: Duration of mounts and accumulated remounts

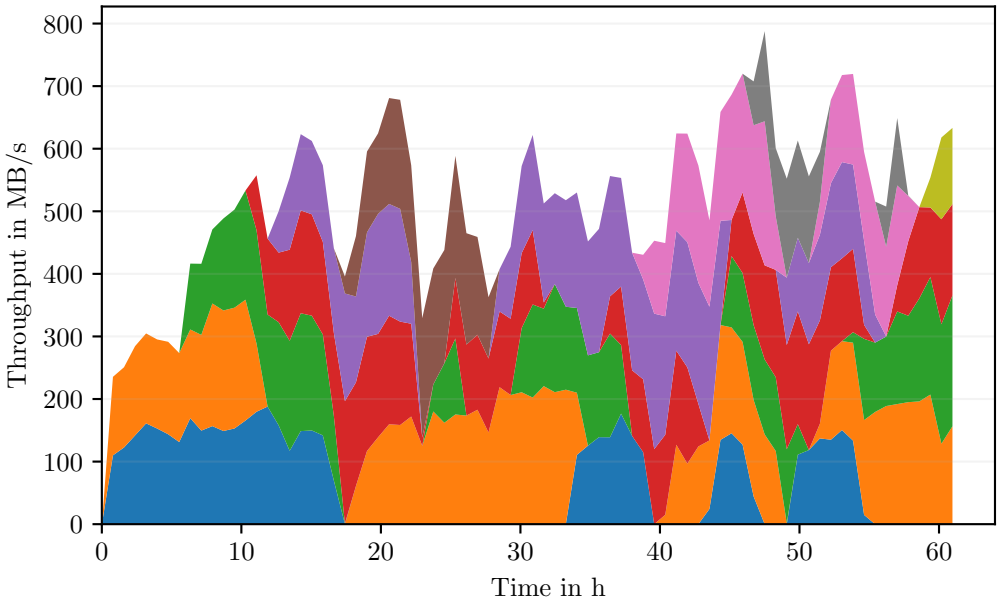


Figure 6.10.: Simulation with requests for 2 tapes in parallel and a small-sized tape system request queue of 2000: Stacked throughput per drive

Requests for 12 Tapes in Parallel When increasing the parallelization to the maximum level that seems useful, requesting as many cartridges in parallel as there are tape drives, this results in the best performance for this queue size of all intermediate parallel request strategies. During the operation of the simulation, the following behavior was observed:

- Number of files recalled: 90,331 (18 %)
- Total volume recalled: 244.921 TB (22 %)
- Number of cartridges mounted in period overall: 269
- Number of distinct cartridges mounted in period: 71

The number of files and volume recalled is similar to the results when requesting by dataset. However, as can be seen in Figure 6.11, the average variation, but also the peak performance at almost 200 MB/s is larger than in the former case. When looking at the accumulated and overall performance shown in Figure 6.12, it is still larger and more stable than requesting by dataset directly, more regularly above 1200 MB/s and occasionally above 1400 MB/s, averaging at about 1100 MB/s. It still fluctuates by up to 400 MB/s, which is due to the different volumes that need to be recalled from each tape overall. This fluctuation would likely decrease when more data were located on each cartridge, the data better collocated per dataset.

The percentages recalled shown in Figure 6.13a would eventually resemble the percentages that need to be recalled per tape overall, as shown in Figure 4.7a in Section 4.3. In the period of simulation, less than 20 % was recalled from most cartridges, but several mounts resulted in recalling between 20 and 40 %, a few even over 60 % of the total capacity of a cartridge, which were responsible for the peak performances by individual drives. The average capacity recalled was at 8.5 %, however. As shown in Figure 6.13b, the request answer times are in most cases below 150 min, slightly better than in the case of requesting by dataset. In direct comparison the duration per tape mount has increased so that while most mounts lasted less than 5 h, several took up to 15 or even 15 hours. This is also evident in Figure 6.14b, where the accumulated

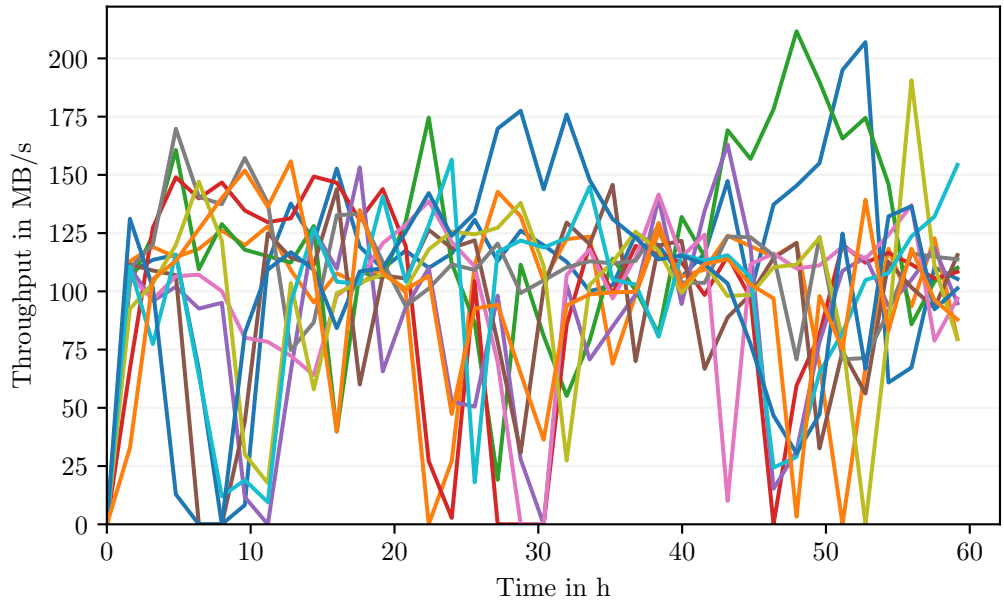


Figure 6.11.: Simulation with requests for 12 tapes in parallel and a small-sized tape system request queue of 2000: Throughput per drive

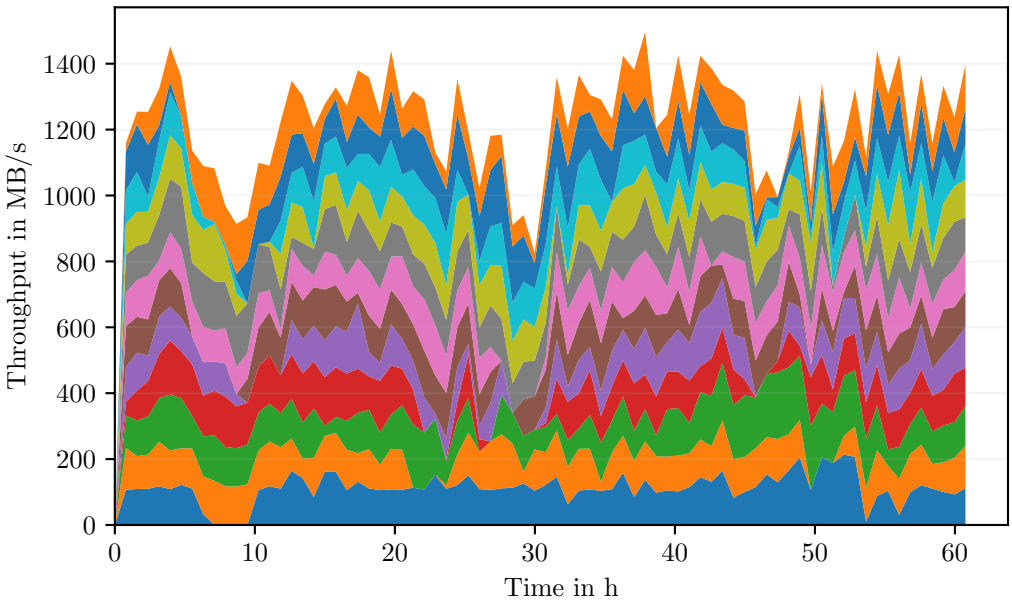
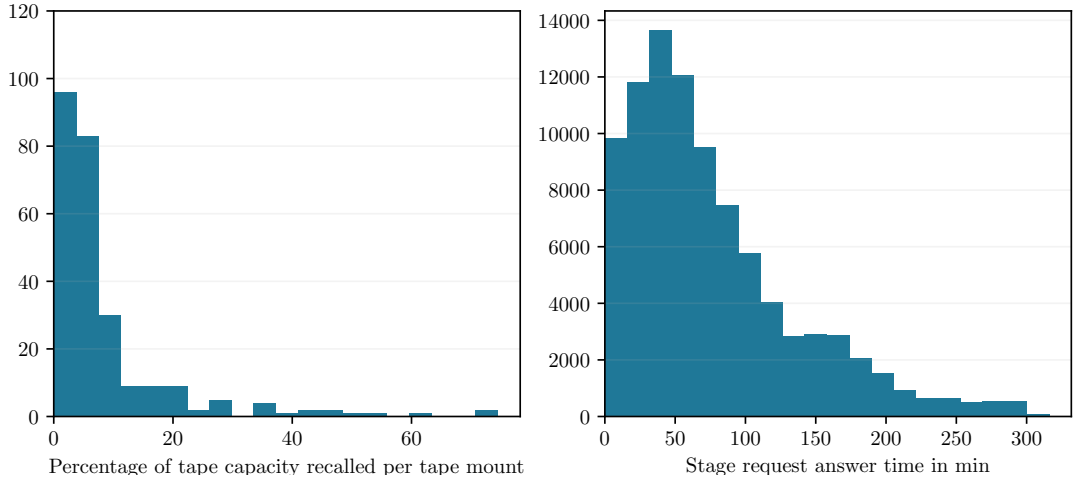


Figure 6.12.: Simulation with requests for 12 tapes in parallel and a small-sized tape system request queue of 2000: Stacked throughput per drive

numbers of remounts are relatively low, ending with only 175 in total. The number fluctuates more over time, which is also due to the smaller overall mount count.



(a) Percent of tape capacity recalled per mount (b) Time for staging

Figure 6.13.: Simulation with requests for 12 tapes in parallel and a small-sized tape system request queue of 2000: Percent of tape capacity recalled per mount and time for staging

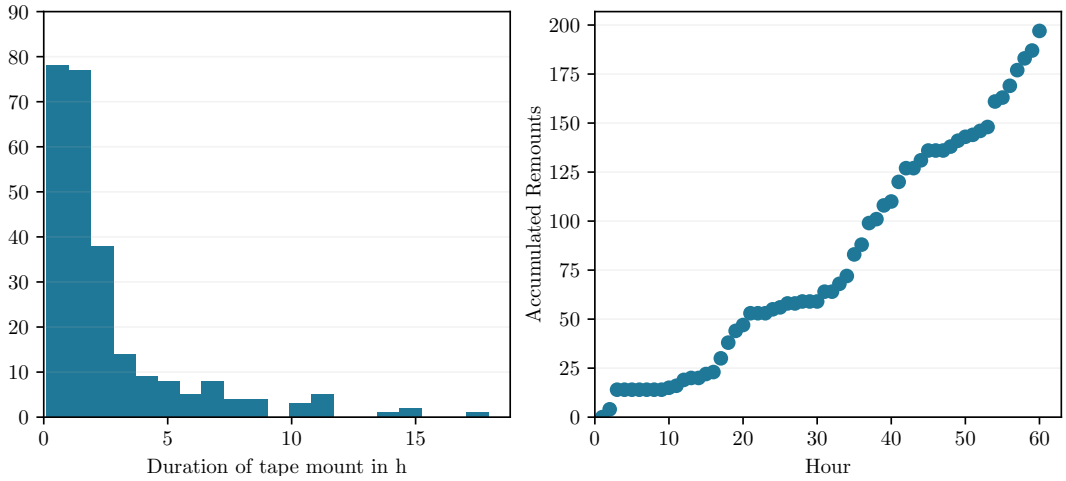
6.2. Large Tape Request Queue Length

The large requests queue of length 30000, chosen because of the new configuration of the KIT setup since the beginning of this year, is 15 times larger than the small reference queue of 2000. It is expected that the performance of recalling data from tape will improve, because the tape system is able to optimize recalls by reordering according to tape and on-tape location. If it is assumed that a cartridge on average contains 2500 files of interest, then requests for twelve cartridges may just fit into the new queue size. When files are better collocated on fewer tapes, this will no longer be the case.

Random Request Pattern When requesting data in a random pattern to a tape system with a queue size of 30000, the queue saturation was ensured, as shown if Figure 6.15, where saturation is reached after 14 hours. This is similarly the case for all request patterns and therefore not shown again.

During the operation of the simulation, the following behavior was observed:

- Number of files recalled: 96,892 (20 %)
- Total volume recalled: 270.850 TB (25 %)
- Number of cartridges mounted in period overall: 1004
- Number of distinct cartridges mounted in period: 330



(a) Duration of tape mounts

(b) Accumulated number of remounts

Figure 6.14.: Simulation with requests for 12 tapes in parallel and a small-sized tape system request queue of 2000: Duration of mounts and accumulated remounts

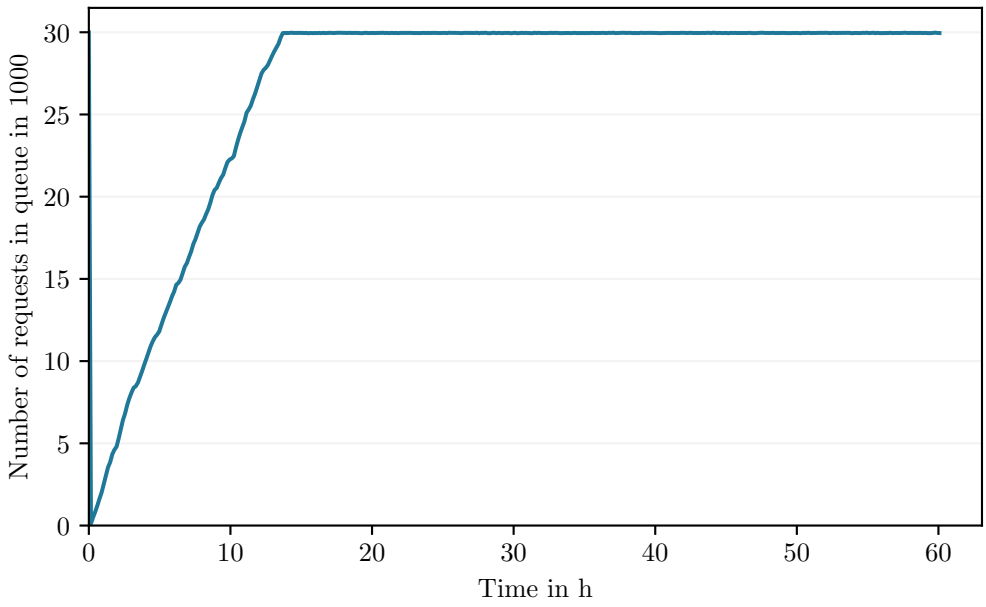


Figure 6.15.: Simulation with random request pattern and a large-sized tape system request queue of 30000: Requests in tape system queue over time

As seen in the random request pattern testing with a queue length of 2000, the individual drive performances cluster around a common medium performance with irregular spikes, as can be seen in Figure 6.16. The spikes into lower as well as higher performances are larger than in the former case, just as the average performance, which has increased from 80 MB/s to about 90 MB/s. Both is reflected in the stacked performance shown in Figure 6.17, where the overall performance can be seen to vary up to 400 MB/s and the average performance is just below 1100 MB/s.

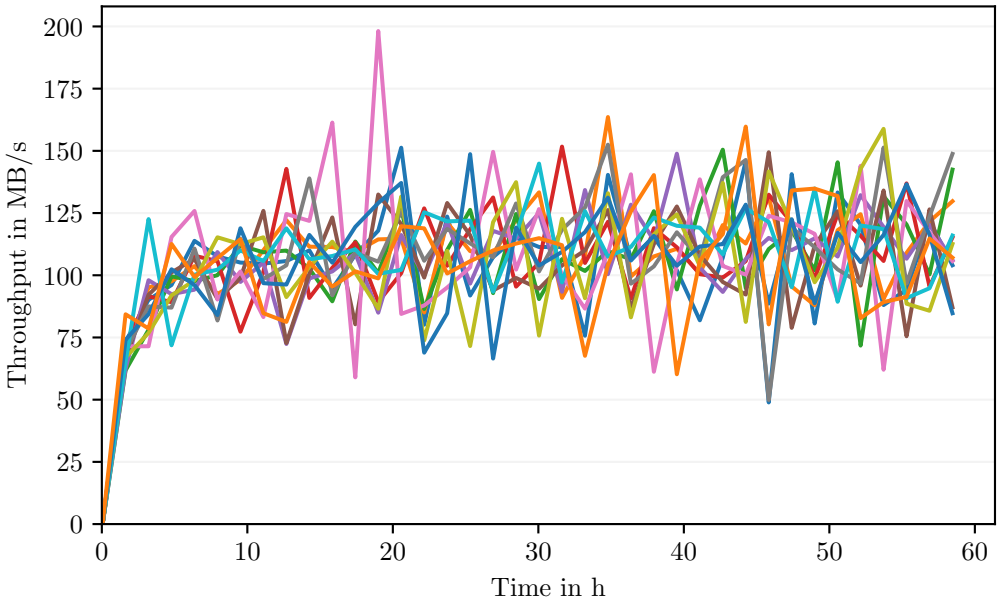


Figure 6.16.: Simulation with random request pattern and a large-sized tape system request queue of 30000: Throughput per drive

Figure 6.18a shows the percentage of tape capacity recalled, which is largely below 6 %, averaging at 2.4 %, which is 8 times as much as previously. The stage request answer time per request has increased, however, which may be expected with larger queue sizes. All durations are below 2000 min, which translates to about 33 h, averaging at 783 min or 13 h and is therefore almost 9 times as long as in the other random simulation.

As shown in Figure 6.19a, the maximum durations of tape mounts have also increased almost fourfold to the majority being below 2 h, but still peaking most prominently at less than half of a percent and averaging at 40 min. Interestingly, the number of cartridge remounts has decreased drastically by about 90 % of the previous maximal value, now just above 670 at the end of the simulation. The increase is again rather stable, becoming linear after 10 h, increasing with an approximate rate of slightly more than 12 remounts per hour, a 50th of the previous slope. Clearly, increasing the queue length has significant benefits for several of the metrics of interest even without conscious scheduling efforts, increasing the average accumulated performance by 27 % and decreasing the number of remounts by 90 %. This is at the cost of the average stage request answer time, which has increased by almost 790 %, because more requests can be submitted to the queue while the speed of processing them has not risen by the same percentage, because the number of drive is the same and the increase in performance merely due to better recall percentages per tape mount.

Requests by Dataset When requesting by dataset, a similar pattern can be seen. During the operation of the simulation, the following behavior was observed:

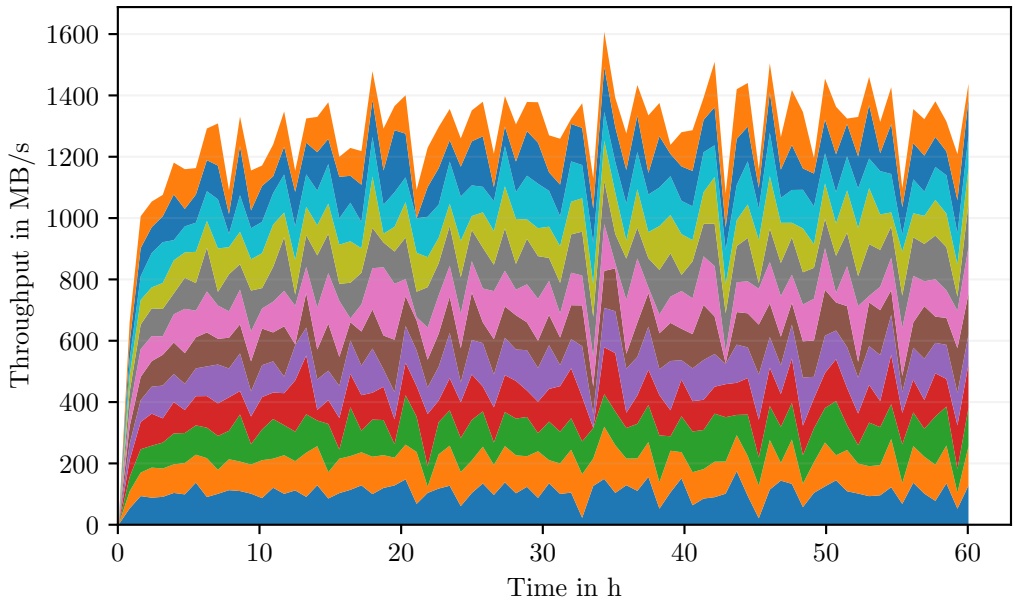
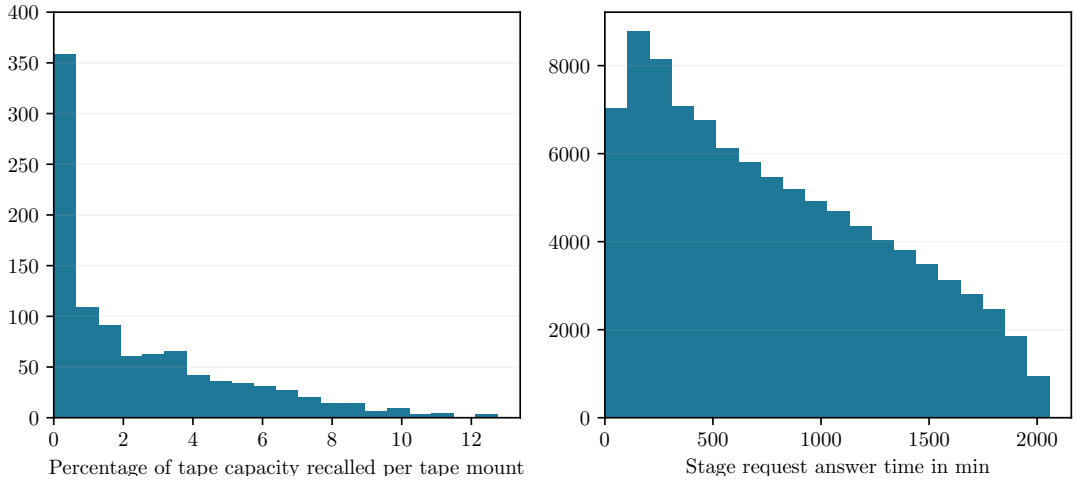


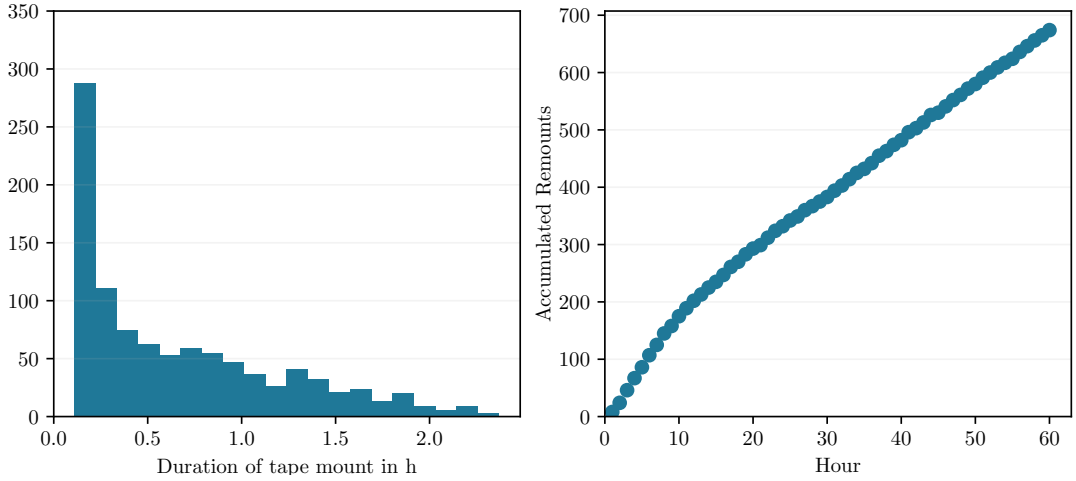
Figure 6.17.: Simulation with random request pattern and a large-sized tape system request queue of 30000: Stacked throughput per drive



(a) Percent of tape capacity recalled per mount

(b) Time for staging

Figure 6.18.: Simulation with random request pattern and a large-sized tape system request queue of 30000: Percent of tape capacity recalled per mount and time for staging



(a) Duration of tape mounts

(b) Accumulated number of remounts

Figure 6.19.: Simulation with random request pattern and a large-sized tape system request queue of 30000: Duration of mounts and accumulated remounts

- Number of files recalled: 163,792 (33 %)
- Total volume recalled: 442.474 TB (40 %)
- Number of cartridges mounted in period overall: 192
- Number of distinct cartridges mounted in period: 133

The number of files retrieved has increased by 80 %, the volume recalled has almost doubled compared to the same request pattern with a queue length of 2000, while the number of remounts has decreased from 6256 to 59 by 99 %. As can be observed in Figure 6.20, the variations in individual drive performance have increased to up to 250 MB/s at a certain point in time, which is largely due to better performances overall. Figure 6.21 shows the stacked performances of all drives and the resultant overall performance, which reflects the observations made for the individual drives. The overall performance is now around 1800 MB/s compared to 850 MB/s, which equals an increase of about 108 %, which is significantly larger than in the random cases. This is explainable by a much larger increase in the collocation of requested data due to the less random distribution of files within datasets on tape. The pattern is similar to the results for a queue of 2000, with an overall better performance and larger variations, which account for the irregular distributions of dataset chunks over cartridges.

The percentage of tape capacity recalled per mount as shown in Figure 6.22a has increased drastically, almost resembling the relevant volumes per tape as shown in Figure 4.7a, which is the most that is possible to retrieve per cartridge and thus optimal performance for the respective drive. It may be assumed that the larger the queue size becomes, the more the recall volumes approach the optimal value; assuming, that the requests arrive fast enough, which becomes less likely with increasing queue size. On average, 22 % of the total tape capacity have been recalled per mount. Figure 6.22b shows the stage request answer times in minutes, which significantly peaks at 600 min and averages at about 540 min, which is less than random requests with the current queue size but significantly more than any other request pattern so far with the small queue and almost five times as large as the corresponding pattern in the smaller queue.

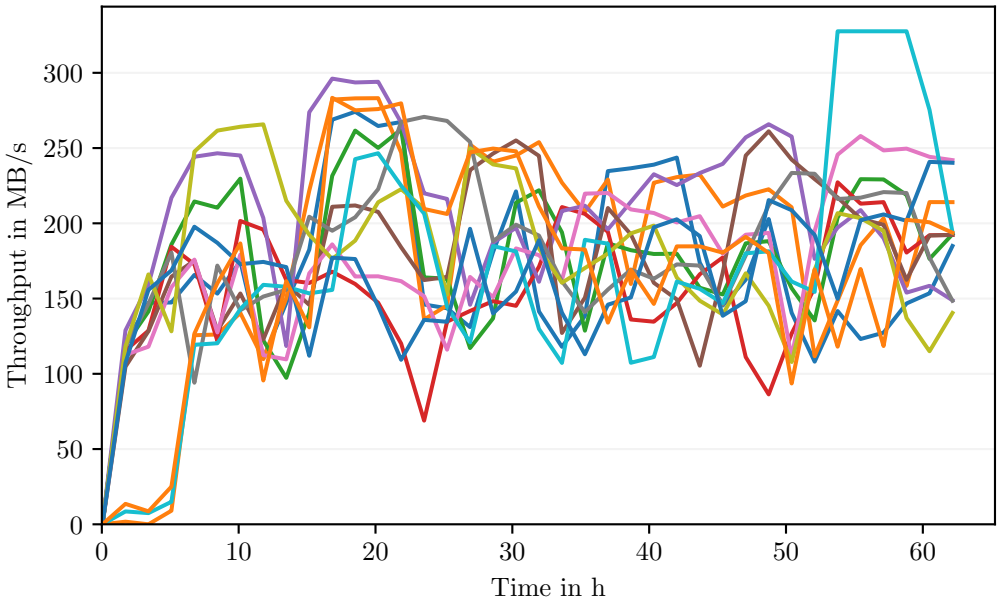


Figure 6.20.: Simulation with requests by dataset and a large-sized tape system request queue of 30000: Throughput per drive

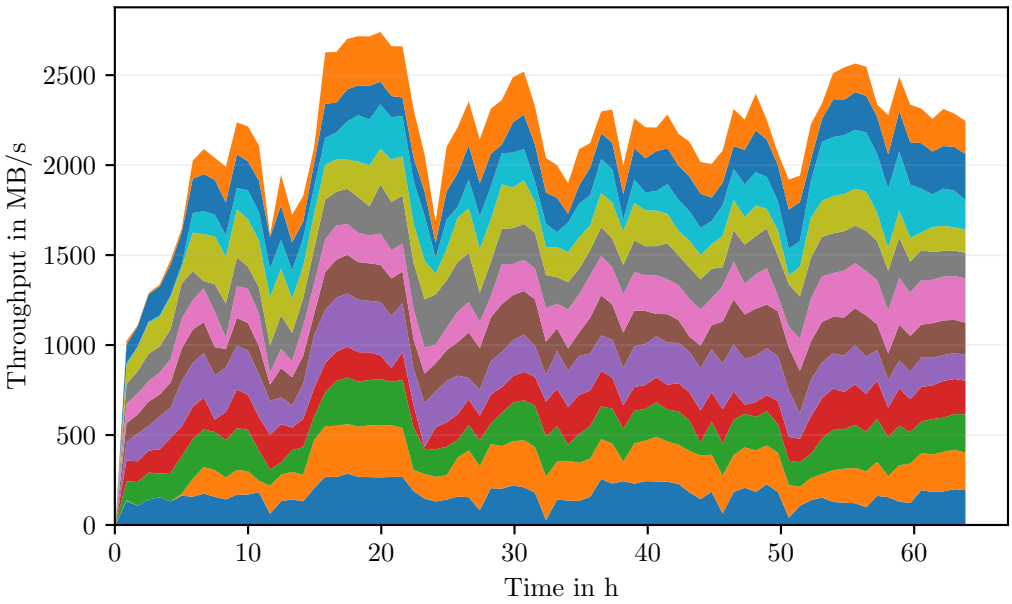
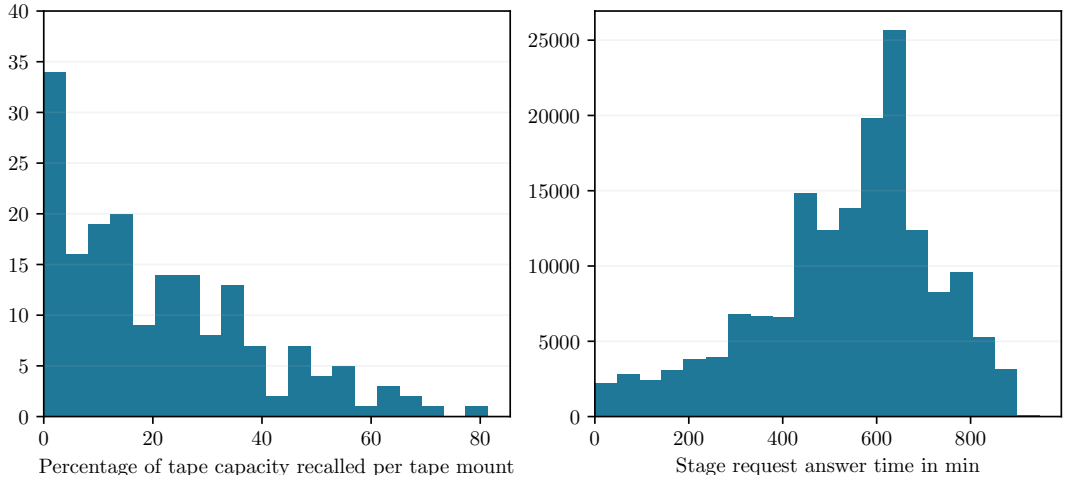


Figure 6.21.: Simulation with requests by dataset and a large-sized tape system request queue of 30000: Stacked throughput per drive



(a) Percent of tape capacity recalled per mount (b) Time for staging

Figure 6.22.: Simulation with requests by dataset and a large-sized tape system request queue of 30000: Percent of tape capacity recalled per mount and time for staging

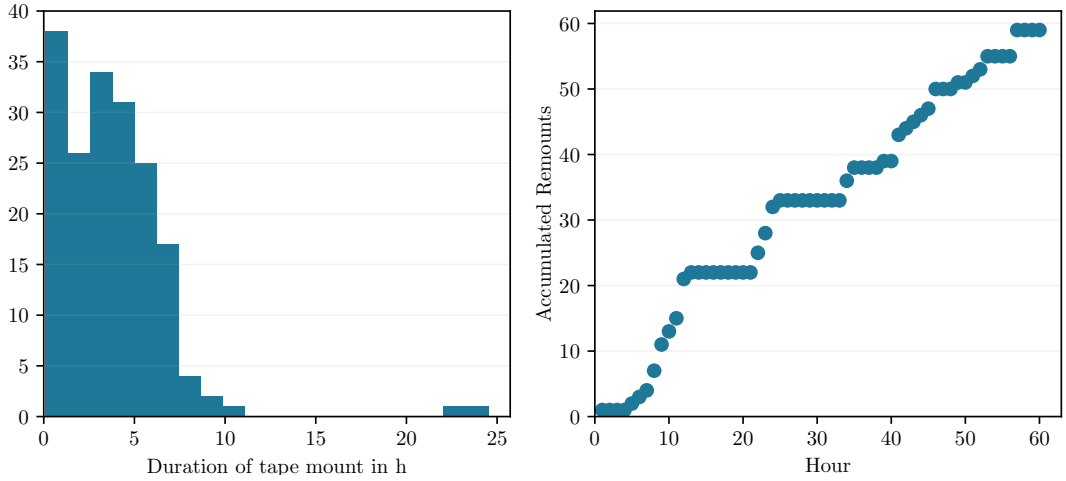
The distribution of the tape mount duration is shown in Figure 6.23a. Most mounts last less than 10 hours, averaging at 225 min or 3.75 h, which is five times as much as with a queue size of 2000. The accumulated remounts frequently stagnate at a certain level, not adding any in several hours. There are overall 60 remounts during the time of this simulation, which is significantly lower than in the smaller queue case, where tapes were remounted a total of 1144 times. This is compatible with the larger recall volumes and especially durations of tape mounts, which are directly related.

Overall, the request pattern significantly improved over its counterpart in the 2000 queue case with a 113 % larger overall throughput (836 MB/s vs. 1779 MB/s), a 95 % reduction in the number of remounts (1144 vs. 59), but also a 586 % increase of the average time for answering staging requests (78 min vs. 536 min). Compared to the random request pattern, the overall summed performance increased by 63 % (1090 MB/s vs. 1779 MB/s), the number of remounts decreased by 91 % (674 vs. 59) and the time for staging decreased by 35 % (783 min vs. 536 min).

Requests for 2 Tapes in Parallel When the request queue is large enough, any number of tapes requested in parallel will likely result in better performances than in the 2000-sized queue. However, when requesting large numbers in parallel, partial request list per cartridge are more likely than when requesting small amounts. Two in parallel was chosen because this queue size might be able to encompass all requests for twelve cartridges with the current sparse distribution over tapes, which is the number of drives. The slots for the last and second to last or after-last one should be filled up with as few as possible additional cartridges of the maximally possible performance. It is assumed that requesting more cartridges in parallel will result in lower overall performances.

During the operation of the simulation, the following behavior was observed:

- Number of files recalled: 188,463 (38 %)
- Total volume recalled: 511.806 TB (47 %)
- Number of cartridges mounted in period overall: 146



(a) Duration of tape mounts (b) Accumulated number of remounts

Figure 6.23.: Simulation with requests by dataset and a large-sized tape system request queue of 30000: Duration of mounts and accumulated remounts

- Number of distinct cartridges mounted in period: 144

Apparently, large volumes were recalled at low number of remounts. Figure 6.24 shows the performance of the individual tape drives. The variation between different performances is large, frequently up to 200 MB/s. No clustering is apparent at first sight. The peak performances of individual drives in several places reach the maximally possible value of 327 MB/s. In Figure 6.25 the stacked performances show that the overall value is always above 2000 MB/s, frequently even above 2500 MB/s, almost reaching 3000 MB/s in a few places. The average overall performance is approximately 2263 MB/s, which is the highest value out of all experiments fo far.

The distribution of recalled tape percentages per mount is shown in Figure 6.26a. It almost resembles the overall data distribution plot 4.7a, but lacking the peak capacities of up to almost 100 parcent and some of the bulk at 30 %. With an average recall of 30 % of a tape capacity, this is already close to the optimally possible 37 % for recalling everything. The distribution of the time it took to stage a file from the time of request submission is shown in Figure 6.26a. It almost resembles a Gaussian distribution with a trailing lower end, the most data points being in the region between 300 and 800 min, averaging at 509 min.

The duration of tape mounts, as shown in Figure 6.27a, reflects the large recalled volumes with durations of up to 8 h, averaging at 4.7 h. Most impressively, the accumulated numbers of remounts over the duration of the 60 h of simulation show that merely two remounts occurred overall; one after three and one after 9 hour.

Overall, the request pattern significantly improved over its counterpart in the 2000 queue case, most importantly with a 480 % larger overall throughput (393 MB/s vs. 2263 MB/s), an absolute increase of two remounts to 2 (0 vs. 2), but also a 186 % increase of the average time for answering staging requests (178 min vs. 509 min). Compared to the random request pattern with a queue length of 30000, the overall summed performance increased by 108 % (1090 MB/s vs. 2263 MB/s), the number of remounts decreased by 99.7 % (674 vs. 2) and the time for staging decreased by 32 % (783 min vs. 509 min). Compared to the request scheduling by dataset under the current queue length, the request scheduling by two tapes in parallel improved with regard to the overall

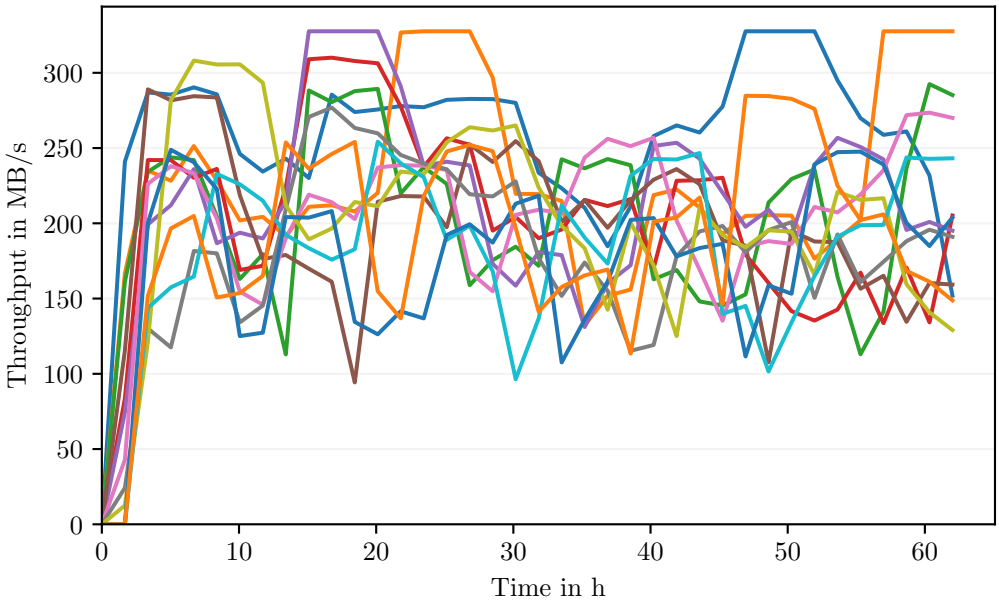


Figure 6.24.: Simulation with requests for 2 tapes in parallel and a large-sized tape system request queue of 30000: Throughput per drive

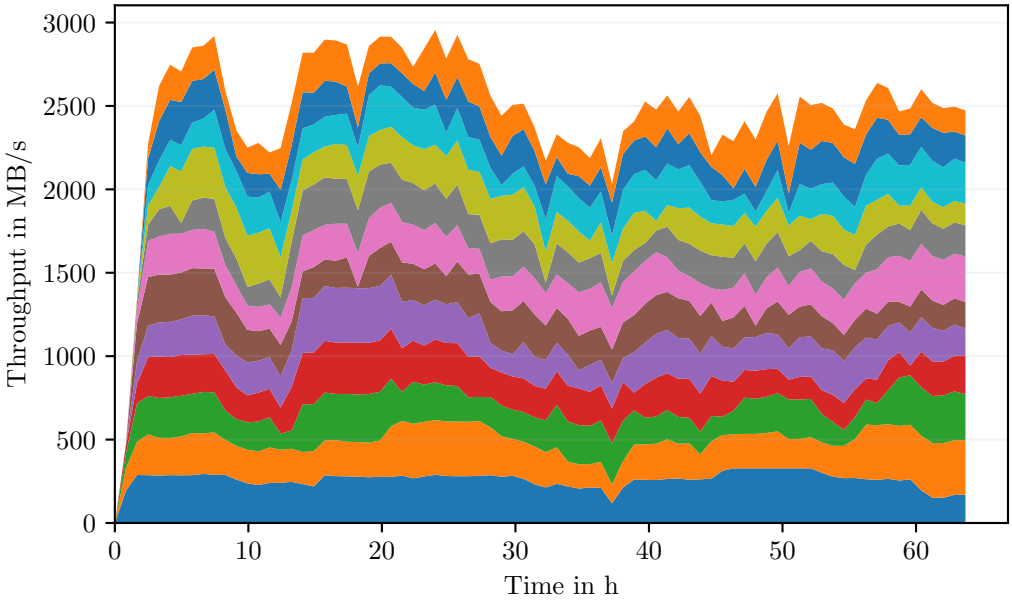
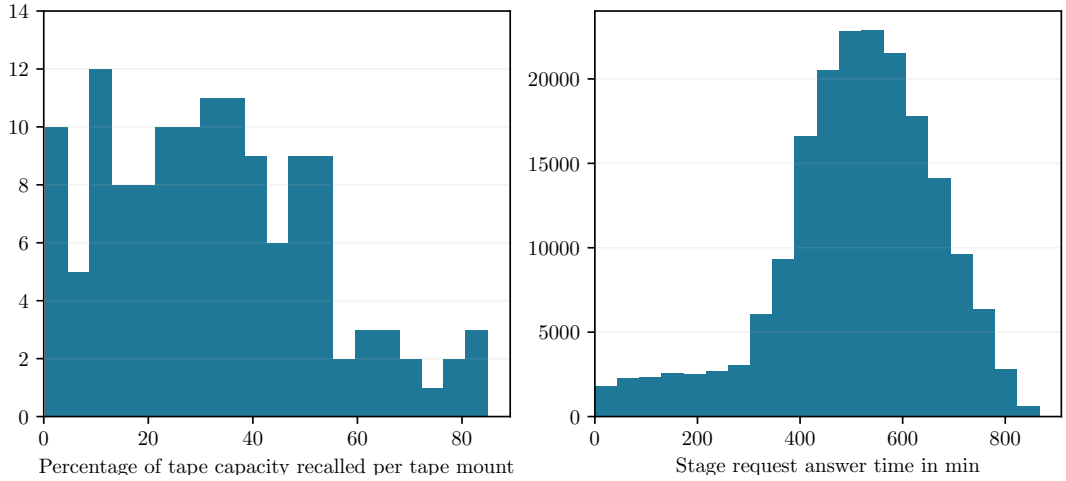


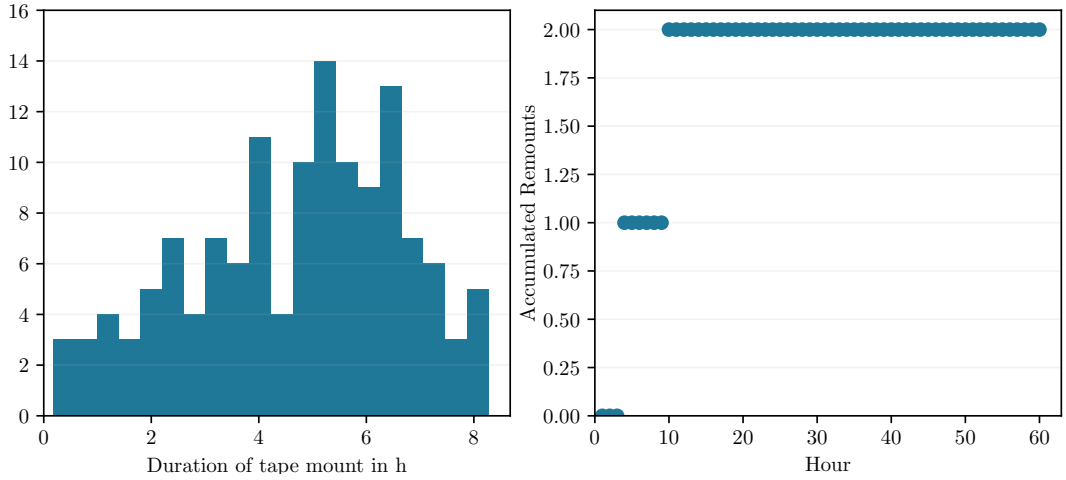
Figure 6.25.: Simulation with requests for 2 tapes in parallel and a large-sized tape system request queue of 30000: Stacked throughput per drive



(a) Percent of tape capacity recalled per mount (b) Time for staging

Figure 6.26.: Simulation with requests for 2 tapes in parallel and a large-sized tape system request queue of 30000: Percent of tape capacity recalled per mount and time for staging

throughput by 27 % (1779 MB/s vs. 2263 MB/s), the number of remounts 97 % (59 vs. 2) and the average time for staging by 5 % (536 min vs. 509 min).



(a) Duration of tape mounts (b) Accumulated number of remounts

Figure 6.27.: Simulation with requests for 2 tapes in parallel and a large-sized tape system request queue of 30000: Duration of mounts and accumulated remounts

Requests for 12 Tapes in Parallel Requesting more tapes in parallel than two is expected to be less optimal with regard to the current the KIT data distribution, because more tapes will be requested in small parts when the queue can not contain a multiple of that number. Requesting twelve tapes in parallel was the optimal approach in the case of the small queue, because on average requests for the contents of at most one complete cartridge may be held at once, no an even spread is a better approach than leaving a few drives idle, the possible variations in request

count per drive would not improve the overall performance. This test is conducted in order to verify this hypothesis and test whether the overall performance might be improved nonetheless.

During the operation of the simulation, the following behavior was observed:

- Number of files recalled: 175,800 (36 %)
- Total volume recalled: 476.456 TB (43 %)
- Number of cartridges mounted in period overall: 168
- Number of distinct cartridges mounted in period: 135

Figure 6.28 shows the stacked performances of the individual drives and thus the overall performance achieved with this access pattern. The variations in the overall performance is occasionally larger than 500 MB/s, peaking at approximately 2700 MB/s and averaging at 1960 MB/s. This is a decrease of 14 % compared to the case when requesting cartridges in parallel. Additionally, as can be ssen in Figure 6.29a, the number of remounts is larger than 30 after the 60 h of simulation, which is over a magnitude larger than the other case. Even the stage request answer time distribution, which is shown in Figure 6.29b and peaks around 600 min is slightly longer on average, although neglectibly so. This case is therefore not further considered.

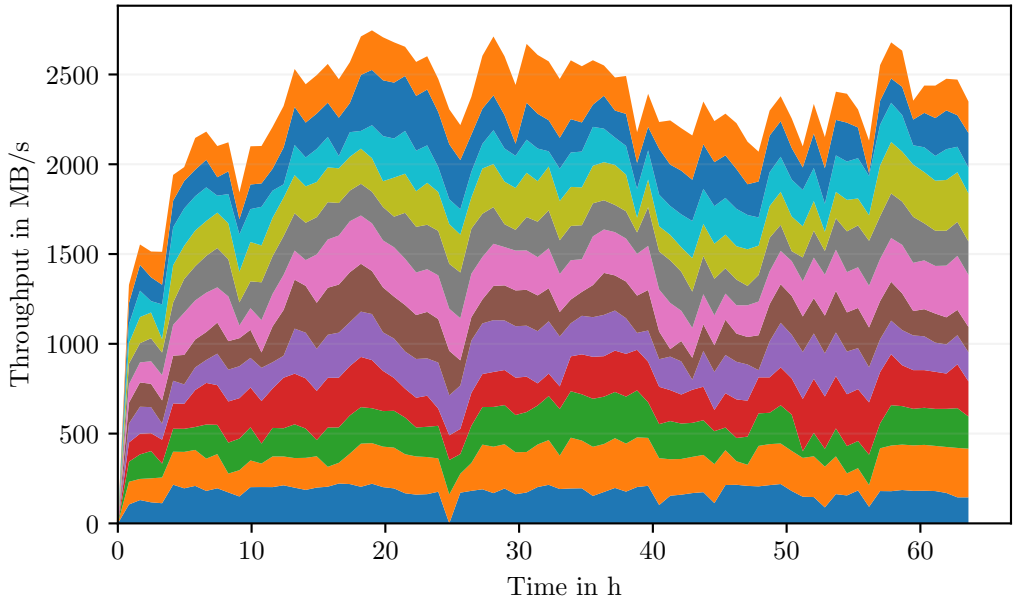
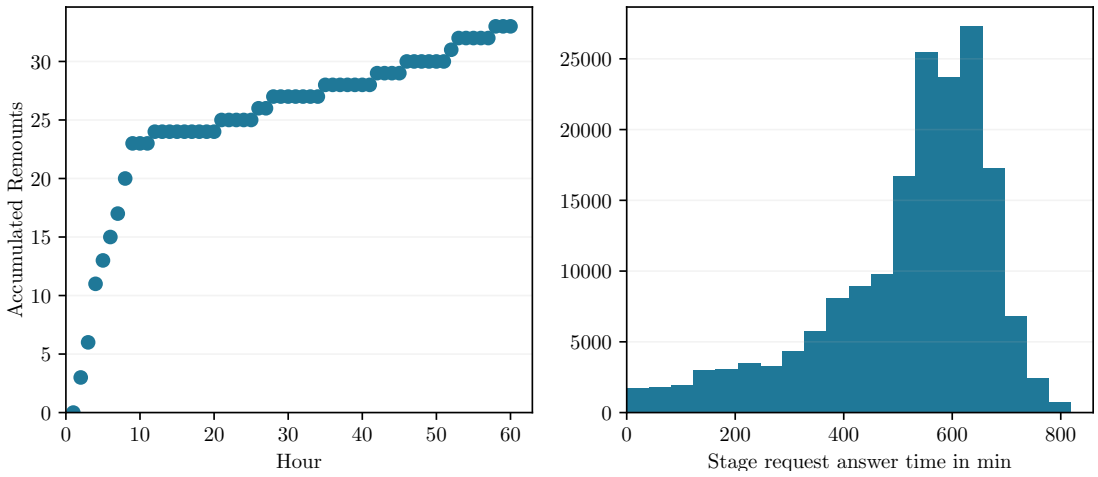


Figure 6.28.: Simulation with requests for 12 tapes in parallel and a large-sized tape system request queue of 30000: Stacked throughput per drive

6.3. Summary of Results

The most relevant metrics regarding measuring the value of different request scheduling patterns are collected in Table 6.1 with the respective values for each simulation mode and queue size. These are primarily the averaged overall throughput, the average time for staging a file with respect to the time of request submission to the queue as well as the number of mounts and distinct mounts, the difference of which is the number of remounts. Additionally, the average



(a) Accumulated number of remounts

(b) Time for staging

Figure 6.29.: Simulation with requests for 12 tapes in parallel and a large-sized tape system request queue of 30000: Accumulated number of remounts and and time for staging

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 6.1.: Summary of simulation results

percentage of capacities recalled per mount and average mount duration are included for better comprehension and context.

Overall, the simulation results have clearly shown that a larger queue size is beneficial, independent of the request pattern used. With random request selection, the overall performance was improved by 27 %, the number of remounts were reduced by 89 %. The idealized request pattern by dataset, which is the way that requests are sent and intended to arrive at a site's dCache and get scheduled for the tape system recall queue, even performs slightly worse in terms of the overall performance than the random request pattern for the queue size of 2000, although the number of remounts and average time for staging improves significantly. 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, which are 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, which is usually able to contain all requests for the twelve drives used in the current reprocessing campaign, scheduling requests for two cartridges in parallel was observed to be more optimal than sending one or as many in parallel as there are drives. This is because when all requests for twelve cartridges do not fit into the queue, because they are too large, the requests for each are only partially submitted, leading to trickling requests, whereas

when all requests for the first twelve cartridges do fit into the queue, the remaining space is equally divided between twelve or more new cartridges. Both cases lead to small recalls and more remounts, which is not desirable.

It has been shown, that the proposed request scheduling strategy for maximizing performance and minimizing the number of remounts from segment 4.2.2 is successful in the simulated reproduction of the KIT setup. In the case where the queue may on average contain all requests for as many cartridges as there are drives, represented in these experiments with the large 30000-sized queue setup, requesting two cartridges in parallel is the optimal strategy. Compared to the idealized basic request scheduling by dataset, the proposed way of scheduling was able to improve the overall performance by 27 % (1779 MB/s vs. 2263 MB/s), reduce the number of remounts by 97 % (59 vs 2) and reduce the average time for staging by 5 % (536 min vs. 509 min). When the queue may on average contain at most all requests for a single cartridge, selecting requests equally distributed according to the number of drives is optimal, with additional slight improvements with regard to the number of remounts and the average time for staging as seen in the case of the small queue size of 2000. Here, compared to the idealized basic request scheduling by dataset, the proposed way of scheduling was able to improve the overall performance by 32 % (836 MB/s vs. 1104 MB/s), reduce the number of remounts by 83 % (1144 vs 198) and reduce the average time for staging by 2 % (78 min vs. 76 min). Although the relative benefits decrease with the queue size because all scheduling approaches benefit, it is still highly significant for the analyzed queue sizes.

7. Conclusion

The following chapter summarizes the contents of this thesis, focusing on its results and limitations. Finally, an outlook is given, where future work is discussed.

7.1. Summary

This thesis analyzed approaches for improving tape restore request scheduling by the disk storage system dCache in front of automated tape storage in the context of the ATLAS LHC experiment. Different technologies, institutions and processes were introduced. The approaching challenge of the HL-LHC imposes important new requirements on the analysis workload and data management of the large experiments at the LHC. The resources required are several times larger than will be available with the current funding model and data management. Several working groups are trying to improve different aspects of the current workflows, such as data reduction, to lower the overall cost.

The data carousel concept was described as an approach being evaluated by the ATLAS experiment group to use the less expensive magnetic tape storage more actively, while it is currently mainly used for archival storage, rotating the data that is recalled and kept on disk. In order to successfully implement this concept, tests are being conducted to asses the current staging performances of the Tier 1 sites, which manage tape storage for ATLAS. A long chain of workflow and data management software needs to optimally cooperate. Most of these sites use the dCache software to manage their disk storage, which receives requests to by the high-level data management software Rucio and needs to pass these on to the tape system, which manages the writing to and recalling from magnetic tape storage. Because tape systems usually have smaller request queues than the number of buffered requests known within the storage system, and dCache insists on reserving space for the data which it requests from the tape system in order to be able to store it, only a subset of these may be passed on to the tape system queue. There they usually get reordered by tape and the file's position on the respective tape before they are read by one or several tape drives. The dCache system currently passes on read requests without attempting to optimize the resultant recall performance, of which it has no knowledge. A method is suggested for retrieving certain information from the tape system to be used for optimizing the scheduling approach, which requires the knowledge of certain tape system behaviors. Therefore, an approximate method for calculating the performance of a single tape drive or a multi-drive system is introduced. It is shown that the problem of selecting the optimal set of requests to be passed to the tape system queue is NP complete and pseudo-polynomial time algorithms that exist for related problems are only viable for small lists that need to be selected from. This is not the case for the problem at hand, where request queues within dCache may be tens of thousands to hundreds of thousands of requests large. Therefore, an approximated solution is sought, where the number of cartridges for which requests are selected is considered to ideally be equal to the number of tape drives and an optimal distribution of requests per tape is sought. An iterative approach to find a good solution is proposed.

Because these scheduling strategies can not realistically be tested in a real environment with the data volumes and time periods of interest, a simple tape system simulator based on discrete

event simulation is conceptualized and implemented. The KIT center, which takes part in the tape carousel exercises as the German Tier 1 site, has volunteered to cooperate with the dCache development team to understand and improve the behavior of the software in this context. By providing detailed information on the datasets that are recalled in the current data carousel reprocessing exercise and their locations on tape as well as the general storage setup at KIT, it could be reproduced in the simulated environment. The behavior of the simulation can therefore be approximated and assessed more realistically. While the model is highly abstracted, general conclusions regarding behavior tendencies may be drawn. Different scheduling approaches are subsequently analyzed in the virtual environment with different queue lengths as the limiting factor. They are evaluated according to certain criteria of interest, primarily the overall average recall performance, the number of remounts, which may drastically reduce the overall lifetime of magnetic tape media, as well as the average time for a request being completed, starting from their submission to the tape system. Several other parameters are evaluated for context and better overall understanding. In this way, the impact of different queue lengths and scheduling under optimal conditions is assessed, the proposed method for approximating optimal scheduling is empirically verified for two queue sizes by comparison to baseline and standard scheduling. Compared to the idealized standard request submission pattern, it was able to improve the overall performance by approximately 30 %, reduce the number of remounts by 90 % and lower the average time for staging by 3 %.

7.2. Outlook

Realizing the data carousel concept is highly complicated and depends on several different systems, setups and mechanisms, optimizing the request scheduling by a dCache instance in front of a tape system is only a small part of the overall effort. Nonetheless, it may lead to large improvements, as was shown in the relative performances and mounting behaviors of the simulated tape system. Because the problem can be shown to be NP complete, the proposed way of scheduling is only a first approximation which may be further improved. Implementing this solution will need to start with realizing a component to query a tape system for information on the locations of the presented files that are requested, in order to better reorder and select a subset to be passed on.

Although the tape system simulator was conceived for a specific setup and is highly abstracted, different configuration options and ways of extending it exist or may be added. It is thinkable to extend and use it for testing further optimization approaches and even data placement strategies. So far neglected components such as network, writing behavior and staging buffer may be added. *Smart writing* is being discussed in the data carousel working group alongside strategies for optimally recalling already existent data, which aims to better write files to tape so that recalling them is more efficient. The primary aim is to optimize the collocation of data per dataset. This would ideally also improve the optimized recall strategies for the general case. If data is better collocated, the proposed strategies of this thesis are able to make use of it and further improve the metrics of relevance.

A. Simulation Results

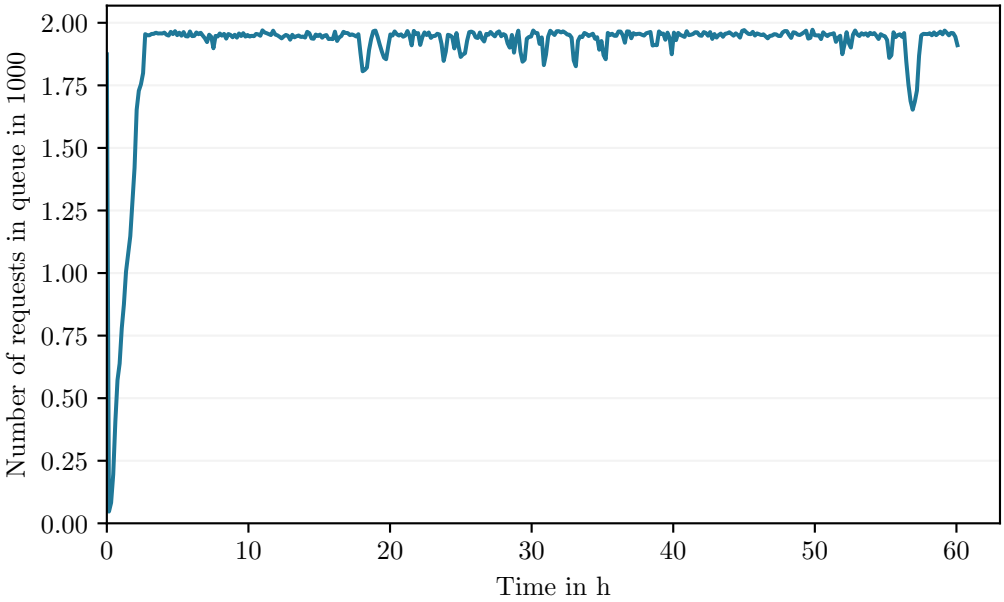


Figure A.1.: Simulation with requests by dataset and a small-sized tape system request queue of 2000: Requests in tape system queue over time

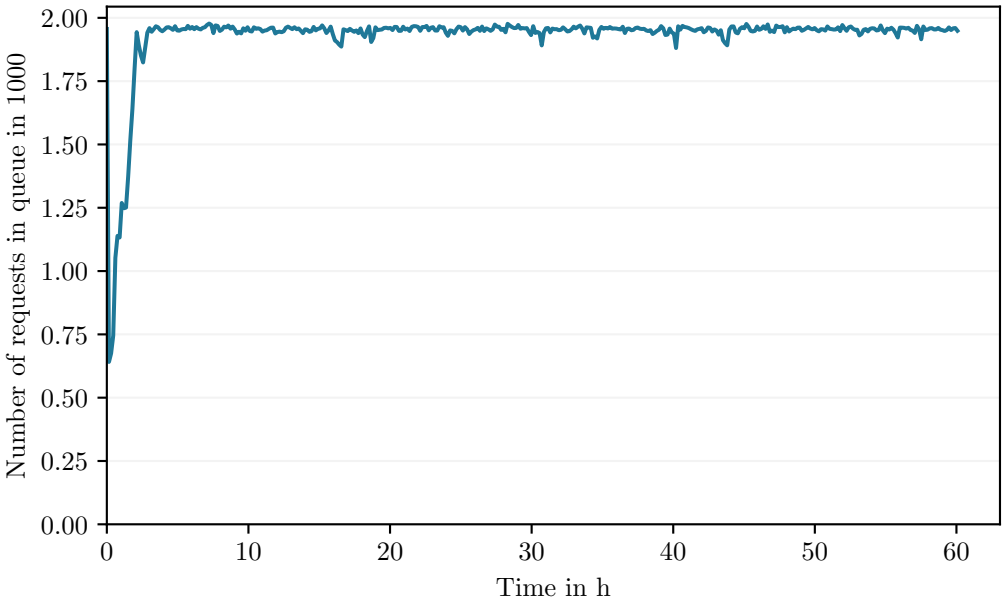


Figure A.2.: Simulation with requests for 12 tapes in parallel and a small-sized tape system request queue of 2000: Requests in tape system queue over time

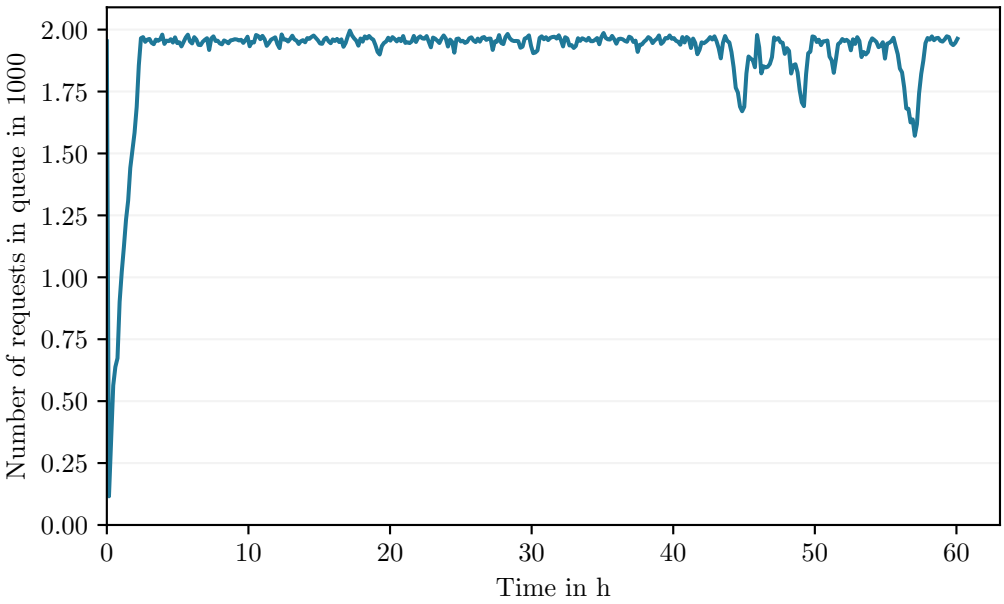


Figure A.3.: Simulation with requests for 2 tapes in parallel and a small-sized tape system request queue of 2000: Requests in tape system queue over time

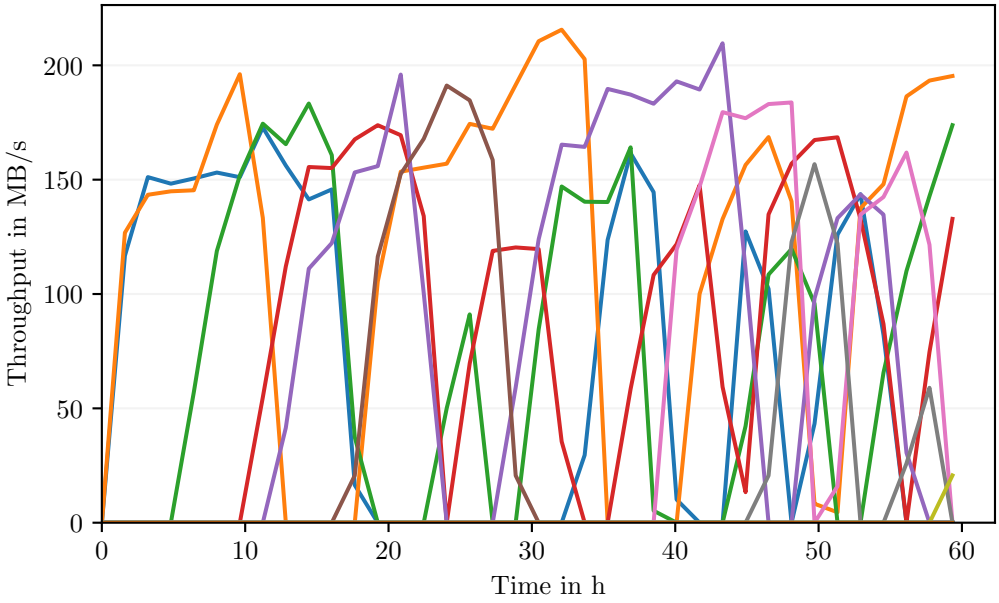
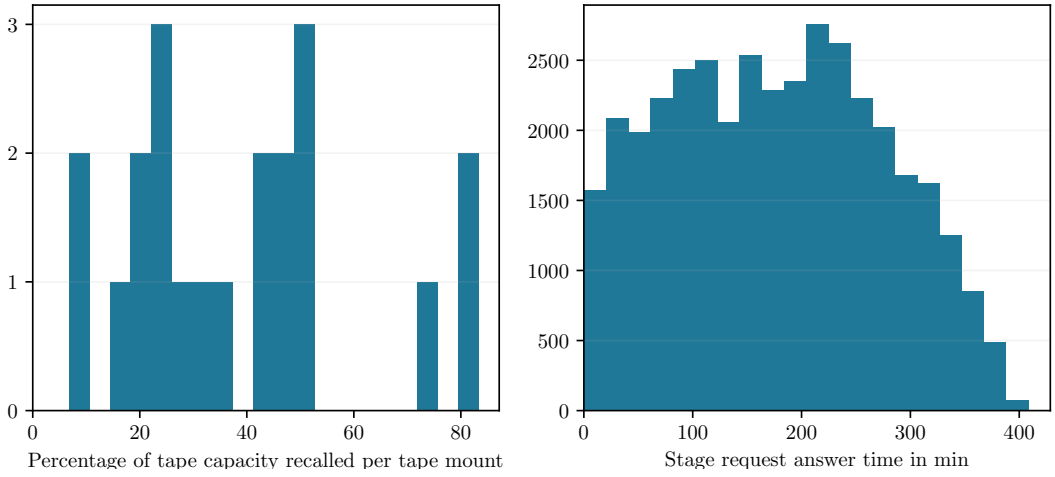


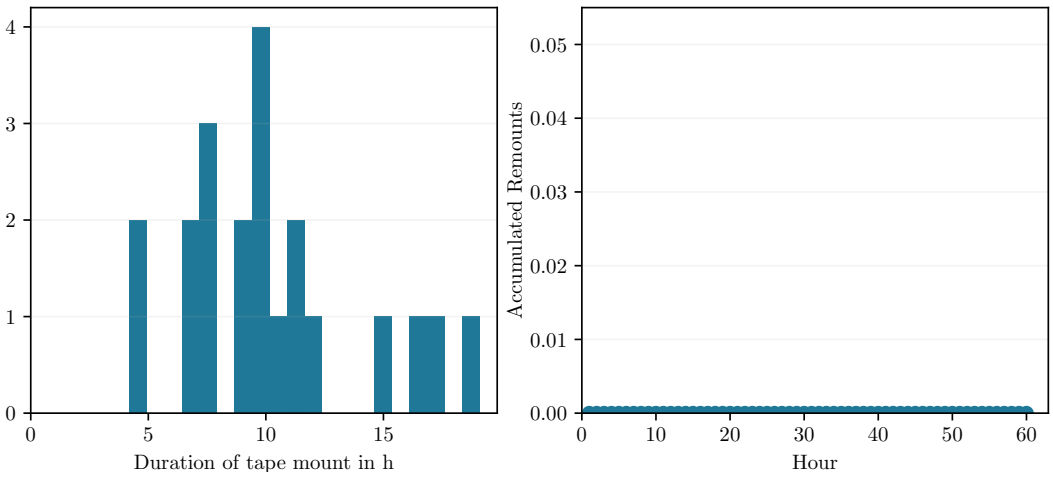
Figure A.4.: Simulation with requests for 2 tapes in parallel and a small-sized tape system request queue of 2000: Throughput per drive



(a) Percent of tape capacity recalled per mount

(b) Time for staging

Figure A.5.: Simulation with requests for 2 tapes in parallel and a small-sized tape system request queue of 2000: Percent of tape capacity recalled per mount and time for staging



(a) Duration of tape mounts (b) Accumulated number of remounts

Figure A.6.: Simulation with requests for 2 tapes in parallel and a small-sized tape system request queue of 2000: Duration of mounts and accumulated remounts

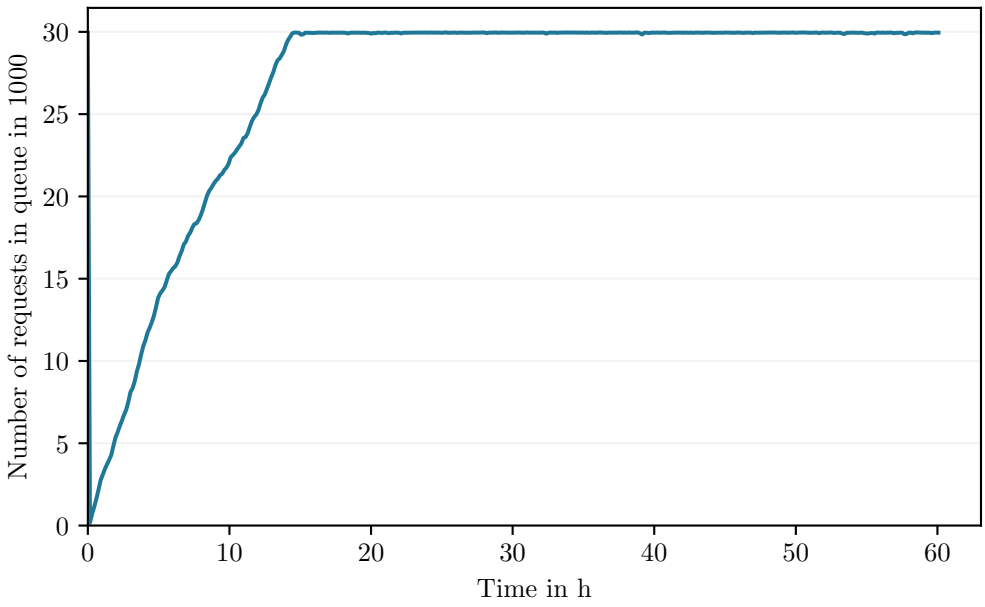


Figure A.7.: Simulation with requests by dataset and a large-sized tape system request queue of 30000: Requests in tape system queue over time

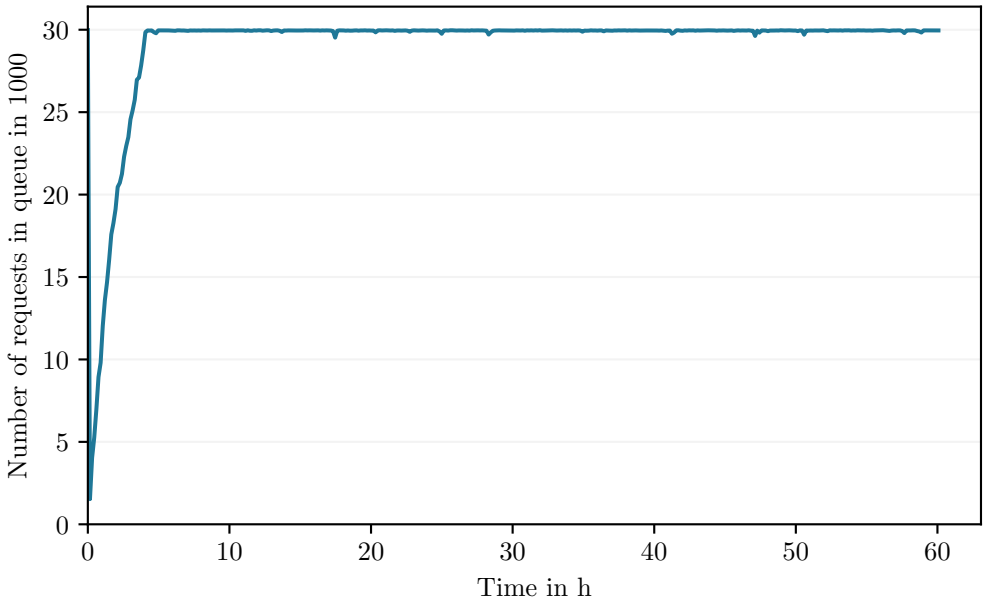


Figure A.8.: Simulation with requests for 2 tapes in parallel and a large-sized tape system request queue of 30000: Requests in tape system queue over time

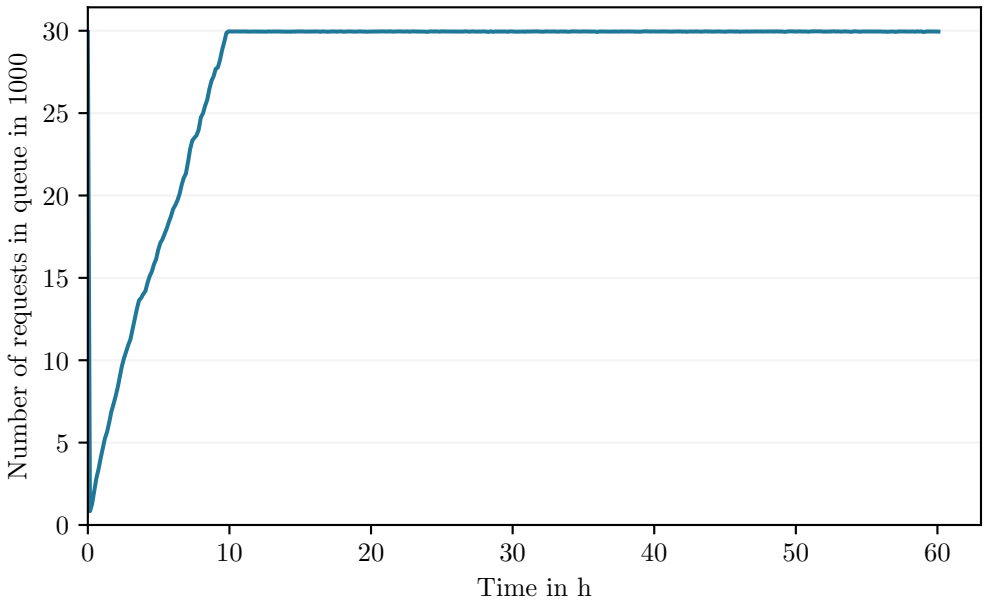


Figure A.9.: Simulation with requests for 12 tapes in parallel and a large-sized tape system request queue of 30000: Requests in tape system queue over time

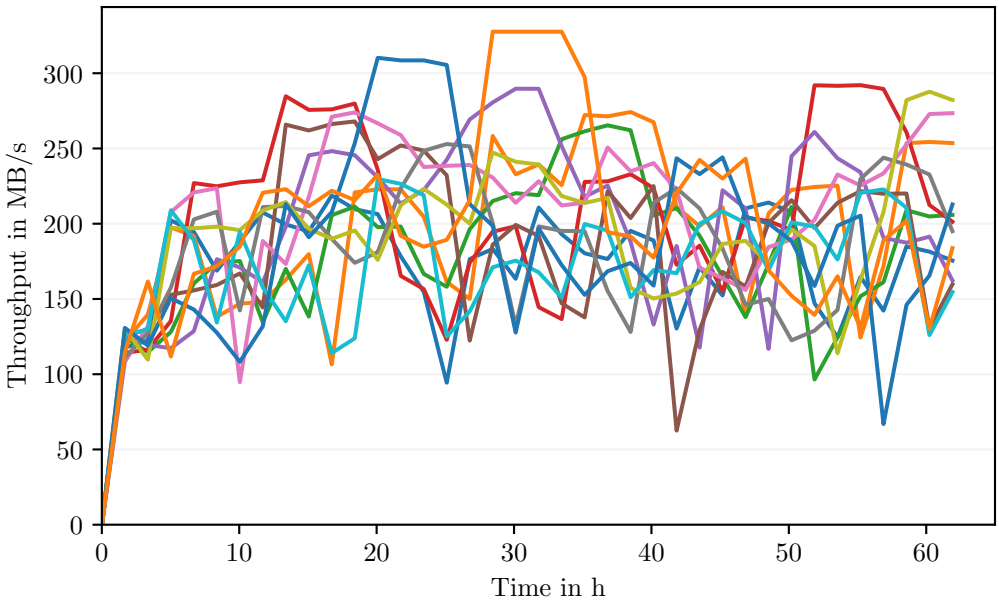
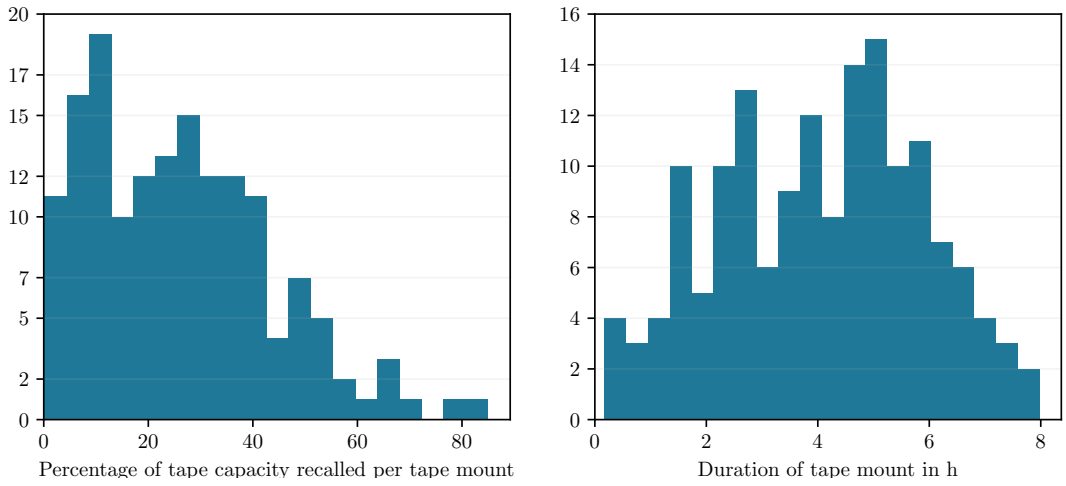


Figure A.10.: Simulation with requests for 12 tapes in parallel and a large-sized tape system request queue of 30000: Throughput per drive



(a) Percent of tape capacity recalled per mount

(b) Duration of tape mounts

Figure A.11.: Simulation with requests for 12 tapes in parallel and a large-sized tape system request queue of 30000: Percent of tape capacity recalled per mount and duration of tape mounts

Bibliography

[1] G. Aad et al. The ATLAS Experiment at the CERN Large Hadron Collider. *JINST*, 3, August 2008.

[2] G. Aad et al. The performance of the jet trigger for the ATLAS detector during 2011 data taking. *The European Physical Journal C*, 76(10), September 2016.

[3] Georges Aad, Tatevik Abajyan, B. Abbott, J. Abdallah, S. Abdel Khalek, Ahmed Ali Abdelalim, O. Abdinov, R. Aben, B. Abi, M. Abolins, et al. Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. *Physics Letters B*, 716(1):1–29, 2012.

[4] Johannes Albrecht, Antonio Augusto Alves, Guilherme Amadio, Giuseppe Andronico, Nguyen Anh-Ky, Laurent Aphecetche, John Apostolakis, Makoto Asai, Luca Atzori, Marian Babik, et al. A Roadmap for HEP Software and Computing R&D for the 2020s. *Computing and Software for Big Science*, 3(1):7, 2019.

[5] C. Allon, V. Breton, G. Cancio, P. Christakoglou, A. Connolly, F. Gaede, J. Hernandez, J. Kleist, H. Meinhard, and P. Sinervo. Computing Resources Scrutiny Group Report. Technical report, CERN, April 2019.

[6] ATLAS. Estimated Disk Resource Needs for 2018 to 2032. `https://twiki.cern.ch/twiki/pub/AtlasPublic/ComputingandSoftwarePublicResults/diskHLLHC_18.png`, 2017. Accessed: 2020-01-09.

[7] Martin Barisits, Thomas Beermann, Joaquin Bogado, Vincent Garonne, Tomas Javurek, Mario Lassnig, Andrea Manzi, Edoardo Martelli, Cedric Serfon, Tobias Wegner, et al. Evolution of the open-source data management system Ru-cio for LHC Run-3 and beyond ATLAS. In *EPJ Web of Conferences*. EDP Sciences, 2019.

[8] Bharat Bhushan. Historical evolution of magnetic data storage devices and related conferences. *Microsystem Technologies*, 24(11):4423–4436, November 2018.

[9] Ian Bird, F. Carminati, R. Mount, B. Panzer-Steindel, J. Harvey, Ian Fisk, B. Kersevan, P. Clarke, M. Girone, P. Buncic, et al. Update of the Computing Models of the WLCG and the LHC Experiments. techreport CERN-LHCC-2014-014 ; LCG-TDR-002, CERN, 2014.

[10] Oracle Corporation. Oracle Tape Cartridge T1000 T2 Data Sheet. `http://www.oracle.com/us/products/servers-storage/storage/tape-storage/storagetek-t10000-t2-cartridge-296699.pdf`. Accessed: 2020-02-01.

[11] R. A. Coyne, H. Hulen, and R. Watson. The High Performance Storage System. In *Supercomputing '93:Proceedings of the 1993 ACM/IEEE Conference on Supercomputing*, pages 83–92, November 1993.

[12] Ali E. Dashti and Cyrus Shahabi. Data placement techniques for serpentine tapes. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*. IEEE, 2000.

[13] Davis, Michael C., Bahyl, Vladímir, Cancio, Germán, Cano, Eric, Leduc, Julien, and Murray, Steven. CERN Tape Archive - from development to production deployment. *EPJ Web Conf.*, 214:04–015, 2019.

[14] D. C. Deatrich, Simon Xinli Liu, and Reda Tafirout. A high performance hierarchical storage management system for the Canadian tier-1 centre at TRIUMF. In *Journal of Physics: Conference Series*, volume 219. IOP Publishing, 2010.

[15] Johannes Elmsheuser and Alessandro Di Girolamo. Overview of the ATLAS distributed computing system. In *EPJ Web of Conferences*, volume 214. EDP Sciences, 2019.

[16] Lyndon Evans and Philip Bryant. LHC Machine. *Journal of Instrumentation*, 3(08), August 2008.

[17] V. Garonne, R. Vigne, G. Stewart, M. Barisits, T. Beermann, M. Lassnig, C. Serfon, L. Goossens, and A. Nairz. Rucio – The next generation of large scale distributed system for ATLAS Data Management. *Journal of Physics: Conference Series*, 513(4), June 2014.

[18] Vincent Garonne, Graeme A. Stewart, Mario Lassnig, Angelos Molfetas, Martin Barisits, Thomas Beermann, Armin Nairz, Luc Goossens, Fernando Barreiro Megino, Cedric Serfon, et al. The atlas distributed data management project: Past and future. In *Journal of Physics: Conference Series*, volume 396. IOP Publishing, 2012.

[19] IBM. IBM TS1155 Tape Drive. `https://www.ibm.com/downloads/cas/AZGD8GMB`. Accessed: 2020-02-01.

[20] IBM. IBM TS1160 Tape Drive. `https://www.ibm.com/downloads/cas/ZV2V7D8Q`. Accessed: 2020-02-01.

[21] Theodore Johnson and Ethan L. Miller. Benchmarking tape system performance. In *Proc. of Joint NASA/IEEE Mass Storage Systems Symposium*, 1998.

[22] Michael Kaczmarski, Tricia Jiang, and David A. Pease. Beyond backup toward storage management. *IBM Systems Journal*, 42(2):322–337, 2003.

[23] Peter Kunszt, Paolo Badino, R. Brito da Rocha, James Casey, Akos Frohner, and Gavin McCance. The gLite File Transfer Service. In *1st EGEE User Forum (March 2006)*, 2006.

[24] Jérôme Lauret and David Yu. ERADAT and Data Carousel systems at BNL: A tool and UI for efficient access to data on tape with fair share policies capabilities. *ACAT 2010 Proceedings*, 2010.

[25] Dmitry Litvintsev, Alexander Moibenko, Gene Oleynik, and Michael Zalokar. Enstore with Chimera namespace provider. *Journal of Physics: Conference Series*, 396(5), December 2012.

[26] H. Lukolf and H. F. Welsh. The Uniservo- Tape Reader and Recorder. In *Managing Requirements Knowledge, International Workshop on*, page 47. IEEE Computer Society, December 1952.

[27] Tadashi Maeno. PanDA: distributed production and distributed analysis system for ATLAS. In *Journal of Physics: Conference Series*, volume 119. IOP Publishing, 2008.

[28] Chris Mellor. LTO-8 tape media patent lawsuit cripples supply as Sony and Fujifilm face off in court. `https://www.theregister.co.uk/2019/05/31/lto_patent_case_hits_lto8_supply/`, May 2019. Accessed: 2020-01-03.

[29] Nicholas Metropolis and Stanislaw Ulam. The monte carlo method. *Journal of the American statistical association*, 44(247):335–341, 1949.

[30] Paul A. Millar, Olufemi Adeyemi, Gerd Behrmann, Patrick Fuhrmann, Vincent Garonne, Dmitry Litvinsev, Tigran Mkrtchyan, Albert Rossi, Marina Sahakyan, and Jurgen Starek. Storage for Advanced Scientific Use-Cases and Beyond. *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pages 651–657, March 2018.

[31] Tigran Mkrtchyan, Olufemi Adeyemi, Patrick Fuhrmann, Vincent Garonne, Dmitry Litvintsev, Paul Millar, Albert Rossi, Jürgen Starek, Sibel Yasar, et al. dCache-joining the noWORM storage club. In *EPJ Web of Conferences*, volume 214. EDP Sciences, 2019.

[32] Jussi Myllymaki and Miron Livny. Disk-Tape Joins: Synchronizing Disk and Tape Access. In *In Joint International Conference on Measurement and Modeling of ComputerSystems. SIGMETRICS '95/PERFORMANCE '95*, 1995.

[33] Oracle. StorageTek LTO-6 Tape Drives. `https://docs.oracle.com/cd/E38452_01/en/LTO6_Vol4_E1/LTO6_Vol4_E1.pdf`. Accessed: 2020-02-01.

[34] Oracle. StorageTek LTO-7, LTO-8 Tape Drives. `http://www.oracle.com/us/products/servers-storage/storage/tape-storage/033631.pdf`. Accessed: 2020-02-01.

[35] Oracle. TCartridge Label Orientation. `https://docs.oracle.com/cd/E24306_05/SLEUG/img/t104_085.png`. Accessed: 2019-12-30.

[36] Quibik. Computer Memory Hierarchy. `https://commons.wikimedia.org/wiki/File:ComputerMemoryHierarchy.svg`, 2010. Accessed: 2019-11-28.

[37] Thomas M. Ruwart. Analysis Report for Exascale Storage Requirements for Scientific Data. techreport, Sandia National Lab. (SNL-NM), Albuquerque, NM (United States), February 2018.

[38] Jamie Shiers. The Worldwide LHC Computing Grid (worldwide LCG). *Computer Physics Communications*, 177(1):219 – 223, 2007. Proceedings of the Conference on Computational Physics 2006.

[39] Arie Shoshani, Scott Klasky, and R. Ross. Scientific data management: Challenges and approaches in the extreme scale era. *Proceedings of SciDAC*, 2010.

[40] SimPy. SimPy 3.0.11 Documentation – Overview. `https://simpy.readthedocs.io/en/latest/`. Accessed: 2020-02-02.

[41] Oracle Storagetek. Storagetek T10000C Tape Drive Data Sheet. `http://www.oracle.com/us/products/servers-storage/storage/tape-storage/t10000c-drive-ds-289750.pdf`. Accessed: 2020-02-01.

[42] Oracle Storagetek. Storagetek T10000D Tape Drive Data Sheet. `http://www.oracle.com/us/products/servers-storage/storage/tape-storage/t10000d-ds-1991052.pdf`. Accessed: 2020-02-01.

[43] P. Toth. Dynamic programming algorithms for the Zero-One Knapsack Problem. *Computing*, 25, March 1980.

[44] WLCG. WLCG Resource, Balance and Usage Site Topology. `https://gstat-wlcg.cern.ch/apps/topology/`. Accessed: 2020-01-18.

[45] WLCG. WLCG Tiers. `https://wlcg-public.web.cern.ch/sites/wlcg-public.web.cern.ch/files/inline-images/WLCG-TiersJun14_v9-transp.png`. Accessed: 2020-01-10.

[46] www.ltoultrium.com. LTO Roadmap. http://www.ltoultrium.com/wp-content/uploads/2019/03/lto-roadmap.jpg. Accessed: 2020-01-01.

[47] www.zookeeper.apache.org. Apache ZooKeeper. https://zookeeper.apache.org/. Accessed: 2019-07-22.

[48] David Yu, Guangwei Che, Tim Chou, and Ognian Novakov. Best Practices in Accessing Tape-Resident Data in HPSS. In *EPJ Web of Conferences*, volume 214. EDP Sciences, 2019.

[49] David Yu and Jérôme Lauret. Efficient Access to Massive Amounts of Tape-Resident Data. In *Journal of Physics: Conference Series*, volume 898, 2017.

Declaration of Academic Integrity

I hereby declare that I have authored this thesis independently and that I have not used any other resources than the ones indicated. All thoughts taken directly or indirectly from external sources are properly denoted as such.

This thesis was not previously presented to another examination board and has not been published.

City, Date Lea Morschel