Editorial Manager(tm) for Journal of Physics: Conference Series
Manuscript Draft

Manuscript Number: CHEP164R2

Title: TReqS: The Tape Request Scheduler

Article Type: Poster

Corresponding Author: Andres Gomez Casanova

Corresponding Author's Institution: CC IN2P3

First Author: Jonathan Schaeffer

Order of Authors: Jonathan Schaeffer;Andres Gomez Casanova

TReqS: The Tape REQuest Scheduler

Jonathan Schaeffer [1], **Andrés Gómez Casanova** [2]

[1]IUEM, rue Dumont Durville, 29280 Plouzané, FR
[2]CC-IN2P3, 21 av Pierre de Coubertin, 69100 Villeurbanne, FR

E-mail: jonathan.schaeffer@univ-brest.fr, andres.gomez@cc.in2p3.fr, treqs@cc.in2p3.fr

Abstract. TReqS is a new layer in the storage infrastructure at CCIN2P3 controlling the read operations for data stored on tapes in HPSS (a HSM) by centralizing all requests. The development and deployment of TReqS at CCIN2P3 has brought major advances in data access services on several aspects. First, by sorting and scheduling tape mounts, we radically reduce the waiting time due to media movements in the tape library. Second, in a mutualised environment where tape drives are globally available to all users, TReqS ensures the fair share of the drives. Finaly, TReqS softens the impact of unpredictable grid jobs and transfers on the actual tape library and efficiently masks the data access complexity to the end user.

1. Introduction

At CCIN2P3, HEP experiments use tape technology to store not only archive data, but also live data, which have to get accessed with a minimum of latency because grid jobs require this data quickly. Furthermore, the way jobs access the data is unpredictable, they may ask for several files, possibly stored on different tapes, generating high activity on tape libraries.

We use IBM's HPSS as Hierarchical Storage Manager (HSM), whose design focus is in writing, and not really on reading performance. To improve the throughput of data restore, there are several well-known strategies, such as increasing the number of tape drives, dedicating tape resources for each experiment or keeping a copy of the stored data on disks; however, each of those solutions has significant drawbacks as cost, underutilization of the resources, infrastructure limitations. With those constraints in mind, and knowing that experiments continuously increase the required throughput for the data, we decided to find a durable solution to this issue. In this paper, we identify two factors causing significant latency on data access: tape movements in the library and reading heads positioning once a tape is mounted in the reader. For this reason we chose to focus on the design and implementation of a scheduling system in front of the HSM to optimize data access. We introduce our solution, the Tape REQuest Scheduler (TReqS) and analyze the performance increase in a real world situation.

2. Problem description

In order to understand the data access difficulties at CCIN2P3, one has to understand the global context of data management in this particular Tier1 and Tier2.

The CCIN2P3 provides data management infrastructure for various scientific communities. Most of our users are related to high energy physics, such as LHC experiments, but there is also a significant fraction of them related to other domains. While all these users need to store their

data in some way, the access patterns are very different. We provide several service levels to our users, with the same HSM software using a common infrastructure (libraries and drives), staying consistent with the policy of resource sharing promoted by the CCIN2P3.

After having described the users at the CCIN2P3, let us review the method to store and retrieve files.

All files are stored in one backend system, HPSS. This application receives the files, stores them temporarily in cache disks, and then writes (migrates) them on tapes. The way HPSS writes on tapes is not predictable as which data will be migrated and when the operation will happen, and it is kept transparent to the users.

To retrieve data, the user asks for a list of files sorted without any pertinent criterion for the HSM point of view. This order does not reflect how the files are really stored in tapes. Because HPSS retrieves files in the order they have been requested (like a FIFO queue), this kind of discrepancy makes file retrieval very inefficient when reading lots of files. In the described context, the HPSS design is a major concern to achieve the data access rates especially needed by LHC experiments.

CCIN2P3 is culturally and strongly attached to resource sharing. However, in the case of concurrent access to physical resources, it is difficult to guarantee a quality of service (QoS) to our users. Currently, all users are treated equally, however, there are users with higher requirements of throughput than others. One goal of our solution is to solve this issue by implementing a fair-share, and assuring QoS with accounting metrics.

The idea of sorting all file accesses was already implemented and used successfully at Brookhaven National Laboratory (BNL) and Oak Ridge National Laboratory. In order to validate this concept in the particular environment of the CCIN2P3, we deployed the BNL implementation, called Eradat (formerly known as BNL Batch Scheduler.) After a testing period in a limited environment, the BNL implementation proved to be very efficient. Nonetheless, extensive modifications had to be made in order to fit it in our architecture, because this solution lacked flexibility, and it was tightly designed for the BNL infrastructure. Additionally, this middleware did not accomplish all our requirements (fair-share, accounting, ...) For these reasons, we concluded that in order to have a clean and global scoped scheduler for HPSS, the best solution would be to code a brand new scheduler taking the already existing key-factors from the previous systems, such as:

- Client-Server architecture
- Client independent
- Transparent to the user

And assuring that our next ideas are included:

- Easily adaptable to other HSM APIs or newer APIs from HPSS
- Enabling QoS in data access
- Simply monitorable and accountable
- Modular
- Clean code (i.e. documented and reusable), with modern design and coding methods (object oriented, patterns, models)
- Open source (under CeCILL)
- Easily maintainable

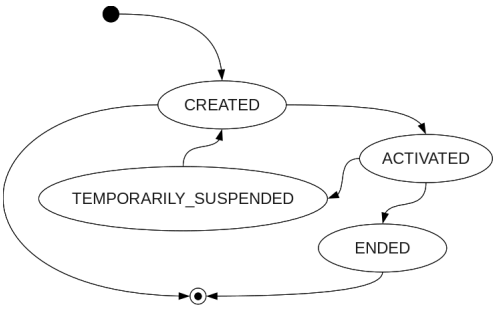

Figure 1. Queue states diagram.

3. Technical considerations

Definitions

Let us have a look at the design and the algorithm. The TReqS server acts as a proxy for HPSS, and ideally, all HPSS read requests should go through this server.

The first concept presented here is a 'file request' that represents one demand to access a file.

TReqS manages sorted lists of file requests called *queues*. As HPSS is aware of the physical position of a file on a tape, TReqS is able to keep those lists sorted by positions. A queue has the following properties:

- Each queue can only be in one of these states at a given moment:

  Created : The queue is waiting for activation.
  Activated : The files of the queue are being submitted to the HSM.
  Temporarily Suspended : The queue was in activated state but the files submission to the HSM was stopped.
  Ended : All files from the queue have been processed.

  Figure 1 shows the states map and the transitions between them.
- There can be only one queue per tape in "created", "activated" and "temporarily suspended" state.
- A queue has a position marker indicating the next file request to submit to HPSS.
- A queue has an owner who is the user having the most file requests registered on this 'queue'.

Inside a queue, file requests are sorted by physical position on the tape as provided by HPSS.

When a queue is activated, file requests are submitted to HPSS sequentially. In order to force the staging system, TReqS provides a constant number of requests to HPSS ahead of the current active request. It guarantees that HPSS has always the next file request in its FIFO list. This reduces the seek time because files are found one after the other, or just by forwarding, but not by rewinding. This strategy helps to achieve easily the maximal drive read speed when the movement is in just one direction, provided that the drive is storing data in one way only. This is not true for modern serpentines tapes. We will discuss this later.

After having read a file, HPSS waits some time for more requests in the same tape to come, then rewinds the tape. To prvent this behaviour and to efficiently force the staging order, we have to make sure that HPSS allways knows about the next file to stage. Therefore, TReqS makes sure that HPSS is allways aware of the few next requests to process. In our production context, we set this quantity to a maximum of 3 to ensure the continuous reading in HPSS. As soon as a request is completed by HPSS, TReqS updates the position marker in the corresponding queue and submits the next file request.

File request submission workflow
When a new file is submitted, TReqS acts as follow. First, it gets all useful meta-data from HPSS through the API:

- File size.
- Name of the tape storing the file.
- Physical file position on the tape.

Second, it tries to register the file in a queue by applying the following rules:

if there is an existing queue for the tape in "Activated" state **and** the current position marker is inferior to the file position **then**
  add the file request in this queue, keeping the queue sorted by physical position.
else if there is a queue in "Temporarily Suspended" state for the tape **then**
  add the file request in this queue, keeping the queue sorted by physical position.
else if there is an existing queue for the tape in "Created" state **then**
  add the file request in this queue, keeping the queue sorted by physical position.
else
  create a new queue for the tape, registering the given file request in it.
end if

As can be seen, if there is an "Activated" queue for a tape, and the requested file is located before the current position marker, this file request will be registered in a queue in "Created" state. By using a "Created" one, the current staging flow will continue forward, and when the "Activated" queue is done, TReqS will choose the "Created" queue to activate.

Applying these simple rules ensures that data reading will be the most efficient and that a drive will never rewind a tape unnecessarily. The rewind would only occur when TReqS activates a "Created" queue on the same tape. Even in this case, we keep an optimal staging workflow. In other words, during the processing of a queue, the next file's position to be read is always superior (forward) to the current position.

Best Queue Selection
The second job of TReqS is to choose the best queue to activate. Here again, a simple algorithm is determining TReqS's behavior.

The queue activation process is triggered on a regular basis as long as there are waiting queues and free resources. The algorithm considers the available resources and the user owning a queue to choose one and activate it. The resources for each media type are the number of drives per drive technology. Nevertheless, due to limitations in HPSS API, there is no logical link between resources in TReqS and the drives used in HPSS. In other words, TReqS does not know about the number of drives in HPSS or their statuses. It is the responsibility of the operator to configure the TReqS resources consistently with HPSS drives. If the resources are overbooked (more resources than actual drives), HPSS will do a lot of tape movement due to its read strategy and the inability to mount enough tapes simultaneously.

In a production environment, the defined resources will be less than the actual number of drives, allowing HPSS to use the unused drives for other operations such as migration, repacks, cleaning, etc.

There are two classes of users in TReqS: The registered ones and the unregistered ones. Registered users are given a share for each resource type, expressed in a percentage. Unregistered users are treated as having a null share; all these users are treated equally when selecting one.

The best queue selection is a two-step process. First, TReqS selects the best user. For each user, a score is computed as follows:

$$S_u = (A_u \times R_a) - R_u$$

Where S_u is the score for a user, A_u is the allocation (share) of the user, R_a is the allocated resources for a type of drive, R_u is the resources for a type of drive used by this user.

This algorithm is very important, because it reflects the user's selection policy, giving priority to users depending on the amount of resource allocated to them and the activity they already generate.

If a user is consuming a lot of resources, its score will be decreased and at some point even be less than zero. It means that it consumes more resources than allocated by the operator. This is fine as long as there is no concurrent activity. As the over-consuming user gets a bad score, the next requests by another user will have better chances of being activated, even if it has a null share.

There is a special case when the allocation is set to a negative number for a user; TReqS will simply skip the user in this step. This is convenient to lock users temporarily.

The aim of this process is to ensure to each registered user a minimum of resources. In other words, TReqS ensures that a user will have access to a minimum number of drives when there is high concurrent demand of drives. This is especially convenient in a shared resource environment like the one we have at CCIN2P3.

The second step is to select the best queue owned by a user. In our implementation, the best queue is the one containing the largest number of files of that particular user.

As mentioned above, due to limitations in HPSS API, TReqS cannot be aware of a tape being already in a drive. Otherwise, this would be a first criterion for selecting a queue.

Even though the design of TReqS facilitates the implementation of several queue selection algorithms, there has not been any other algorithm tested yet. The one implemented at this time is simple and efficient and does the job, so that there has not been any further investigation on selection algorithm at this time.

4. Statistical analysis

A good way to measure the impact of TReqS on the global activity is to look at the number of tape mounts. We collected this data from the past two years. This period gives us a reasonable timeframe to draw some conclusions. Although the monthly activity in data reading is not comparable month by month over the two years, the global activity by year is similar in 2009 and 2010.

In order to analyze the data, we will present the history of tape scheduling at the CCIN2P3, and how it impacted the system. The history has three steps:

- Manual prestaging.
- BNL batch adaptation.
- TReqS.

The manual prestagings were done for specific reprocessing campaigns, especially for Atlas. We processed a list of files to be staged, and we sorted the file requests by tape and physical position on the tape. We manually asked HPSS to stage the list produced. This was done taking into account the drive availability: Not too many to generate "drive waits", nor too few. This process used to take a lot of time because it was completely manual. Furthermore, we had to adapt the staging process to the overall activity in HPSS.

The staging time was good because the files were read in the best order, but at the expense of a lot of manual work and monitoring.

After this first try, there was the BNL batch system period. As stated before, we adapted it to our environment and pushed it in production just for the dCache front-end and limited to specific experiments. This solution gave good reading times at the end of the summer 2009. As can be seen in Figure 2, in September, the mounts represented in blue, were lower that the precedent months, passing from more than 3000 in July, to less than 1800 in September.

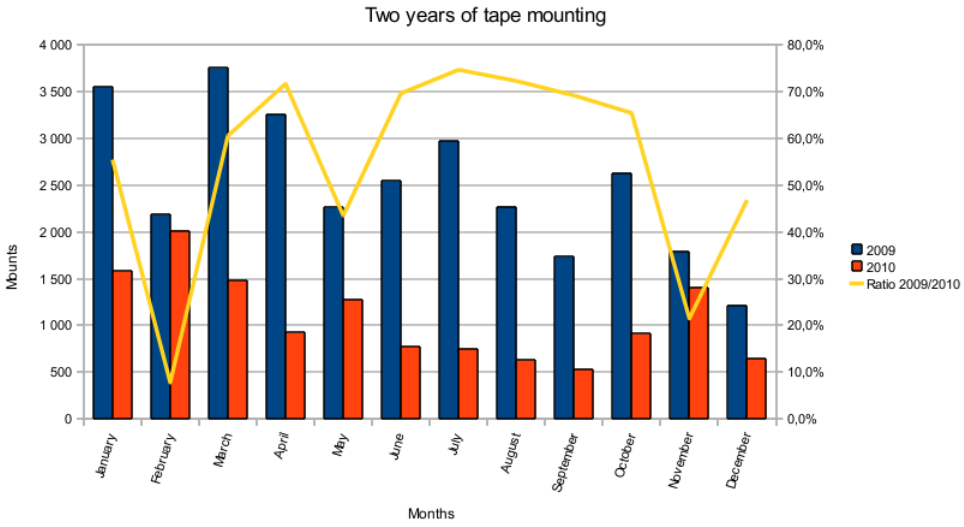

Figure 2. Evolution of the mount operations per months in 2009 and 2010

At the end of 2009 we put TReqS in production. First, we started with just some front-ends and some clients, as we did with BNL Batch, and then progressively everywhere. Figure 2 also shows that since September 2009, we never have more than 2000 mounts per month.

The complete migration to the TReqS scheduling system, including BNL Batch operations, took around 6 months at the CCIN2P3, from August 2009 to mid March 2010. During this time, we can clearly see that the mounts were progressively reduced as we deployed TReqS to other front-ends. After this period, from March 2010, the quantity of mounts per month is almost constant, as shown in Figure 2. The plotted graphs are from the accounting system of the tape library (robot), and the mounts include some external activities such as tape importing, or TSM. Nevertheless, these activities do not require more than a hundred of mounts per month.

5. Conclusion

The deployment of TReqS and its usage in the past years has shown that it has achieved our expectations: Remarkable performance improvements, better control on production, enabling quality of service. There is also a positive side effect on the efficiency in the computing farm, as jobs spend less time idle waiting for input data.

Optimizations can be done in the future by adapting the algorithm to serpentine tapes. But at this time, HPSS does not provide enough metadata through the API for TReqS to be aware of the real physical position of a file on such tapes.

Thanks

- David Yu from BNL, for his kind assistance and interest the project
- The storage team at CCIN2P3 and Suzanne Poulat for the statistics gathering

Learn more!

- The TReqS project page: https://forge.in2p3.fr/projects/show/TReqS
- ERADAT http://drupal.star.bnl.gov/STAR/presentations/acat-2010/jerome-lauret-0
- HPSS Performance study: http://drupal.star.bnl.gov/STAR/comp/sofi/hpss/hpss-performance-study