TASK QUARTERLY 8 No 4, 537–548

PERFORMANCE ASPECTS OF DATA DELIVERY USING HIERARCHICAL STORAGE MANAGEMENT SYSTEMS IN THE GRID

DARIN NIKOLOW[1], RENATA SŁOTA[1]
AND JACEK KITOWSKI[1,2]

[1] *Institute of Computer Science, AGH University of Science and Technology, Al. Mickiewicza 30, 30-059 Cracow, Poland*
{*darin, rena, kito*}*@uci.agh.edu.pl*

[2] *Academic Computer Center CYFRONET, Nowojki 11, 30-950 Cracow, Poland*

(Received 6 June 2004; revised manuscript received 29 July 2004)

Abstract: Grid computing has recently gained in popularity. Grid applications can be very demanding of the data storage facilities in the Grid. The existing data grid services are often insufficient and additional optimization of the data access is necessary. This paper presents research concerning the optimization aspects of data management for Hierarchical Storage Management (HSM) systems in the Grid.

Keywords: HSM, Grid, access time estimation

1. Introduction

Grid applications can be divided into two main categories: computationally intensive applications and data intensive application. The first category has been recognized as the main topic of interest of the early grid research [1]. With the growing demand for grid applications concerning data storage and access to the data distributed on the grid, particular methods and services are needed to meet the requirements of data intensive grid applications [2]. The specific requirements and characteristics of this kind of data are the following: the amount of data is huge (tera- and peta-bytes), access to the data has to be secure and fast in terms of latency and bandwidth, and data have to be protected against loss. In order to meet these requirements the following methods are applied in the Grid middleware layers: data encryption, parallel data transfer, and data replication.

The security of access to data is also an issue in the traditional computational grids and has evolved into GSI (Grid Security Infrastructure) [3], based on public key encryption algorithms. GSI allows authorized access to the distributed Grid resources

with a single login. GridFTP addresses the fast file transfer problem by allowing parallel data transfer for better utilization of the existing network bandwidth [4].

Data replication is the main method of assuring good level of data protection. Examples of such usage are RAID systems. In data grids, data are replicated for two main reasons: data protection and reduction of data access time. In the case of replicated data sets the problem of replica selection arises [5]. The application or rather the underlying grid middleware, has to decide which replica to choose so that the application obtains its data as fast as possible. Another problem concerning replication is automatic generation of new replicas and eventually deleting the unused replicas [6, 7]. The decision about whether, where and when to create a new replica is not trivial. Many parameters have to be taken into account, including frequency of use, locations where the file has been requested from, load of the storage systems, the available and future network bandwidths. Some of these parameters are static but others are dynamic, which makes the solution even more complex. Some prediction methods about the future resource utilization are necessary to properly create new replicas. Yet another problem is keeping the data replicas consistent with the original data. The replicas have to be updated in a non-disruptive and efficient way. Some grid-related projects [8] assume that the data have a WORM (Write Once Read Many) access pattern, which simplifies the process of keeping the data consistent.

In order to select the best replica (*i.e.* a replica which can be accessed the fastest) information about the access time for a particular data set is needed [9]. The access time depends mostly on the network's bandwidth, but other factors matter as well,such as system load and performance characteristic of the storage devices. This information can be obtained from the Grid resources monitor services, part of almost every grid project.

Because of the huge volume of data, TSS (Tertiary Storage Systems) are often used to economically store this amount of data. Tapes are the usual choice of media. Tertiary storage facilities (tape libraries, optical jukeboxes) are managed by HSM (Hierarchical Storage Management) software. An HSM system uses various media with different storage characteristics, organized in a hierarchy. The fast access devices such as hard disks (or secondary storage) are used for the most wanted data, acting in the same way as cache memory in computer systems. Data which has not been accessed for a long period of time is kept in slower tertiary storage like tapes or magneto-optical disks. The storage hierarchy in HSM systems introduces another problem: the access time depends on the localization of the data. The time can vary greatly – from mili-seconds to tens of minutes. This is why special care should be taken when replicated data is stored in HSM systems and when best replica selection has to be done. Adequate estimation of access time for data stored in HSM systems is of great importance [10].

Data kept in an HSM system is available when it resides in the disk cache. If the data is not there it has to be staged first. If the file is large, this could take a long period of time. The estimation system has to be aware of the peculiarities of the underlying storage system in order to estimate access time properly.

The following sections give a more detailed description of the research carried out by the authors concerning data management for HSM systems in the Grid. The

HSM systems discussed in this paper are storage systems each located at a particular site, sharing their resources within the Grid. Nevertheless, the HSM system itself can be locally distributed running on a few servers and managing one or more automated media libraries. The main areas of the research have been access time estimation for HSM systems (Section 2) and optimization of access to data in tertiary storage for Grid environments (Section 3). Conclusions are presented in the final section.

2. Access time estimation for HSM systems

Access time, t_{access}, is considered in this study as the sum of start-up latency time, t_{latency}, and transfer time, t_{transfer}:

$$t_{\text{access}} = t_{\text{latency}} + t_{\text{transfer}}. \tag{1}$$

The HSM system shares its resources in the Grid environment via a network. The following equation presents the latency time which a client[1] will experience:

$$t_{\text{latency}} = t_{\text{latency\_HSM}} + t_{\text{latency\_network}}. \tag{2}$$

In the above equation $t_{\text{latency\_HSM}}$ is the latency time introduced by HSM itself and $t_{\text{latency\_network}}$ is the latency time introduced by the network. The next equation expresses the transfer time measured on the client side:

$$t_{\text{transfer}} = \frac{S_{\text{file}}}{\min(B_{\text{network}}, B_{\text{HSM}})}, \tag{3}$$

where S_{file} is the file size, B_{network} is the available network bandwidth, and B_{HSM} is the available bandwidth of the HSM system itself.

With the incredible progress made in the field of network technologies (availability of 10 Gb networks) the network is no longer assumed to be the constant bottleneck of the system. Instead, the storage devices like hard disks and removable media drives can become the bottleneck. That is why research on the data access time estimation is necessary.

In our study we concentrate on the access time of a "stand-alone" HSM system, which means that the eventual overhead imposed by the network is not taken into account.

Two approaches to estimating the access time for an HSM system have been considered:

- a open HSM approach, in which the source code of the HSM system is available, so that event reporting functions can be introduced, and
- a gray-box HSM approach, in which the essential HSM system information is accessible via the system's native tools only.

The next subsections describe in more detail the gray-box approach to data access time estimation for HSM systems and our experimental results.

2.1. The gray-box approach to HSM access time estimation

The system can be considered as gray-box if at least partial knowledge of its behavior is available and the information about the state of the system can be obtained

1. In this case a client can be a Grid application or a Grid middleware component such as a replica manager.

by using the native utilities provided with the system. This is the case with commercial HSM systems where access to the source code and knowledge needed to change it are not available. Knowledge about the system can be obtained by studying the system's documentation and appropriate testing.

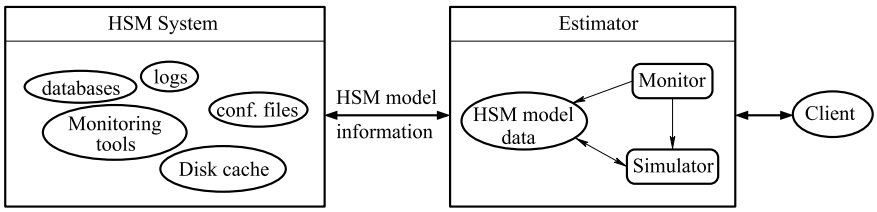

Figure 1. Data access time estimation for an HSM system

The proposed HSM estimation system consists of two main modules: a Monitor and a Simulator (Figure 1). The Monitor obtains essential information about the HSM system and stores it in the model data structures (HSM model data). The Simulator estimates the access times for the requested files using event-driven simulation. The Simulator is supposed to be independent of the particular HSM system, while the Monitor is a system-dependent module, which strongly relies on the HSM software for gathering the essential data.

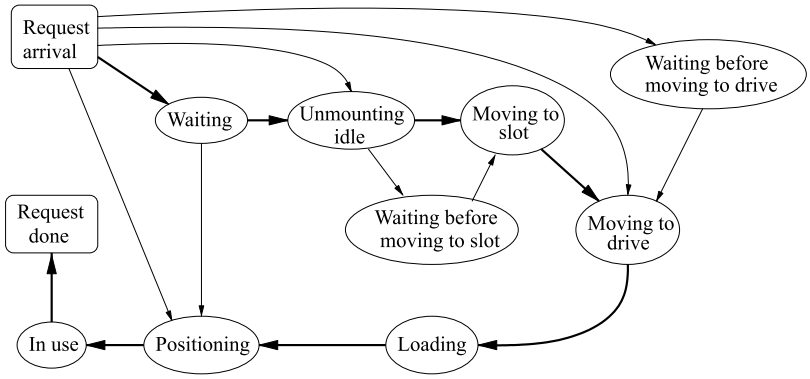

Figure 2. State transition diagram of a request processed by the TSS

Processing a request by the HSM system goes through its subsequent states, triggered by the events shown in Figure 2. The most probable path is shown by thicker lines. Table 1 briefly describes each state. The simulation algorithm for HSM systems is based on this state transition diagram. The diversity of HSM systems appears in the Simulator at two levels: the performance characteristics of the devices and the behaviour of the HSM software. The performance parameters are specified in a configuration file. However, the behavior of the HSM software impinges directly on the simulator algorithm, which has to be designed to simulate various typical behaviors. These typical behaviors concern the policy of request queue serving and the policy of idle tape[2] handling.

2. An idle tape is a tape which has just been used and is still in the drive but not actually used at the moment.

Table 1. States in serving a request by a HSM system

State	Description
Waiting	Waiting for resources (robot arm, drive or tape)
Unmounting idle	Unmounting of an idle tape
Waiting before moving to slot	This state will be visited if the robot arm is busy serving another request
Waiting before moving to drive	This state will be visited if the robot arm is busy serving another request
Moving to slot	Moving an idle tape from drive to slot
Moving to drive	Moving the required tape from slot to drive
Loading	Bringing the tape on-line
Positioning	Positioning the tape to the first block of the file being accessed
In use	Transferring data from the tape

Two different queue serving policies are possible: strict FIFO, where requests are served according to their time of issue, and priority FIFO, where a request posted later can be served earlier for better overall performance optimization, for instance when the tape needed is already mounted.

Three different policies of handling an idle tape are possible: immediate eject, non-forceable lazy eject and forceable lazy eject. With the immediate eject policy the tape gets unmounted as soon as it becomes idle. With the non-forceable lazy eject policy the tape is unmounted only after it has been idle for a certain period of time, $t_{\text{lazy\_eject}}$. With the forceable lazy eject policy the tape remains in the drive for at most a $t_{\text{lazy\_eject}}$ time interval and for at least an aggressive eject, $t_{\text{aggressive\_eject}}$, time interval. If a request needs the tape to be unmounted, it can become unmounted before the $t_{\text{lazy\_eject}}$ time has elapsed, but not before the $t_{\text{aggressive\_eject}}$ time has elapsed.

The Monitor is running all the time and is constantly monitoring the HSM system. In this way the data in the model is kept up to date as much as possible. Since the monitoring of an HSM system depends strongly on vendor-provided utilities, the Monitor is HSM system-dependent and has to be ported for a particular system. For example, in order to gather the necessary data the Monitor can fork HSM-specific tools, scan logs (having different formats), parse the configuration files, check the disk cache and database files.

The accuracy of estimation depends on many factors, so it is not easy to obtain estimation times with small errors. The factors are as follows: the exact HSM algorithm is not known or is too complicated, the positioning and transfer times of the storage devices vary widely even for the same request sequence, and non-fatal read errors occur which increase access time. The following section presents our experimental results concerning the accuracy of estimation.

2.2. Experimental results

The experiments described below were conducted at the CYFRONET-AGH Academic Computer Center in Cracow. This site is equipped with two tape libraries with DLT7000 tape drives and a magneto-optical jukebox.

The tertiary storage hardware is managed by the DiskXtender HSM software. The system runs on two HP9000 servers.

Two types of tests have been made: single request tests and multiple request tests. The latter case is more realistic for standard HSM operations. For a more detailed description of the experiments, see our paper [11].

In these tests estimated t_{access} (Equation (1)) were compared with measured (real) t_{access} for given requests and errors were calculated. In Table 2 the single request tests results are presented for files residing on tapes. We can see that for some cases the relative error is significant. The average relative error is 19%, while the average absolute error is 50s.

Table 2. Single request test results for tapes

No	Measured t_{access} [s]	Estimated t_{access} [s]	Absolute error [s]	Relative error [%]
0	180	264	84	46
1	365	203	162	44
2	253	234	19	8
3	181	203	22	12
4	283	279	4	1
5	256	204	52	20
6	286	203	83	29
7	237	238	1	0
8	278	254	24	9

Table 3. Multiple request test results for tapes

No	Measured t_{access} [s]	Estimated t_{access} [s]	Absolute error [s]	Relative error [%]
0	202	180	22	11
1	207	180	27	13
2	430	310	120	28
3	524	410	114	22
4	576	360	216	38
5	541	470	71	13
6	842	550	292	35
7	800	560	240	30
8	909	670	239	26
9	991	960	31	3

The results from the multiple request tests are shown in Table 3. In this case, relative errors are also significant for some requests. The average relative error is 21%, while the average absolute error is 138s.

The main source of inaccuracy is the problem of modelling the positioning and transfer time for the tape drives, especially for compressed data. If data are well-compressed the transfer rates are higher by a factor of two compared with non-

compressible data. The serpentine recording of the most modern drives results in another source of complication.

3. Data access optimization for data kept in TSS

Tertiary storage devices, especially tapes, have high access times. This is due to the sequential nature of tape drives, in which the positioning of the head takes much longer than in direct-access devices. The transfer rates are also lower, but here the difference is less significant. Two methods have been used in our work in order to make the access to tape-resident data more efficient: direct fragment access and subfiling. These methods are described in detail in the following subsections.

3.1. Fragment access

One of the possible optimizations of data access for HSM systems is direct fragment access (DFA). By this method we can access a file fragment without staging the whole file. Since only the necessary fragment is staged, less space is used in the disk cache and the tape-to-disk transfer time is shorter. DFA has been implemented in MMSRS (Multimedia Storage and Retrieval System) [12] and VTSS (Video Tertiary Storage System) [13] enabling efficient access to video sequences.

3.1.1. MMSRS

MMSRS is a system for efficient access to video fragments, which was essential in the development of a medical video database system in the PARMED project [14].

The architecture of MMSRS based on HSM software is shown in Figure 3. The system consists of an Automated Media Library (AML) and the following software components: an AML managing system (DiskXtender, comercial HSM software), an MPEG Extension for HSM (MEH) and a public-domain WWW server. The MEH consists of an MPEG Store Application (MSA) and an MPEG Retrieve Application (MRA). The MMSRS is implemented as a middleware layer between the client application and the HSM system based on the Legato commercial DiskXtender software.

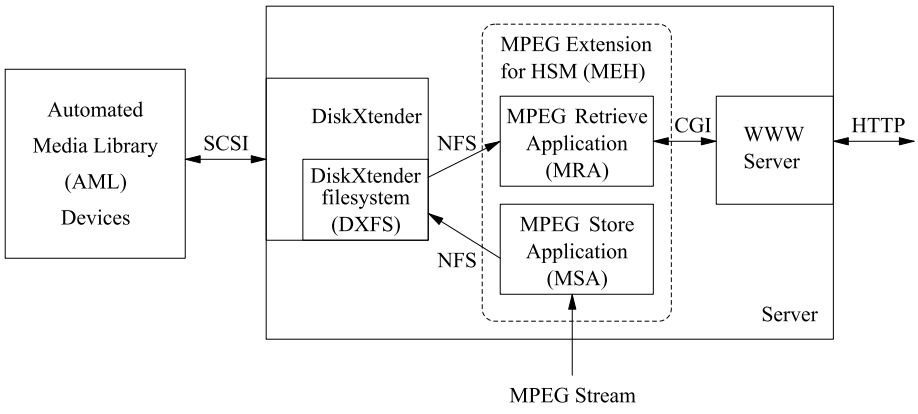

Figure 3. The architecture of MMSRS (Multimedia Storage and Retrieval System)

Because of the features imposed by the DiskXtender software and our requirements for the MMSRS it was necessary to carefully design the way in which videos are written to the DiskXtender filesystem. If videos were stored in whole as usual

files, a user requesting a file that existed only in the tertiary storage would experience start-up delay, as the whole file must first be staged. When the file is large the delay may be unacceptable.

In order to apply the DFA technique we decided to cut the videos into pieces of similar size and store them as different files. The cutting substantially reduces the start-up delay. Each piece starts with an I-frame. We called the application splitting and storing the videos into DXFS an MPEG Store Application (MSA). Due to the chosen method of storing the videos, another application for retrieving them is needed. We called it an MPEG Retrieve Application (MRA). MRA is executed whenever there are client requests waiting to be served, while MSA is used by the administrator when a new video is to be stored.

In order to keep the MPEG stream as smooth as possible (when the pieces reside on tapes), a simple prefetching technique is used. It is based on staging the next file to be transmitted while transferring the current one.

3.1.2. VTSS

VTSS is the second system in which we have applied the DFA technique. It has been designed from scratch and does not rely on any other tertiary storage management software. VTSS is capable of accessing a fragment of a video or normal file by directly addressing the blocks on the tape where the requested fragment resides.

The architecture of VTSS is shown in Figure 4. The core of the system capable of storing arbitrary files is called a FiFra TSS (File Fragmentation TSS). FiFra TSS features file fragment access by directly accessing the file blocks from tape. The system consists of two main daemons: a Repository Daemon (REPD) and a Tertiary File Manager Daemon (TFMD). REPD controls the Automated Media Libraries (AML) and is responsible for mounting and dismounting the required tapes. TFMD manages the files *filedb* and *tapedb* (in which the information about files stored on VTSS and the tape usage is kept) and transfers files from the removable media devices to a Client.

In order to have fragment access to video files (a video fragment is specified in frame units), two interface modules were added to the FiFra TSS.

The first module, called VSI (Video Storage Interface), is responsible for storing video files in the VTSS. It stripes the header out of the MPEG file and stores it on the hard disk. The rest of the MPEG file (without the header) is then stored on a tape and the index file for that video is stored on a disk. A record describing the video is put into the `namedb` file. The index file contains the time stamp and offset for each GOP (Group Of Pictures) of the MPEG file. VSI has to be run on the host where the `namedb` file resides and itself starts two external programs: `splitter`, which stripes the MPEG file and produces the index and `clnnew`, which stores the MPEG body into the FiFra TSS.

The second interface module, called VRI (Video Retrieval Interface), is responsible for retrieving fragments of videos. It constructs the requested video fragment by concatenating the header of the video with the appropriate file fragment of the body of the MPEG file. The file fragment range is obtained by searching the index file of that video for start offset and end offset matching the requested time range.

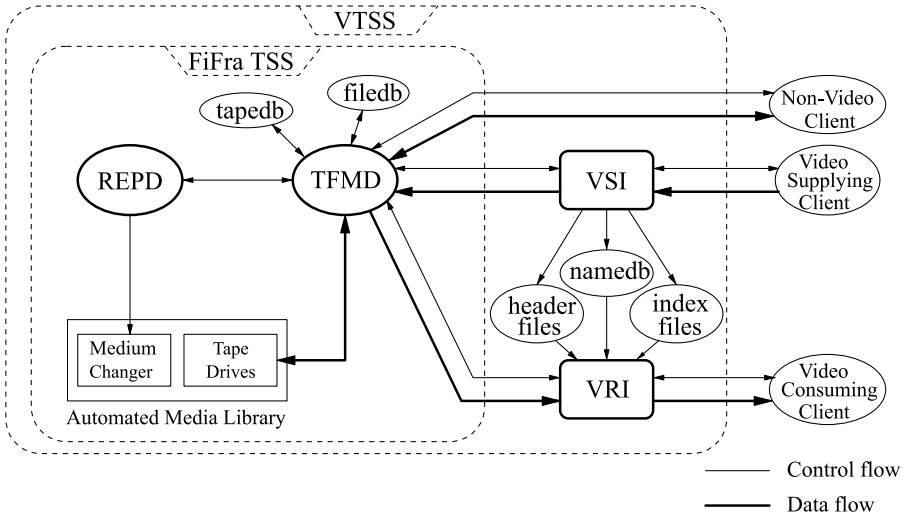

Figure 4. the architecture of the Video Tertiary Storage System

3.2. Fast access to large files on tapes

When a file residing on tapes is being accessed, it is usually first copied to the disk cache and only after that it is available for reading. If the file is large the transfer from tape to disk can take a long time. During that time the file is inaccessible, even the part that is already on disk and could be accessed. In order to accelerate access to large files the subfiling technique can be used. The technique is based on splitting a large file into smaller ones (subfiles), which can be accessed faster. From the user's point of view the file is still large and in one piece, but internally it is represented by a set of smaller physical files.

This technique has been used in MMSRS to make the access to fragments of video files more efficient. At the same time, VTSS can access fragments by directly accessing the tape blocks containing the required fragment. Additionally, VTSS allows access to the portion of the file that has already been staged. However, VTSS is a prototype system used mainly for testing the performance of such approach and is far from a full production system. When a general purpose HSM system with no direct file fragment access feature is used, middleware is needed to transparently split the files on writing and concatenate on reading.

3.2.1. Implementation of the subfiling technique for HSM systems in the Grid

The subfiling technique can be implemented for the Grid environment in many ways. For the case when there is a dedicated Data Archiving System (DAS) in the grid [15] it is implemented as an FTP server with fragment access capabilities. For the Crossgrid project [16], where the data management system relies on the Globus toolkit [17], it is implemented as a wrapper to the existing GridFTP server. In both cases the application can benefit from this implementation by making faster its access to large data files residing on tapes. The implementation of the FTP server for the DAS is described below as an example.

The code of a normal FTP server has been appropriately modified to transparently split and concatenate the requested files. The server can access segmented files

(which are internally stored as a set of subfiles) and non-segmented, normal files. The client (which can be a user or an application) can decide if the file to be written has to be segmented or not. Since no changes in the FTP protocol are allowed, the following method to choose between the segmented mode and the non-segmented mode has been proposed: if the file is stored under certain directory path (specified in the config files) it has to be segmented, otherwise it is a normal file. For example, if we have specified the directory path as `SegmentedFiles/` and if we issue the ftp command `put myBigFile SegmentedFiles/myBigFile`, then this file will be segmented.

Segmented files are written as follows:

- a directory having the same name as the file to be written is created;
- the subfiles of the file being written are put into that directory.

3.3. Experimental results

The goal of these experiments has been to compare the performance for large file transfers for three different systems: DiskXtender, MMSRS and VTSS (see Subsections 3.1.1 and 3.1.2).

The experimental results for whole video file transfer are shown in Table 4. The parameters presented in the table are local, which means that the network overhead is neglected. Consequently, the start-up latency corresponds to the latency of the HSM itself, $t_{\text{latency}} = t_{\text{latency\_HSM}}$ (see Equation (2)) and the transfer time is the time necessary to send the data assuming that the network has a greater bandwidth than the bandwidth of the system, $t_{\text{transfer}} = \frac{S_{\text{file}}}{B_{\text{HSM}}}$ (see Equation (3)). The average rate, r_{avg}, and the total throughput, th_{total}, are defined by:

$$r_{\text{avg}} = \frac{S_{\text{file}}}{t_{\text{transfer}}}, \tag{4}$$

$$th_{\text{total}} = \frac{S_{\text{file}}}{t_{\text{access}}}, \tag{5}$$

where $S_{\text{file}} = 790\,\text{MB}$.

Table 4. System performance for whole video file transfer

	DiskXtender		MMSRS		VTSS	
	DLT2000	DLT7000	DLT2000	DLT7000	DLT2000	DLT7000
t_{latency} [s]	718	610	90	110	70	55
t_{transfer} [s]	135	164	710	615	677	160
t_{access} [s]	853	774	800	775	747	215
r_{avg} [MB/s]	5.85	4.82	1.11	1.28	1.17	4.94
th_{total} [MB/s]	0.93	1.02	0.99	1.02	1.06	3.67

We can see that VTSS outperforms the other two systems, especially when the start-up latency is considered. DiskXtender stages the file into the disk cache before actually transfering it, which manifests itself in low transfer times and high start-up latencies. That is why DiskXtender has the highest average rate. Nevertheless, the total throughput (in which latency is taken into account) is much lower (0.93 MB/s). In spite of the fact that MMSRS relies on DiskXtender for managing the tertiary storage hardware, it was possible to significantly decrease start-up latency by using

the subfiling technique. DiskXtender and MMSRS performance with DLT7000 drives is only slightly better than that with the DLT2000 drives, despite the transfer rate of the DLT7000 tape drive being four times higher than that of DLT2000[3]. VTSS has achieved average transfer rates of 1.17MB/s and 4.94MB/s (for DLT2000 and DLT7000, respectively), which are close to the hardware-imposed limit, thanks to its better utilization of the tape drives.

4. Conclusions

In this paper we have presented our latest research concerning data management for HSM systems in the grid environments. Optimization of access to data distributed in the Grid is not a trivial task. Many aspects have to be considered to solve this problem like replication methods, data access time estimation techniques, and grid monitoring. An access time estimation system based on the gray-box simulation approach has been described and experimental results for the DiskXtender implementation have been presented. The estimation accuracy is sufficient for most cases of best replica selection. We have also presented an implementation of the subfiling technique to speed up access to large files residing on tapes in HSM systems. The experiments have shown a significant drop in the latency time for this technique.

The developed systems are also useful for the development of strategies for automatic data replication, now in progress.

Acknowledgements

The work described in this paper was supported in part by the European Union through the IST-2001-32243 "CrossGrid" project and by the Polish Committee for Scientific Research (KBN), grant no. 4 T11C 028 24. AGH-UST grant is also acknowledged.

References

[1] Foster I and Kesselman C 1999 *The Grid: Blueprint for a New Computing Infrastructure*, Morgan-Kaufman

[2] Chervenak A, Foster I, Kesselman C, Salisbury C and Tuecke S 2001 *J. Network and Computer Applications* **23** 187

[3] Foster I, Kesselman C, Tsudik G and Tuecke S 1998 *Proc. 5th ACM Conf. on Computer and Communications Security*, San Francisco, California, USA, pp. 83–92

[4] Allcock W (Ed.) 2003 *GridFTP Protocol Specification (Global Grid Forum Recommendation GFD.20)*, http://www.globus.org/research/papers/GFD-R.0201.pdf

[5] Vazhkudai S, Tuecke S and Foster I 2001 *Proc. of the IEEE Int. Conf. on Cluster Computing and the Grid (CCGRID 2001)*, Brisbane, Australia, pp. 106–113

[6] Bell W, Cameron D, Capozza L, Millar P, Stockinger K and Zini F 2002 *Proc. 3rd Int. Workshop, Grid Computing GRID*, Baltimore, USA, LNCS **2536** 46

[7] Lamehamedi H, Shentu Z, Szymański B and Deelman E 2003 *Proc. 12th Heterogeneous Computing Workshop (HCW2003)*, Nice, France, IEEE Computer Science Press, p. 100b

[8] *DataGrid – Research and Technological Development for an International Data Grid*, EU Project IST-2000-25182

[9] Stockinger K, Stockinger H, Dutka L, Słota R, Nikolow D and Kitowski J 2003 *Proc. 4th Int. Workshop on Grid Computing (Grid2003)*, Phoenix Arizona, IEEE Computer Society Press, pp. 149–157

3. DLT2000 transfer rate= 1.25MB/s, DLT7000 transfer rate= 5MB/s

[10] Nikolow D, Słota R and Kitowski J 2004 *Proc. 5th Int. Conf. Parallel Processing and Applied Mathematics*, Czestochowa, Poland, LNCS **3019** 182
[11] Nikolow D, Słota R and Kitowski J 2002 *Proc. of the Cracow Grid Workshop*, Cracow, Poland, pp. 209–216
[12] Słota R, Kosch H, Nikolow D, Pogoda M, Breidler K and Podlipnig S 2000 *Proc. Int. Conf. High Performance Computing and Networking*, Amsterdam, The Netherlands, LNCS **1823** 517
[13] Nikolow D, Słota R, Kitowski J, Nyczyk P and Otfinowski J 2001 *Lecture Notes in Computer Science* (LNCS) **2110** 435
[14] Kosch H, Słota R, Boszormenyi L, Kitowski J, Otfinowski J and Wojcik P 2000 *Proc. Int. Conf. High Performance Computing and Networking*, Amsterdam, The Netherlands, LNCS **1823** 543
[15] SGIgrid: Large-scale Computing and Visualization for Virtual Laboratory Using SGI Cluster (in Polish), KBN Project, http://www.wcss.wroc.pl/pb/sgigrid/
[16] *CROSSGRID – Development of Grid Environment for Interactive Applications*, EU Project IST-2001-32243
[17] The Globus Project, http://www.globus.org/