International Journal of Enhanced Research in Management & Computer Applications, ISSN: 2319-7471
Vol. 2 Issue 9, Nov.-Dec., 2013, pp: (30-35), Available online at: www.erpublications.com

Reinforcement Learning Approach for Data Migration in Hierarchical Storage Systems

T.G. Lakshmi, R.R. Sedamkar, Harshali Patil
Department of Computer Engineering, Thakur College of Engineering and Technology,
Kandivali, Mumbai, India

Abstract: There are several shortcomings in the existing data migration techniques in Hierarchical Storage Systems (HSS). The first and the most important among them is that data migration policies are user defined - hence static and reactive. Secondly, data migration at the host side is not yet completely explored. The other major drawbacks are that each storage tier is modelled as an agent; the data migration methodology is I/O triggered and the tier cost represented as a complex fuzzy rule base (FRB). This paper proposes a simple and single data migration agent in the HSS. The data migration agent will be a standalone daemon which implements the Reinforcement Learning (RL) algorithm. The agent will formulate and tune policies based on which the data migration will take place. The proposed model in this paper aims to achieve comparable results with existing systems in data migration, input/output queue length and response time of storage tiers.

Keywords: HSS, Data Migration, Reinforcement Learning.

1. INTRODUCTION

Businesses are having a very difficult time in maintaining the data center infrastructure. With the rate of generation of data growing annually by 55 %, the cost of (1) acquisition of new disk arrays, (2) backup, (3) floor space, (4) electricity, is growing exponentially. Data Storage Management is a difficult problem as data should be placed on resources which are able to fulfill user's request. It is also necessary to optimize the usage of resources and minimize the cost of their usage [1]. Traditional solution is to buy more disk space, or to store the data on external media and retrieve it as required [2]. Hierarchical Storage Management (HSM) provides an automatic way of managing and distributing data between the different storage layers to meet user needs for accessing data while minimizing the overall cost [3]. Hierarchical Storage Management (HSM) is used to effectively utilize the storage space according to its capability and also meet customer demands [3]. The data placement in HSM is done to minimize the cost of data storage while maximizing accessibility. In HSM the storage devices are separated into tiers based on their capacity and speed i.e. TIER –I consists of disks that have a very high I/O speed and are hence expensive. The frequently accessed data is placed on higher tiers and the least frequently accessed data on lower tiers. Thus in essence data lives where it is more efficient. This gives rise to significant savings and revenue opportunities to the business.

As we see in Fig 1, the fastest storage components are the most expensive. To maximize the return on investment (ROI) on the computing resource investment we need to utilize the resources to their best capability. The data arrangement is not static. Based on the current system's needs the data is moved between the tiers. The challenge is to find the right placement of the data.

Figure 1: Cost and Speed of Access of different storage components

Migration is an operation in which the selected data is physically moved, or migrated, to different storage components. It is an effective technique to move data across based on its access pattern. Though data migration would essentially place the data where it is most efficient, the process of data placement itself should not be an overhead i.e. data migration should not violate the QoS of the applications.

In a SAN environment, the data migration can happen at either at the storage controller or the host. Most of the existing data migration is placed at the storage controller. VMware and EMC^2 are the two major storage vendors who have implemented data migration at the storage controller end [4]. The data migration technique when implemented at the controller end is proprietary and does not support the use of heterogeneous storage components. The key is to look for a data migration technique which is vendor neutral. The existing data migration techniques at the host end are based on heuristic policies which are static and user defined. The policies are reactive in nature and consist of IF-THEN statements which get triggered when a particular threshold is violated. E.g. IF space utilization in TIER-I is greater than 80%, THEN migrate *.tmp files greater than 1MB ordered by size to TIER-III until the space utilization of TIER-I is 60%.

To address the above shortcomings our proposal is to develop a proactive data migration method initiated at the host end. Data Migration as a process closely resembles a 'Markov Decision Process' (MDP). MDP can be solved via Dynamic Programming and Reinforcement Learning [5]. Dynamic Programming has the following pitfalls: (1) it requires knowledge of the complete model of the environment which is often not obtainable; (2) it rapidly becomes computationally intractable as the number of state variables increases. Hence in our proposal we used Reinforcement Learning algorithm to map situations to actions so as to minimize the input/output request response time of every storage tier.

The next sections are organized as follows: Section 2 introduces related work and also provides a background on the use of reinforcement learning as a solution methodology. In Section 3 and 4, we describe the research questions and details of the implementation. We end with a discussion and conclusion in Section 5and Section 6.

2. RELATED WORK

Learning most often happens by constant interaction with the environment. This interface gives a lot of information about (i) cause – effect and (ii) consequences of action. The knowledge gained, helps us in determining the actions required to achieve our goal. Most of the learning done by infants and animals is based on the above approach.

Reinforcement Learning does not need prior knowledge; it autonomously gets optimal policy with the knowledge obtained from trial and error and continuous interaction with the dynamic environment. It has characteristics of self-improvement and online learning. Reinforcement Learning (RL) is a goal- directed computational approach to learning from interaction. In RL the learner is learning how to map situations to actions so as to maximize the long term reward [5]. The learner is not told the best action for a situation but discovers the best action for a situation by trial and error [5]. The differences between RL and standard supervised learning are that in RL (i) correct input/output pairs are never presented, (ii) sub-optimal actions explicitly are not corrected and (iii) the focus on striking the right balance between exploration and exploitation.

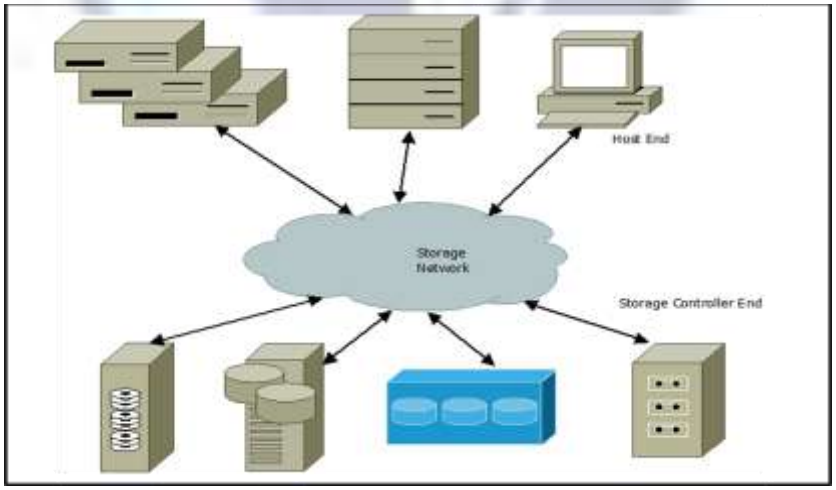

Figure 2: Storage Area Network Architecture

2.1 A RL approach to Virtual Machine Auto-configuration

VCONF [6] is a reinforcement based approach to automate the virtual machine (VM) configuration process. Virtual Machine enables multiple virtual machines to share resources on same host. Resources allocated to the VM's should be re-configured dynamically based on the workload.

Self-configuration systems automatically adapt software parameters, hardware resources for the purpose of correct function or better performance. The traditional methods such as analytical and feedback control theory are found to be unqualified to solve dynamic tuning problem. The RL algorithm was able to find near optimal configurations of VM. VCONF implements a look up table based Q – function [6]. The system configuration parameters are tuned adaptively based on the incoming workload. In the presence of workload dynamics, VCONF was able to adapt to a good configuration within 7 steps and showed 20% to 100% throughput improvement over basic RL methods [6].

2.2 A RL framework for online data migration in hierarchical storage systems

In Multi-tier storage systems, a very important performance indicator is the response time of the file I/O requests. The response time can be minimized if the frequently accessed files are located in the fastest storage tier. The challenge is that the file access patterns change over time. The methodology proposed by David Vengerov [7] presents an RL algorithm which automatically tunes the file migration policies to minimize the average request response time. File migration is the process of displacing the file storage location to other tiers based on the file's access frequency. The migration rules are formed as policies which associate every situation to an action. It employs the temporal difference (TD) learning to learn the cost of the policy for every state of the system. The file migration is I/O triggered. The RL methodology applied here is the I/O triggered migration policy. [7]

2.3 Reinforcement Learning Model, Algorithm and its Application

Reinforcement learning is learning what to do, how to map situations to actions, so as to maximize a numerical reward signal [8]. The learner discovers the best action by trial. Action taken at a situation affects the long term reward. The two characteristics, trial-and-error search and delayed reward, are the two most important distinguishing features of reinforcement learning [8].
Reinforcement Learning has evolved from animal learning. The most common example is - a dog learning to obey commands. During the learning process, whenever the dog obeys commands he is rewarded for his/her good behavior. In a RL system there are 2 main elements - agent and the environment, and four main sub- elements: policy, reward function, value function, model of the environment. Almost all reinforcement learning algorithms involve estimating value functions – functions of state(or of state-action pairs) that estimate how good it is for the agent to be in a given state or how good it is perform a given action in a given state. There are four main RL algorithms SARSA, Temporal Difference Learning, Q-learning and Function approximation [9].
Reinforcement Learning is mainly used in process control, dispatch management, robot control, game competition and information retrieval etc., the widest application is the field of intelligent control and intelligent robot [9, 10].

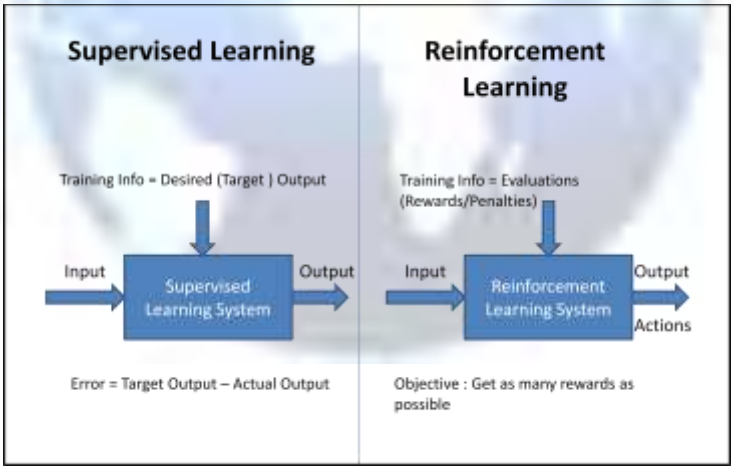

Figure 3: Supervised Learning and Reinforcement Learning

3. RESEARCH QUESTIONS

There are several shortcomings in the existing data migration techniques (i) policies are reactive and not proactive, (ii) data migration initiated at the controller end would not allow heterogeneous storage components in SAN architecture. The design intention is to come up with a proactive data migration initiated at the host end.

This paper aims to answer the following research questions:

(i) Can a single RL agent tune file migration policies for all the storage tiers?
(ii) How does the single agent formulate and then tune policies for the file migration between tiers using RL algorithm?

4. IMPLEMENTATION DETAILS

As a part of implementation this proposal intends to prepare a single data migration agent in a Hierarchical Storage System (HSS) in Linux OS. The data migration agent will be a standalone daemon which is proactive and adaptive. The proactive characteristic will ensure that the agent keeps in check the average response time of all storage tiers and perform data migrations before the waiting time deteriorates. Correspondingly the adaptive nature of the agent will dynamically adjust the data migration rate to avoid QoS violations for the applications. The RL algorithm (Q-Learning) in the agent will formulate and tune policies based on which the data migration will take place. A storage tier is chosen as a candidate for data migration by the agent based on the values of state variable. Figure 4 illustrates the organization of the RL agent and its environment.

4.1 State variables:

The state variables for each storage tier are based on the notion of "temperature" of each file, which approximates its access rate. There are three state variables formed:

(i) S1 : No of read/write requests to the disk per second
(ii) S2 : No of sectors read from and written to the disk per second
(iii) S3 :The average queue length of the requests that were issued to the device

The values of S1, S2 and S3 will help us identify the tier under load. The command iostat gives the value of the three state variables. The values are recorded with timestamp so as to track the improvement/decline of the state variables as a result of migration.

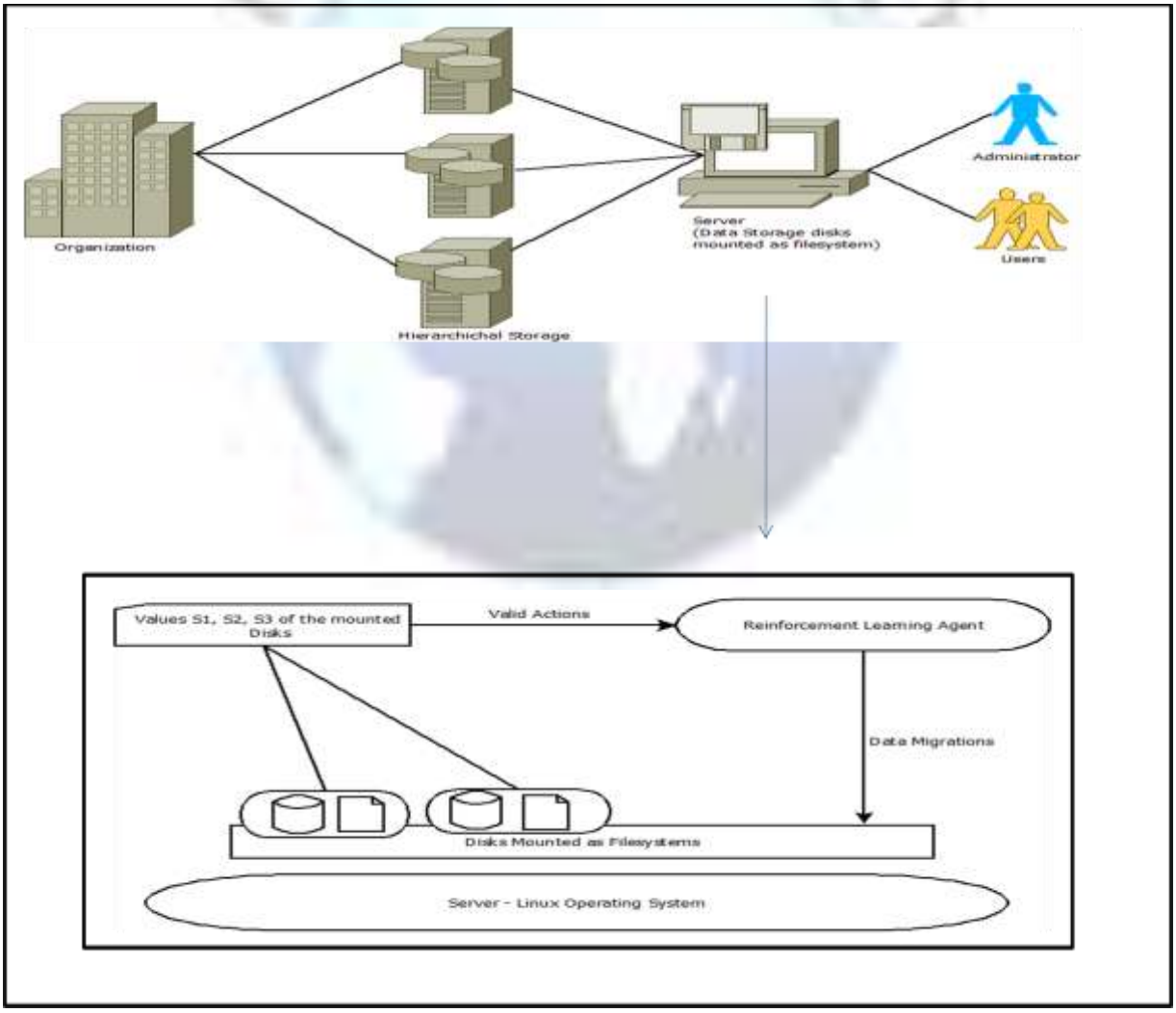

Figure 4: The organization of the system

4.2. Iostat Command:

The iostat command is used to monitor system input/output (I/O) devices (physical and logical). It generates reports which can help us perform load balancing between the devices. In our case we use iostat to generate the 3 state variables. The agent looks at the values of the state variables at regular intervals. If it is found that a particular tier is under a heavy load, that tier becomes a candidate for data migration. The RL algorithm updates the data migration rules and stores them as CRON entries. The CRON entries are triggered to form the data migration.

4.3. CRONTAB:

The cron is software utility in UNIX. It is a time-based job scheduler. Cron is a daemon that executes scheduled commands. Administrators use cron to schedule job to run periodically at fixed times, dates or interval. Crontab is the program used to install/uninstall/run the cron daemon. When the minute, hour, and date match the current time commands are executed by cron. This helps us to initiate data migration at appropriate time so as to not disrupt the application performance. The "data migration rate" is determined by the administrator. It determines the frequency at which the data migration can be triggered by the agent. Crontab is the program used to install/uninstall/run the cron daemon. When the minute, hour, and date match the current time commands are executed by cron. This helps us to initiate data migration at appropriate time so as to not disrupt the application performance. The "data migration rate" is determined by the administrator. It determines the frequency at which the data migration can be triggered by the agent.
The RL-based agent keeps track of the values of the state variables after a data migration has happened. The state variables with most recent timestamps can be compared to find out if the migration task has performed well. If a deterioration of the values is observed a penalty is added to the migration rule, similarly the migration rule which results in improvement of the observed values is rewarded. The migration rules are stored as simple lookup table. Based on the rewards/penalty the agent modifies the destination tiers in the CRON entries.

After some trials, the agent would use the knowledge it learnt to perform near optimal migrations which improves the average I/O request response time of all the storage tiers. Figure 5 models the process and the flow of data between the components in the system.
To validate the effectiveness of data migration policies tuned by the RL algorithm we would need to conduct performance evaluations. Tests can be conducted at the file system level: (1) time to migrate single file of various sizes, (2) time to migrate large number of small files, (3) time to access the migrated files. File system stress test like LTP (Linux Test Project – runltp) also needs to be utilized for appropriate benchmarking of the results. AWRT (average weighted response time) of the different set of policies also need to be considered.

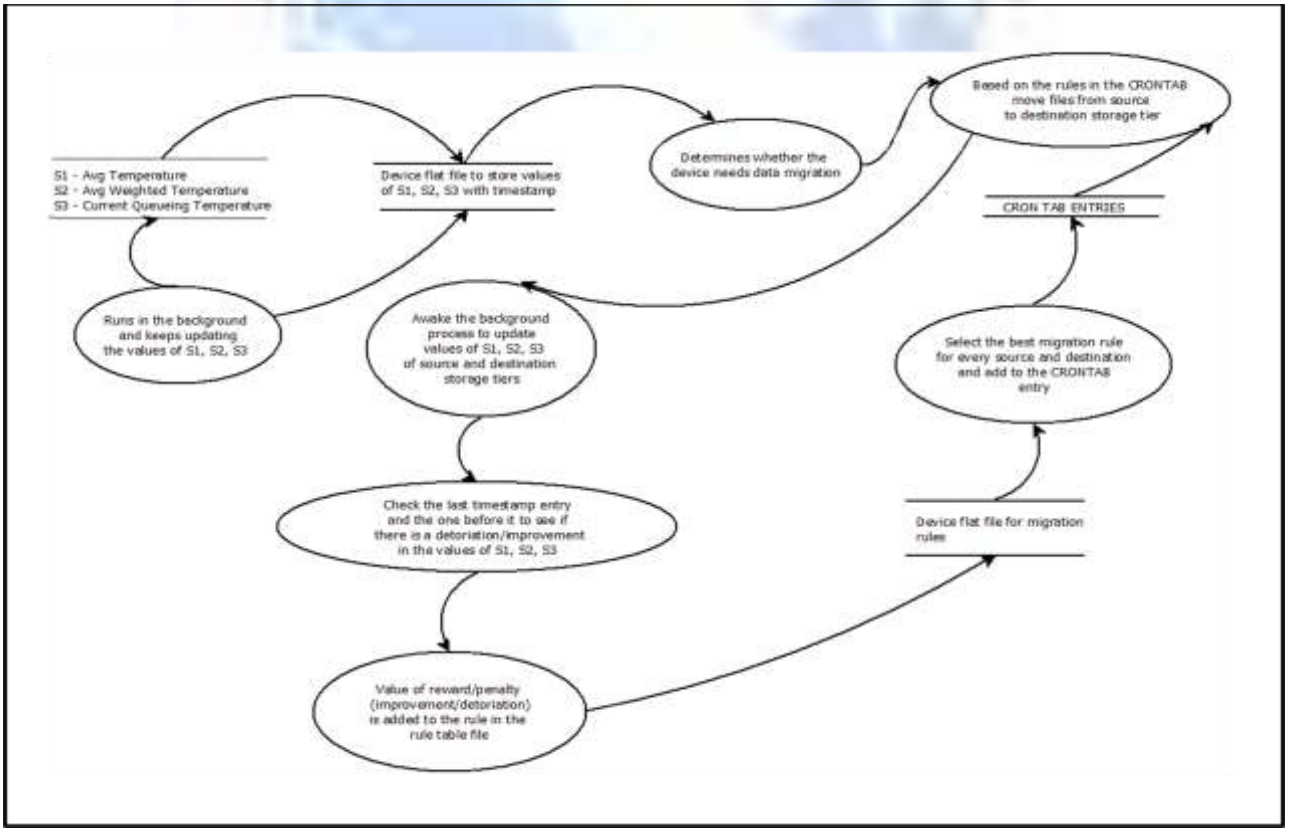

Figure 5: DFD of the proposed system

We can test and compare the AWRT of 2 different set of policies. The first set of policies address the case when no file migration was performed and the policies differ only in the way the files were initially allocated in the system and the second set of pre specified migration policies.

5. DISCUSSION

In the model proposed by David Vengerov [7, 11] it was found that statistically significant performance improvement was achieved only during the first three iterations of the learning cycle, reducing AWRT by 35% in comparison to the initial policy. This corresponds to the practical observations made by many researchers that the agent can learn a very good policy after only several iterations. Another major practical benefit of the RL approach is that it never uses a "bad policy: each subsequent policy is at least as good as the previous one, making RL a good candidate for on-line learning. In the presence of workload dynamics, VCONF [6] was able to adapt to a good configuration within 7 steps and showed 20% to 100% throughput improvement over basic RL methods. The proposed model also aims to achieve similar results in data migration without degrading the existing systems performance.

6. CONCLUSION

This paper presents a novel framework for dynamically re-distributing files in a multi-tier storage system, which explicitly minimizes the expected future average request response time. The previous approaches to storage system management are based on fixed heuristic policies and do not perform any performance optimization. The proposed framework uses a Reinforcement Learning (RL) methodology for tuning policy based on which file migration is performed. Thus we aim to create a proactive and adaptive data migration technique at the host end of SAN which would improve the effective utilization of storage tiers and thereby enhance the total cost of ownership (TCO) of the storage components.

REFERENCES

[1]. Funika, Włodzimierz, Filip Szura, and Jacek Kitowski. "Agent-Based Monitoring Using Fuzzy Logic and Rules." Computer Science 12 (2011): 103-113.
[2]. Floyd Christofferson, "Addressing the Problem of Inactive Data Filling Up Expensive Active Disk Silos ", SGI, April 2011, white paper .
[3]. IBM Redbook, "Hierarchical Storage Management - AS/400e IBM"
[4]. EMC2 Whitepaper, "Data Migration Techniques – A Detailed Review"
[5]. Kaelbling, Leslie Pack, Michael L. Littman, and Andrew W. Moore. "Reinforcement learning: A survey." arXiv preprint cs/9605103 (1996).
[6]. Rao, Jia, Xiangping Bu, Cheng-Zhong Xu, Leyi Wang, and George Yin. "VCONF: a reinforcement learning approach to virtual machines auto-configuration." In Proceedings of the 6th international conference on Autonomic computing, pp. 137-146. ACM, 2009.
[7]. Vengerov, David. "A reinforcement learning framework for online data migration in hierarchical storage systems." The Journal of Supercomputing 43, no. 1 (2008): 1-19.
[8]. R. S. Sutton, A. G. Barto, "Reinforcement Learning: An Introduction", The MIT Press Cambridge 2012
[9]. Qiang, Wang, and Zhan Zhongli. "Reinforcement learning model, algorithms and its application." In Mechatronic Science, Electric Engineering and Computer (MEC), 2011 International Conference on, pp. 1143-1146. IEEE, 2011.
[10]. Chen, Haifeng, Guofei Jiang, Hui Zhang, and Kenji Yoshihira. "Boosting the performance of computing systems through adaptive configuration tuning." In Proceedings of the 2009 ACM symposium on Applied Computing, pp. 1045-1049. ACM, 2009.
[11]. Vengerov, David. "A reinforcement learning framework for online data migration in hierarchical storage systems." The Journal of Supercomputing 43, no. 1 (2008): 1-19.