Fair-share scheduling algorithm for a tertiary storage system

Pavel Jakl[1], **Jérôme Lauret**[2], **Michal Šumbera**[1]

[1] Nuclear Physics Institute ASCE, Prague, CZ 18086, Czech Republic
[2] Brookhaven National Laboratory, Upton, NY 11973, USA

E-mail: pjakl@bnl.gov

Abstract. Any experiment facing Peta bytes scale problems is in need for a highly scalable mass storage system (MSS) to keep a permanent copy of their valuable data. But beyond the permanent storage aspects, the sheer amount of data makes complete data-set availability onto live storage (centralized or aggregated space such as the one provided by Scalla/Xrootd) cost prohibitive implying that a dynamic population from MSS to faster storage is needed.
One of the most challenging aspects of dealing with MSS is the robotic tape component. If a robotic system is used as the primary storage solution, the intrinsically long access times (latencies) can dramatically affect the overall performance. To speed the retrieval of such data, one could organize the requests according to criterion with an aim to deliver maximal data throughput. However, such approaches are often orthogonal to fair resource allocation and a trade-off between quality of service, responsiveness and throughput is necessary for achieving an optimal and practical implementation of a truly faire-share oriented file restore policy.
Starting from an explanation of the key criterion of such a policy, we will present evaluations and comparisons of three different MSS file restoration algorithms which meet fair-share requirements, and discuss their respective merits. We will quantify their impact on a typical file restoration cycle for the RHIC/STAR experimental setup and this, within a development, analysis and production environment relying on a shared MSS service.

1. Introduction

The amount of scientific data generated by simulations or collected from data acquisition systems for a large scale experiment have reached levels that cannot be stored in the researcher's workstation or even in his/her local computer center. The amount of data is no longer measured in megabytes, but instead their scale is typical order of magnitudes more from TeraBytes (TB) to PetaBytes (PB). High energy and Nuclear Physic (HENP) experiments are well known examples of communities with PetaByte-scale-per-year data problems.
The Solenoidal TRacker at RHIC (STAR) experiment one of the four physic's experiments at Brookhaven National Laboratory (BNL) Relativistic Heavy Ion Collider (RHIC) is an example of an experiment that has reached this data regime. In this kind of experiment and science, most physics topics are statistically driven; in order to have a statistically significant data sample, the experiment has generated several Petabytes (10^{15} of byte) data since May/June 2000 when the first collision was recorded. This data is stored in over 10 millions of files for the past 9 years of collider's running. Figure 1 shows the growth of the size of data samples from 2005 to 2012 as projected [1] to sustain the STAR's physics program and planned upgrades, which generate the increase of data to be

taken. Any experiment facing such truly unprecedented amount of data and growth profile needs a highly scalable tertiary storage system to keep a permanent copy of the data. Tertiary storage systems are often used in *"write once, read rarely"* applications such as storage of large data-sets (data is retrieved and accessed in a seldom manner), backup and archiving. It is however becoming of common practice to use tertiary storage system to store active data as the cost of purchasing and maintaining fast secondary storage is highly exceeding acceptable limit. In general, tertiary storage systems offer cheap and reliable storage, but on the other hand, they suffer from slow access. As data needs to be available on a fast storage location to satisfy the analysis demand, this fact impedes the possibility to migrate data from MSS to disk. We will retain two possible strategies: using a centralized disk pool or using a distributed disk pool approach. While standard protocols (NFS, iSCSI and other) exists to mount and access centralized pools, distributed disk pools is of more interest as it would allow the recovery of scattered and cheap disk space. One of the systems that offers aggregation of a highly distributed set of storage is called Scalla/Xrootd [2]. Scalla/Xrootd has the possibility to restore requested data (missing from the storage pool it manages) from tertiary storage system and such capability is at the very heart of the problem we propose to study and resolve. The ultimate question is: How to achieve reasonable access time when analyzing and processing data which has to be retrieved from the tertiary system and this, in a multi-user environment?.

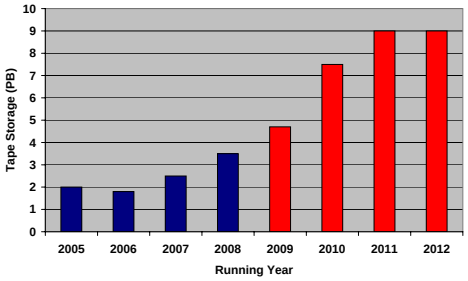


Figure 1. *STAR's experiment tape usage(blue) and prediction(red) of raw data for the upcoming years as shown in [1].*

In this paper, we propose to study a set of three different scheduling algorithms accommodating fairness of requests and maximal performance. The proposed algorithms are further evaluated and compared on the simulator of the tertiary storage system reflecting the STAR's physics data volume and the use pattern of STAR's physicists.

2. Previous work

In [3] and [4], we have reported that Scalla/Xrootd lacks coordination when requesting files from tertiary storage system and that it can result in either poor overall performance, or even the total collapse of the tertiary storage system. The usage of the system then becomes inefficient; user's tasks and jobs waiting for files to be served from the tertiary storage system are at the same time locking a CPU slot from a resource management system (batch). Consequently, the entire facility load usage drops and one loses the battle of balancing cost and optimization.
Therefore, the focus was initially concentrated on understanding key parameters of the tertiary storage system having an effect on its performance. The first of the most costly aspects when dealing with robotic tertiary storage system is the time it takes to switch a tape. The second is the latency associated with searching for a file on a tape depending on the tape size, the location of the file on the tape, the search speed but more importantly the size of the file. We especially identified that the speed of retrieval is directly proportional to the file size to be restored [3]. Working toward avoiding or minimizing these delays results in a large performance gain. In the next section, we will explain how to mitigate the issue associated to frequent tape mount and dismounts.

3. Facing difficulties of scheduling requests for the tertiary storage system

Difficulties rely in the fact that requests can be made for many files at different times and in parallel within a given time interval can span many different data-sets, or different segments of large data-sets. In all cases, the data-sets are spread over many tapes. If treated sequentially,

each individual request may cause a dismount of the currently loaded tape for retrieving the tape (and files) of interest. The logic on how to organize this queue of requests is the key to a winning strategy. We will rely on a scheduling techniques to attempt to resolve this problem.
The ultimate goal of the scheduling algorithm is to organize requests according to several criteria and deliver a sustained data throughput along the maximal quality of service keeping in mind that all users have ideally identical allocations for the provided service (i.e. fair-share for users). The criteria are, for example, parameters influencing the performance in order to accomplish the sustained data throughput, but also user's weight (e.g. priority) determining how the system's allocation should be distributed among users. The algorithm should consider many objectives during its design. In particular, an algorithm should consider fairness, efficiency, response time and throughput and achieve a balance between those without creating resource starvation or resource blocking. The major concern of tertiary storage performance is the high latency and therefore our main focus is reducing the amount of time that a given request has to wait before it is serviced (i.e. minimizing the total waiting time). The quality of service objective is to have all users have almost identical delay in request satisfaction.
One could be tempted to consider minimizing the number of tape mounts through a system in which a tape with most requests is mounted preferentially, and not unloaded until all files from it are retrieved. Such system would provide maximal throughput, but have the disadvantage to lead to resource starvation. A request for a single file on a tape may never be satisfied if the queue of requests is constantly filled with newer requests leading to more than one file per tape. The importance of the size of the requested files relates to the fact that a tape drive has to perform additional tape positioning (tape marks) and seeks. This overhead destroys the data streaming speed of the MSS. Along this line of thought, one can also raise an opinion that it can be more beneficial to service requests in bundles; servicing the biggest files first before requesting smaller files. The value of this consideration is determined by the relative values of the tape switch time and seek time. However since the smaller files would always have the lowest priority, this is similar to the case of servicing tapes with most file requests first, and would lead to resource starvation and locking.
The problem becomes even more complex if one combines the two aspects of size and number of files per tape. If the switch time is large compared to the expected or average seek time, then it is unlikely that switching to a new tape before servicing all requests on the currently loaded tape will be beneficial. On the other hand, if the switch time is small to the expected or average seek time, then servicing requests file by file, regardless of any necessary tape switching may result in shorter waiting times and much higher data throughput.
The seeking attribute also has another dimension, the order in which to service the requests for a loaded tape. The order of servicing requests on the tape is mainly based on the location of files on the tape. Obviously, the simplest solution is to service them in the order of location on the tape and therefore reduce the amount of seeking or fast-forwarding and rewinding. However, we will not consider finding efficient schedules for a loaded tape, instead we will delegate this problem to the mechanism implemented by the tertiary's storage system implementation used at BNL (i.e. HPSS).
In general, we expect that eliminating switches is more important due to the large magnitude of switching cost(robotic arms, tape rewind) and potentially harmful switch (tape worn out, tape destruction due to robotic positioning mistakes or other mechanical problems). Therefore, we strongly believe that scheduling for the tertiary storage system should be aimed to minimize the number of tape switches (i.e. maximum number of files per tape mount) rather than scheduling fewer seeks (i.e. large files) but without impacting the overall throughput so negatively that it would cause resource starvation.

4. Fair-share scheduling algorithms

In this section, we will define several algorithms that will be used to evaluate our findings and needs. The authors of [5] have shown that the problem of scheduling for multiple drives is NP-Complete. They reduced the scheduling problem to the known NP-Complete problem of *Scheduling to Minimize Weighted Completion Time.*
Therefore, we will present 3 different heuristics based on our performance study keeping in mind achieving a certain level of quality of service:

- The First Come First Serve (FCFS) algorithm
- The Weighted Fair Queuing (WFQ) algorithm
- The Weighted Fair-share Grouping (WFSG) algorithm

The algorithms are explained in the order of their complexity, FCFS is the simplest algorithm while WFSG is the highest level of scheduling algorithm incorporating many parameters. All algorithms assume that the practical implementation will provide for each request at least the following fields: the id of the desired tape, the data size to be requested and also a user which submitted the request. Requests are coming with a mean arrival rate, following a particular distribution and are placed into the queue of waiting requests. An algorithm has to scan the queue of requests and extract requests or batch of requests that will be submitted to the tape system.

4.1. The First Come First Serve (FCFS) algorithm

The simplest scheduling scheme is FCFS, sometimes also known as **FIFO** (*First In First Out*). The requests are executed in the order they arrive and sent as such to the tape system. This algorithm is expected to have poor performance and QoS for heavy workloads. It will lead to large number of costly tape switches and the possibility that a single user with a large request locks the system for a large amount of time. In other words, the QoS of this algorithm is limited by the fact that users are served in the order which arrived to the tape system. Furthermore, the tape drive will remain idle for large periods of time, waiting for tape switches resulting in mounting/unmounting/rewinding tapes. The QoS of this algorithm is limited by the fact that users are served in the order which arrived to the tape system. This provides a first and simplest level of fairness (service in the order of arrival). Essentially, FCFS is simplistic scheduling algorithm and it is mainly used for base-lining and comparison purposes.

4.2. The Weighted Fair Queuing (WFQ) algorithm

The idea of **Weighted Fair Queuing** algorithm [6] comes from packet-switched computer networks (routers, switches) where this scheduling technique allows different data streams, represented by packets, to fairly split the available communication channel. WFQ is a generalization of *Fair Queuing (FQ).* FQ allows to fairly share the link capacity by extending the conventional FIFO queuing approach. The model is designed in such a way that an ill-behaved flow (having unfairly many or large data packets) punishes itself and not other flows. This is achieved by dividing the available total resource into separate FIFO queues. Each of the FIFOs holds the packets of one flow. Those flows are defined for instance by source or destination IP addresses.
In WFQ, a bandwidth R is divided in N FIFOs and each gets a weight of W_i (i = 1 .. N). It is important to realize that contrary to FQ, WFQ is NOT telling what the W_i should be, but only that for element i, the *share* is equal to 1. Regular FQ is a special case of WFQ for equal weights of the queues. The WFQ algorithm can be easily adopted to our studied problem. The bandwidth R is considered as N requests which have to be submitted in periodic times to the tape system. The queues are defined as associated to distinct users asking for different files and creating each separated FIFOs. The weights for queues are assigned according to their

pre-defined priorities of users. The FIFO for our case is not strict FIFO but weighted by the location on tapes (an attempt to reduce the number of switches).

$$S_i = R \times \frac{W_i}{\sum_{k=1}^{N} W_k} \tag{1}$$

At the beginning, we look at the assigned priorities of users and calculate the number of slots from N available slots which may be allocated. Then we calculate the fraction of N for each FIFO giving an advantage to requests from the same tape (first key parameter of performance). This is done by looking ahead for $user_i$ at files which may be found on the same tape. This algorithm is currently in use in STAR's environment.
This approach still fulfills the consideration that users with many requests punish only themselves and also that the resource starvation is avoided. On the other hand, the algorithm does not differentiate between small and big files (second key parameter of performance), so the share is the same. Another downside may be that the number of files per tape is not favored enough, since it is applied in each separated FIFO only; the content of the mounted tape should be taken into account from the view of the entire queue. The last and most important finding is that a user with too high priority can still create resource locking, and requests from other users may not be scheduled for a long time. Therefore, relative priorities have to be carefully considered in this approach. This later problem can be resolved by considering a usage history which will then provide full fair-share. We have taken into account the approach of usage history in the next algorithm.

4.3. The Weighted Fair-share Grouping (WFSG) algorithm

The problems associated with WFQ is partially addressed by implementing a charging mechanism [7]. Typically, charging mechanism involves allocation of a certain budget to each user and as users consume resources, they are charged for their use. In the *fixed-budget* model, each user has a fixed size budget allotted to him. As the user uses resources, the budget is reduced and when it is empty, he cannot use the resources at all (assuming that others users have their requests in the queue). Budget consumption models often also have a time dependence: after some time, the budget re-appears and the user could use some resources again (dynamic share). Our approach is to regard each user as having an entitlement to fair share of the system, relative to other users. Then the task of an accounting system (being part of the scheduling system) is to ensure that user's entitlement is decreased as their requests are satisfied.

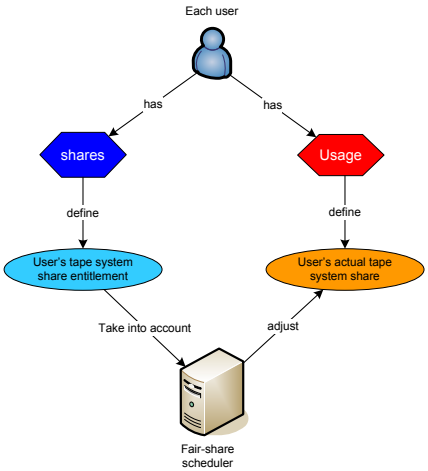


Figure 2. *The overview of the interaction between the user (his shares and usage) and the fair-share scheduler.*

Figure 2 shows two factors that will be incorporated in our fair-share scheduler: user's shares and user's usage. A *user's shares* indicate his entitlement to retrieve files from the tape system. The more shares a user has, the greater his entitlement. If user A has twice as many shares as user B, then in long term, user A will be able to do twice as much work as user B. On the other hand, each user has a *usage* history, which reflects how much work the user has done over period of the time, and which is *decayed* [8].
In practice, and in order to have the decaying process flexible enough, we define 3 parameters:

TIME_WINDOW, N_WINDOWS and DECAY. The time is broken into a number of distinct fair-share windows, where the TIME_WINDOW specifies the duration of each window while the N_WINDOWS parameter indicates the number of windows to consider. The DECAY parameter limits the impact of fair-share data according to its age. Decay parameter is specified as a value between 1 and 0 where a value of 1 (the default) indicates no decay should be specified. The smaller the number, the more rapid the decay is. We then use the following formula:

$$\text{Decayed Usage} = \text{Usage}_i \cdot DECAY^i \text{ (i = window number)} \tag{2}$$

The metric for our scheduler is the number of mega-bytes delivered to a user. The accounting module updates each fair-share window until it reaches the window boundary, at which point it rolls the fair-share window and begins updating the new window. The module also keeps track of utilization information for each user, as well as for the total system. The user's consumption is then computed according to the following formula:

$$Usage\_history^{User} = \frac{\left(\sum_{i=0}^{N} usage_i^{User} \cdot DECAY^i\right)}{\left(\sum_{i=0}^{N} usage_i^{total} \cdot DECAY^i\right)} \tag{3}$$

$$where$$

$$Usage_i^{User} = \text{is usage consumption of user in window number i}$$

$$Usage_i^{Total} = \text{is usage consumption of all user in window number i}$$

$$N = \text{number of windows specified by N\_WINDOWS parameter}$$

$$DECAY^i = \text{DECAY parameter exponentiated to window number i}$$

Having defined usage history, we need to incorporate it with the key performance parameters that influence the data throughput. At the beginning of this section, we have explained that the key relies in the minimum switching model, i.e. restrict tapes as much as we can, followed by the next important consideration that needs to be integrated: the size of requested files. However, the algorithm always needs to minimize switching over any other considerations.
We have decided on the cost model for this study. Each request R_i has an assigned cost C_i that takes to restore it from the tape system. For each request we then define a weight that is used to pick the *best* schedules. The weight is simple defined as the inverse value of the cost, taking into account number of shares that are allotted to the particular user.

$$W_i = shares^{user} \cdot \frac{1}{C_i} \tag{4}$$

The definition of the overall cost for our case is composed of 3 fractional costs being based by already mentioned parameters:

- C_i^{tape} - number of files from the same tape
- C_i^{size} - the size of requested file
- C_i^{usage} - usage history of the user submitted the request

The composition of the overall cost is defined as a flexible combination of those fractional costs:

$$C_i = \textbf{switch\_factor} \cdot C_i^{switch} + \textbf{file\_size\_factor} \cdot C_i^{size} + \textbf{usage\_factor} \cdot C_i^{usage} \tag{5}$$

where

$$\textbf{tape\_factor} + \textbf{file\_size\_factor} + \textbf{usage\_factor} = 1 \tag{6}$$

and

tape_factor = constant factor that characterizes an importance of tape switches

file_size_factor = constant factor that characterizes an importance of file size

usage_factor = constant factor that characterizes an importance of usage history

The reason for the equation 6 equal to 1 is that all fractional costs are ranging values from 0 to 100. This gives the possibility to weight costs according to their importance in the overall cost keeping the overall normalization in check. Each fractional cost has assigned computational function. Each of the functions follows the notion of the cost, i.e. the less number of file from the same tape, the higher the cost, smaller size higher the cost etc. Let's consider N requests R $= (r_1 \dots r_N)$:

$$C_i^{tape} = \underbrace{\frac{file\_per\_tape\_max}{\mid \forall j \in \hat{N} : r_i^{tape\_id} = r_j^{tape\_id} \mid}}_{\#\, files\, with\, tape\_id} \times \underbrace{\left(\frac{100}{\mid R \mid}\right)}_{\text{normalization element}} \tag{7}$$

$$C_i^{size} = \frac{file\_size\_max}{\underbrace{r_i^{file\_size}}_{\text{file size of request i}} \mid} \times \underbrace{\left(\frac{100}{\sum_{j=1}^{N} C_j^{size}}\right)}_{normalisation\, element} \tag{8}$$

As previously mentioned, the C_i^{usage} is equal to the usage history in the equation 3. One can observe that the cost formula has been designed to be fully configurable, allowing the possibility to give a possibility to force user's dimension among the performance or vice-versa. If an administrator adjusts the tape_factor equal to 1 and others equal to 0, the algorithm would convert to the shape of WFQ where each queue is represented by files from the same tape. This approach will certainly achieve the best performance, but lacks from any kind of fairness.

5. Evaluation of scheduling algorithms

All scheduling algorithms are heuristics that have been built from our observations and previous studies of the base principles for fairness, throughput and latencies. Each of the algorithms has its own interpretation of the solution how to achieve the fair-share scheduling of requests. We would now like to evaluate those algorithms in a realistic environment, and quantify the performance of our described and proposed solutions.

An important and desired viewpoint is to see how the algorithms behave under particular varying request rate where the behavior can be characterized in many ways. Parameters of interest can include data rate (MB/s), delay of each request, the number of requests successfully delivered and many others. This of course bears a technical challenge in the evaluation process, since each evaluation may result in many hours, days and weeks of running time on the real system such as HPSS (tertiary storage system used at STAR experiment), not even speaking of the disruptive nature of such "testing" in a real-time setup. Instead of testing our assumption on

a live system, we therefore built a "*simulator*" of the tape system that is able to emulate the basic characteristics of the tape system. Hence, our evaluation of the algorithms is performed on this simulator.

5.1. Design model of the simulator

For the design of the simulator, it is first necessary to abstract from the real system components and their interactions that extract the essential behavior which are considered most important to emulate the real system without having to implement its full complexity. In other words, we need to define and design elements of our simulator that characterize a tape system and then predict how long the restore operations take. These times are predicted by defining a tape system hardware model where the final simulator takes them into account. A detailed explanation of the model is available at reference [9]. In brief, the model contains a robotic arm operation (load/unload of tape from/to tape drive) and a tape drive operations (seeking, loading/unloading, streaming).

Our simulator of the tape system is **Monte Carlo continuous time - discrete event based simulator** that uses Monte Carlo method to randomize operation time of components or to generate requests arrivals that occurs at discrete points of the simulation time. Each event holds a certain operation that should be performed on a specific component at an exact point of the simulation time and therefore changes the state of the component. The event mostly schedules other event (or itself in respect to the future) during the operation period that is a result of the event execution. Such events can for example be: *LoadTapeEvent*, *SeekFirstFileEvent*, *SeekNextFileEvent*, *TransferDataEvent* etc. Figure 3 shows the description of the simulation

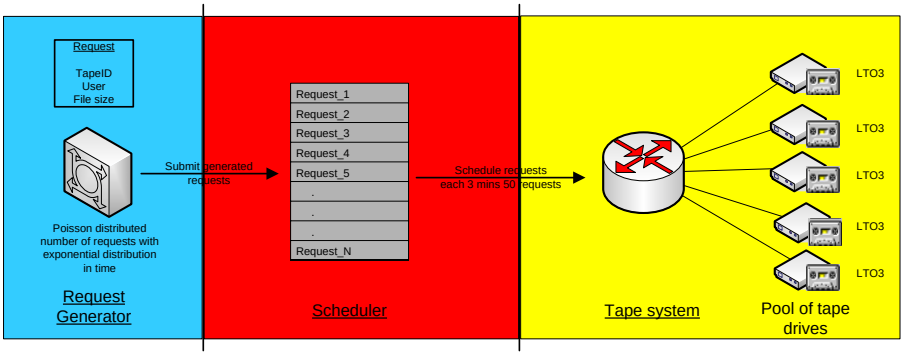


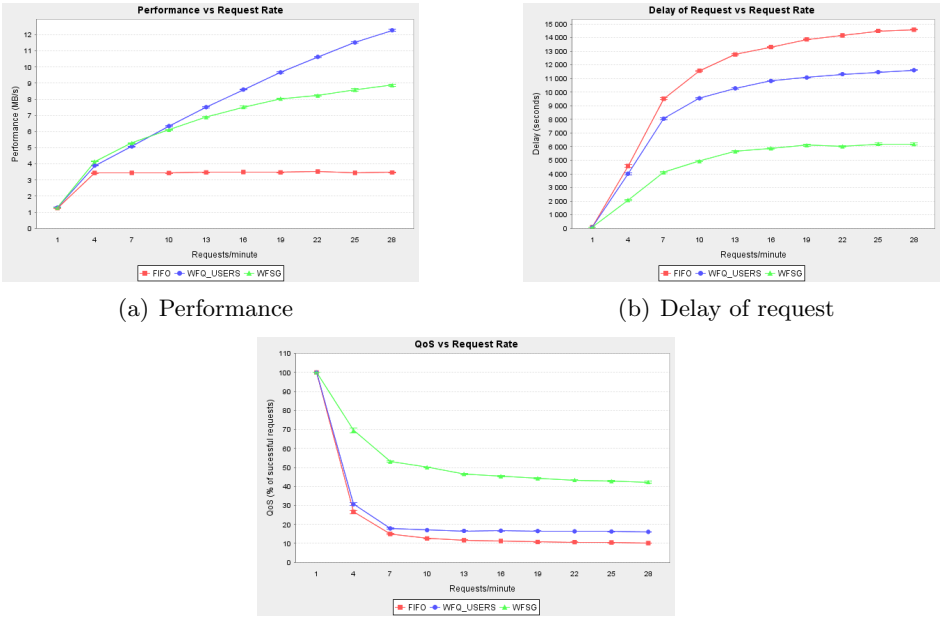
Figure 3. *The layout of the simulation work-flow with its three main parts.*

work-flow. One of the opened questions can for example be: "*Where and how scheduling algorithms fits into the simulation ?*" or "*How the requests are generated and than forwarded to the tape system simulator ?*". Requests are coming from users where in this case are represented by the **Request generator**. The generator is able to produce exactly the same workload as can be seen in the STAR environment. The sizes of the files and the locations of files on the tapes follow observed distributions (empirically extracted from usage accounting) serving as input to our generator. The generated requests are then queued and scheduled by a component called **Scheduler**. The scheduler is where all algorithms are coded. Its responsibility is to submit a selection of requests in each consecutive time periods to the **tape system** (more precisely , to the simulated tape system queue). In the simulated tape system, each tape drive obtains a request for processing a file and this is coded by calling a specific function responsible for emulating the tape drive response.

6. Results of the evaluation

While we noted that an evaluation could look at the problem in many dimensions, our evaluation was based on the 3 following parameters. **Performance** measured in MB/s that the tape system

can achieve using the given algorithm. This characteristic quantifies of how much data can be transferred for some period of time. The **Delay of request** is measured in seconds, defined as the average time of delay per request in the system. The main idea is that some of the algorithms can have high delay in order to achieve better performance and the observation of this parameter (especially in a context of an active job waiting for a file) is directly relevant to our problem. The purpose of this parameter is to quantify the responsiveness of the system. The **QoS (Quality of Service)** measured as a percentage value of successful deliveries. A delivery is considered successful if it has been satisfied within a pre-defined timeout period (e.g. 1 hour). To some extent, QoS is correlated with the delay of requests (the shorter the average delay, more likely we are to see the QoS high). Figures 4 shows our three parameters ((a) performance,



(a) Performance

(b) Delay of request

(c) Quality of service (QoS)

Figure 4. *Results of the simulation on defined algorithms in respect to defined three evaluation parameters as a function of the request rate.*

(b) delay and (c) QoS) as a function of the request rate, or the number of requests that can arrive per minute. In figure 4(a), we observe a rapid saturation of performance as a function of request rate in the FIFO (FCFS) case. This is due primarily to the fact that chaotic and non-optimized requests would in average saturate to an maximal value with the system unable to satisfy a higher rate not scale (finite bandwidth with congestion effects can only lead to a fixed throughput). In contrast, we observe that the WFQ algorithm exhibits the highest performance comparing to the other algorithms, but seems to suffer in delays and QoS when compared to WFSG. In fact, in WFQ the average delay of requests (figure 4(b)) reaches high values close to FIFO algorithm as soon as we reach 5-7 requests per minutes. There, QoS dramatically drops for both FIFO and WFQ and reaches a low plateau of 10 to 20% at rates higher than 10 requests a minute (figure 4(c)). In contrast, WFSG is slightly less performing than WFQ but exhibits by far the least delays - nearly half the amount comparing to the other two algorithms. However, as noted before, the cross correlation of QoS and delays influence this observation (since the delay is short, a request is not likely to exceed given timeout interval). The delay per request therefore seems to be the most important dimension that one should pay attention to, from the perspective of our problem of running jobs waiting for files to be delivered. It is clear then that the WFSG algorithm surpasses the other algorithms, and indicates the best results in terms of minimizing delays and have the highest QoS with minimal loss of performance.

7. Conclusion

One of the most costly aspects of dealing with robotic tertiary storage system are the time it takes to switch a tape, delays associated to tape marks and seeking files on tapes and performance scaling as a function of file size. Working toward avoiding or minimizing these delays results in a large performance gain. To reach this goal, a scheduling algorithm can be used having the flexibility to stage files in tape-optimized order and prioritizing files with the biggest size. However, the algorithm has to be fair enough in respect to the users, where ideally no user is resource starved (indefinitely waiting for his requests to be completed). One can naively consider servicing users in FIFO order, but such approach is neither efficient nor fair enough. To overcome such limitation, one may turn to scheduling algorithm to resolve the problem of restoring files from MSS. A natural next step for example is to define for each user a separated queue and service those queues according to user's priorities along the line of tape optimizations in each separated queue. This scenario still results in poor fairness because users with higher defined priority can again saturate the system easily, not allowing other users to satisfy their requests.

Through simulations of a tape system using realistic input based on usage statistics, we demonstrated that the first, in first served approach is leading to poor saturating performance while another, more complex, algorithm would lead to far better performance and increased quality of service. The approach of separate queue mechanism (WFQ) may provide optimal performance, but on the other hand it creates excessive average delays and poor quality of service to most users making the system un-usable in a context where processing jobs would be waiting for files within a finite amount of time. This algorithm may however be used in bulk file transfer or file restore context (that is, with no need for a timeout). In our study, we showed that an algorithm incorporating elements of shares, priorities and historical usage of our users achieves a reasonable overall performance while minimizing average delays for a request to be satisfied hence, maximizing quality of service. The inclusion of usage history of the user and priority seem essential in reaching good response of the entire system. The usage history accounts user's usage and adjusts the cost of the request according to this usage.

We conclude the WFSG algorithm is the best suitable algorithm for satisfying the need of a multi-user environment where files are being requested for restore from MSS from many ongoing analysis made through a storage aggregation such as Scalla/Xrootd. We also note that a site best performance may be a combination of WFQ (bulk restore preserving some fairness and maximizing performance) and WFSG (fully fair and suitable to analysis subject to "files on demand").

8. Acknowledgment

This work was supported in part by the HENP Divisions of the Office of Science of the U.S. DOE; the U.S. NSF and by grants of GACR 202/07/0079 and LC07048.

References

[1] Brookhaven National Laboratory 2006 The RHIC Mid-Term Plan Tech. rep. BNL
[2] Hanushevsky A and Weeks B 2006 Scalla: Scalable Cluster Architecture for Low Latency Access, Using xrootd and olbd Servers Tech. rep. SLAC
[3] Jakl P *et al.* 2008 *J. Phys. Conf. Ser.* **119** 072019
[4] Jakl P, Lauret J, Hanushevsky A, Shoshani A and Sim A 2006 *Proc. of Computing in High Energy and Nuclaer Physics (CHEP'06)*
[5] Prabhakar S, Agrawal D and Abbadi A E 2003 *Distrib. Parallel Databases* **14** 255–282 ISSN 0926-8782
[6] Stiliadis D and Varma A 1998 *IEEE/ACM Trans. Netw.* **6** 611–624 ISSN 1063-6692
[7] Kay J and Lauder P 1988 *Commun. ACM* **31** 44–55 ISSN 0001-0782
[8] Jackson D, Snell Q and Clement M 2001 *Lecture Notes in Computer Science* **2221** 87–98
[9] Jakl P 2008 *Efficient access to distributed data: A "many" storage element paradigm* Master's thesis Czech Technical University, FNSPE Trojanova 13, Prague