SCIENCE CHINA
Information Sciences

• RESEARCH PAPER • doi: 10.1007/s11432-011-4466-6

AIP: a tool for flexible and transparent data management

ZHANG GuangYan[1,2*], QIU JianPing [1], SHU JiWu [1,2] & ZHENG WeiMin [1,2]

[1]*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084*, China;*
[2]*Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Beijing* 100084*, China*

Received April 13, 2011; accepted July 20, 2011

Abstract Existing data management tools have some limitations such as restrictions to specific file systems or shortage of transparence to applications. In this paper, we present a new data management tool called AIP, which is implemented via the standard data management API, and hence it supports multiple file systems and makes data management operations transparent to applications. First, AIP provides centralized policy-based data management for controlling the placement of files in different storage tiers. Second, AIP uses differentiated collections of file states to improve the execution efficiency of data management policies, with the help of the caching mechanism of file states. Third, AIP also provides a resource arbitration mechanism for controlling the rate of initiated data management operations. Our results from representative experiments demonstrate that AIP has the ability to provide high performance, to introduce low management overhead, and to have good scalability.

Keywords data management, DMAPI, management policy, differentiated collection, resource arbitration

Citation Zhang G Y, Qiu J P, Shu J W, et al. AIP: a tool for flexible and transparent data management. Sci China Inf Sci, 2012, 55, doi: 10.1007/s11432-011-4466-6

1 Introduction

Migrating data between different tiers adaptively in a multi-tiered storage system has long been a well-known problem [1]. The recent interest in scalable data migration solutions stems from some new trends and technologies. First, an incontrovertible trend is the explosive growth of data. This requires that the cost cannot increase linearly with the storage capacity. Many researches [2–7] on large scale file systems indicate that only about 1% of files are used daily while nearly 90% are never used. The non-uniformity in access patterns of data provides a basis for data migration. Another technological change comes in the form of storage devices. The appearance of new classes of online storage devices, such as SATA drives and flash storage, makes it possible that storage administrators deliver differentiated QoS by migrating data between different storage tiers.

As governmental regulatory requirements on electronic data increase, new emphasis is placed on the need for more complete policy support of data management, not only data migration but also unchangeable retention and timely deletion. Furthermore, the explosive growth of data in recent years requires

*Corresponding author (email: gyzh@tsinghua.edu.cn) data management solutions that are scalable to multiple very large file systems with billions of files. Designing the utility functions that comprise the data management solution, one must also have a knowledge of their impact on the system since 24×7 service is now a common requirement. Therefore, there is a renewed interest in scalable, adaptive, and comprehensive data management solutions.

Existing data management tools are implemented in one of the two ways:

• Inside a file system: Those integrated within file systems can take advantage of internal structures of the file system for improving the performance. Furthermore, they have the opportunity to intercept and redirect data accesses, hence making data management operations transparent to applications. Unfortunately, such a tool is restricted to a specific file system.

• Outside a file system: Those implemented outside file systems adapt to almost all file systems due to adoption of the standard POSIX interface. Without the opportunity of taking advantage of file system internal structures, however, one of their drawbacks is their low performance. Another more serious drawback is that data management operations are not transparent to applications.

For the above reasons, it is required to design a new data management tool, which has reasonable performance and good compatibility, while providing the transparence of data management operations to applications.

In this paper, we present AIP (an acronym for all in position), a tool for flexible and transparent data management. AIP is implemented through the standard data management API called DMAPI [8]. Hence, it supports multiple different file systems and makes data management operations transparent to applications. Especially, AIP has the following typical features. First, AIP provides centralized policy-based data management for controlling the placement of files in different storage tiers. Second, AIP uses differentiated collections of file states to improve the execution efficiency of data management policies, with the help of the caching mechanism of file states. Third, AIP also provides a resource arbitration mechanism for controlling the rate of initiated data management operations.

In order to evaluate the performance of AIP, we implemented a benchmarking tool called DMStone, which reconstructs the state of a real file system and generates the corresponding I/O workload. Some conclusions can be drawn from our experimental results. First, AIP can move user data among different storage tires according to a customized policy, which enables a reasonable distribution of user data dynamically. Second, AIP uses differentiated collections of file states to reduce the execution overhead of data management operations. Third, AIP can control the rate of initiated data migration to reduce the negative impact on normal client operations.

2 Related work

Some efforts [9–11] have been put on how to assess the values of files with regard to multiple arguments, such as access frequencies, last access time, file sizes and file correlations. Furthermore, most storage manufacturers have developed their data management systems. Basically, the systems are implemented in two ways. One is implemented inside the file system, and the other is based on the POSIX interfaces.

2.1 Insides file system

STEPS [12] is a data management architecture implemented in the IBM SAN file system (SFS) [13]. It provides policy-based centralized control [14]. The primary advantage of the system is that it makes use of the parallelism and scalability of SFS, and hence provides a high-efficiency HSM system. Also, file level location independence supported in SFS enables transparent movement of a file without affecting its users and even running applications. Unfortunately, it is limited to SFS, and therefore, cannot be used in other file systems.

He et al. [15] proposed a parallel data movement architecture in object-based cluster file systems, and prototyped it on the Lustre file system. This architecture includes two tiers of disks and tapes. A data miss initiates a data migration from a tape to a disk.

2.2 Outsides a file system

Several other commercially available products also offer such data management [16–18]. They exist outside file systems thereby limiting their ability to take advantage of file system internal structures for scalability.

IBM Tivoli storage manager [19] is a data management tool implemented via the POSIX interfaces. It moves inactive files from the workstation or file server to Tivoli Storage Manager Server according to customized policies, and leaving a small stub in the original location. This tool supports multiple file systems. With this tool, however, data management operations are not transparent to users and applications.

Another data management tool implemented in user space is EMC DiskXtender [20]. It provides policy based file archiving and protection in large scale storage systems. It moves data between different storage pools according to specified policies. To enable application integration for data movement in tiered storage, EMC has developed APIs that allow third-party applications to interface directly with DiskXtender. However, this requires modification of application codes, which is inconvenient and sometimes even impossible.

3 AIP design

3.1 System architecture

AIP is made up of the manager on the data management server (DMServer), the agent on each file system (DMAgent), and the daemon on the backup storage server (DMBack) (Figure 1). They are connected via an Ethernet network.

DMServer maintains the management metadata to keep track of states of storage devices and access information of user files. In addition, DMServer provides a powerful yet simple SQL-like policy language that storage administrators can use to specify policies to control all aspects of data management. DMServer parses management policies, and performs them by sending instructions to DMAgent and DMBack.

DMAgent gets states of files on the primary storage, including file attributes and storage pool status, and sends them to DMServer. It receives instructions from DMServer, and registers monitoring with the DMAPI interface to receive file system events and file access events. When a user or application accesses a file, DMAgent moves it back to the primary storage if the file has been migrated. In this manner, DMAgent maintains a transparent view of data accesses to applications.

DMBack manages all storage devices except the primary storage. These lower storage devices hold all inactive files and are invisible to users and applications. Instructed by DMServer, DMBack obtains the states of lower storage, and conducts data management operations in collaboration with DMAgent.

3.2 Uniting compatibility and transparence

AIP unites compatibility to multiple file systems and transparence to applications via the standard data management API called DMAPI. DMAPI allows data management applications to be developed much like ordinary software applications without the need to modify the OS kernel. AIP has higher portability and maintainability.

AIP's data management operations are transparent to users and even running applications. Transparent data migration includes two aspects: (1) Staging out, AIP reads a file from primary storage via invisible reads and replicates it in lower storage. Then, AIP makes the real size of the file 0 via invisible writes. (2) Staging in, as shown in Figure 2, when a nonresident file is accessed, the user thread is suspended. AIP reads data from lower storage and copies them to primary storage via invisible writes. Then, AIP resumes the user thread to read the file.

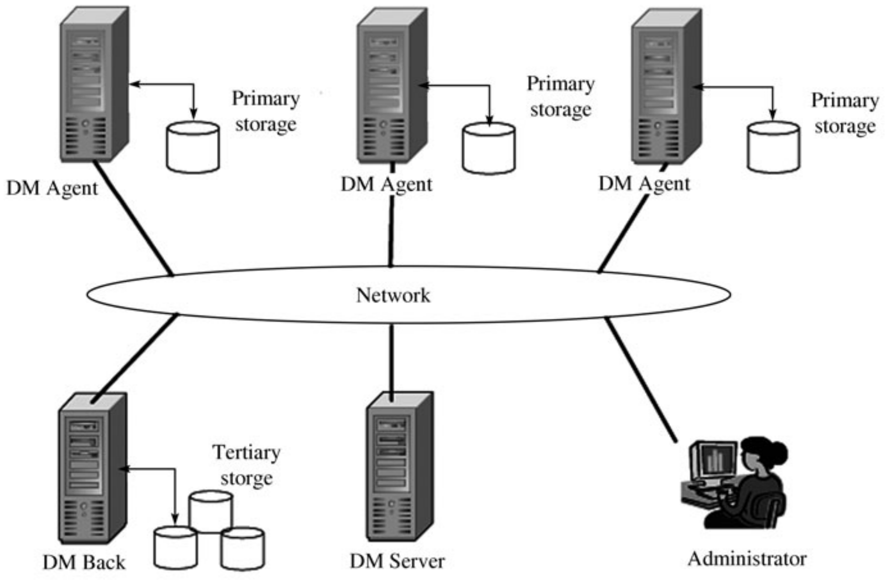

Figure 1 Architecture of the AIP system.

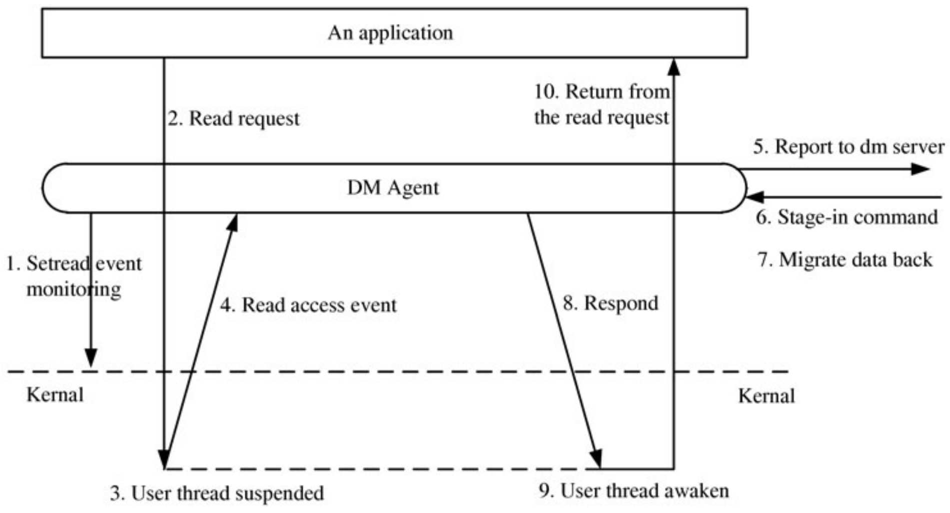

Figure 2 Work flow for reading a nonresident file.

3.3 Policy-based data management

AIP determines the placement of files in different storage tiers with regard to their values. If an assessment method of file values does not work well, it tends to make files migrate frequently between different storage tiers. AIP allows a storage administrator to customize policies for assessing the values of files.

AIP provides a powerful yet simple SQL-like policy language that storage administrators can use to specify policies to control all aspects of data management. The specification of a policy mainly consists of a policy action, file selection clauses, policy trigger and termination conditions, ordering criteria of candidate files, and so on. AIP supports multiple management functions and one non-action specification: migration from one storage pool to another for capacity management, backup, compression, deletion, and exclusion from other policy actions.

An example of policies is shown as follows.

migrate from tier$_1$ to tier$_2$ **where** LAT is 10 days ago
when space utilization of tier$_1$ > 0.8 **until** space utilization of tier$_1$ < 0.5
order by f_size descendly

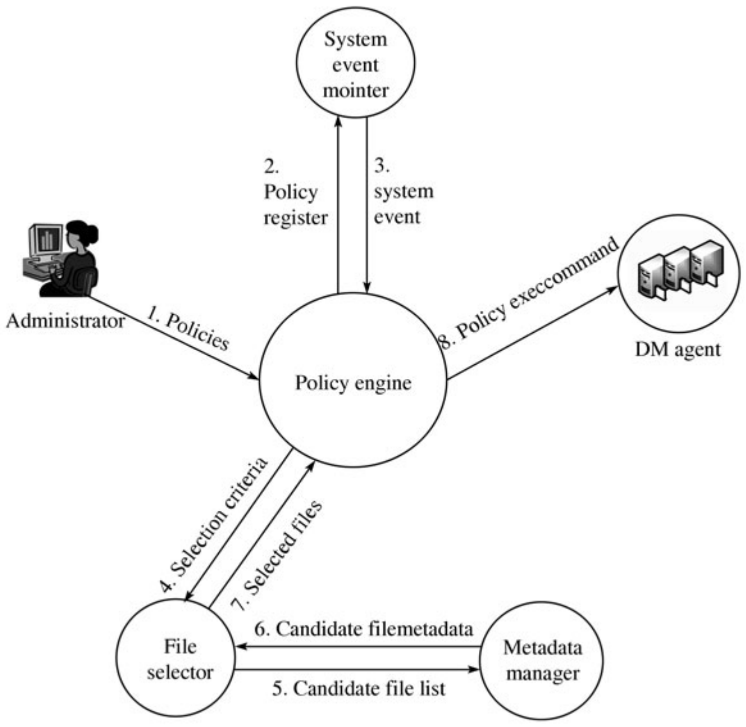

Figure 3 Work flow for executing data management policies.

The policy states that when the utilization rate of the tier$_1$ pool reaches an upper threshold of 80%, specified files should be migrated from the tier$_1$ pool to the tier$_2$ pool, until the utilization rate of the tier$_1$ pool falls below 50%.

The policy engine of AIP controls the whole lifecycle of policies, from their initial creation, activation, execution, to their eventual deletion. Figure 3 demonstrates the work flow of AIP executing data management policies. When the storage administrator delivers a policy, the policy engine parses it and puts it into the policy database. Once a policy is activated, the policy engine registers system events involved in this policy to the system event monitor. When an event is triggered, the policy engine calls the file selector to get the selected files for the corresponding policy action, according to the selection criteria specified in the policy. The file states used by the file selector are retrieved from a database maintained by the metadata manager. The policy engine sends policy execution commands to DMAgent and/or DMBack to initiate data management actions.

3.4 Differentiated collection of file states

The selection of candidate files for data management operations requires the states of all files, including file sizes, last access time, last modified time and so on. Some measurements show that it requires 27 hours to scan one billion files in the file system namespace [12]. To avoid scanning the entire file system metadata for each policy execution, we propose a concept of a file state cache in which the entire metadata is scanned at policy initialization time and a file state database is built in the DMServer for future policy actions. This significantly reduces the cost of policy execution as the problem of candidate file selection is now reduced to database lookup in the DMServer.

When selecting candidate files is performed, the content of the file state database is required to be consistent with the file system metadata related to the policy. Differentiated collections are used to reduce the negative impact of collecting file states on the performance of the managed file system.

First, AIP eliminates attribute collections of all staged-out files. A staged-out file has not been accessed, and hence its attributes cannot be modified. Gibson et al. [5] examined three different file system environments for periods of 150 to 280 days, and found that most files (approximately 90%) are never

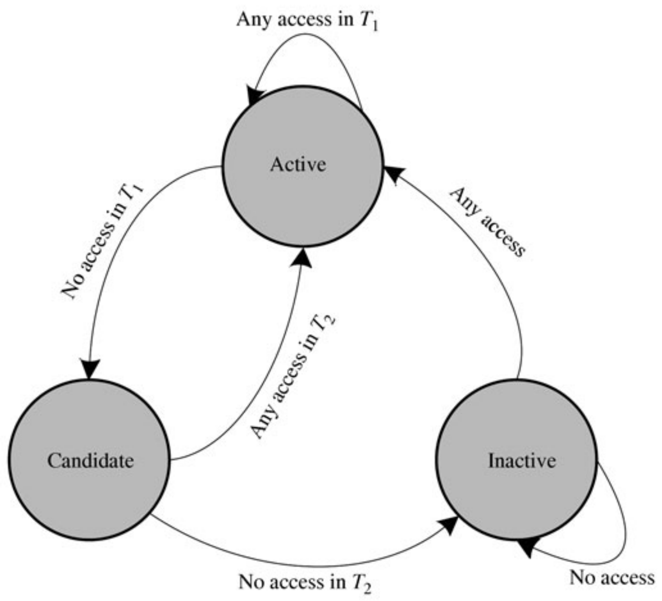

Figure 4 The state transition diagram describing three states of data.

used. The percentage increases with increasing size of the file system. So this technique can obviously reduce the cost of collecting file attributes.

Second, when last access time is concerned in a policy, AIP updates the file state database in a hybrid manner to tolerate potential changes in the file system behavior. File systems usually offer mount options, e.g., noatime and relatime, to determine whether to update the access time of a file when its file data is read or written. Many applications cannot cope with this because they rely on this specific detail of the file system semantics. AIP introduces the candidate file state, besides the active and inactive states. It retrieves last access time from the file state database periodically. As shown in Figure 4, if the access time of a file is within the period of time T_1, this file is active and AIP does not collect its file state. Otherwise, this is a candidate file, and AIP sets up DMAPI monitoring for it. If there is any access to this candidate file within the future period of time T_2, this file gets active and its file state is updated. Otherwise, this candidate file gets inactive, and therefore stages out to the lower storage.

3.5 Resource arbitration for data migration

Since data management is performed online, data migration and foreground applications share and even contend for the I/O resources in the DMAgent system. AIP provides a block-level resource arbitration mechanism so as to maximize migration rate dynamically without a significant impact on application performance.

Due to highly variable service time in storage systems and workload fluctuations on arbitrary time scales [7], AIP uses an on/off logical valve to adjust the migration rate. Whether data migration performs depends on application workload that the DMAgent serves. Data migration performs if the workload is relatively low, whereas it is throttled if the workload is high.

AIP monitors the length of I/O queue inside the DMAgent. When the length is greater than a threshold T, the workload is adjudicated to be high, and vice versa. Generally speaking, average I/O response time increases with the length of the I/O queue. AIP determines the time of a migration operation to suspend W in terms of the length of the I/O queue L by the equation $W = E \times (L - T)$. Here, E is a constant, and the values of E and T are set up by a rule of thumb.

4 AIP implementation

4.1 DMAgent implementation

To provide transparent accesses of applications to a staged-out file, AIP uses synchronous events to monitor any access to this file. To lower the impact of file state monitoring to the performance of foreground applications, AIP uses asynchronous events to collect the access information about those files on the primary storage.

DMAgent gets attributes of files and file systems by calling the POSIX interface *stat* in Linux or by calling the corresponding DMAPI function. DMAgent reads diskstats in /proc file system to get the length of I/O queue, on which the resource arbitration for data migration depends.

4.2 DMBack implementation

DMBack can use an arbitrary file system, and even multiple different file systems to manage lower storage. The main responsibility of DMBack is to move files between different storage pools. In our implementation, a monitor thread is used to monitor the migrate request from DMAgent. In addition, DMBack reports the states of storage pools to DMServer.

4.3 DMServer implementation

DMServer stores file attributes and policies in a database. DMServer creates a monitor thread to get requests from DMAgent or the administrator interface. If a report of file access from DMAgent is received, DMServer initiates different actions in terms of the type of the access. If it is a FILE CREATE operation, DMServer adds the file and its attributes to the database. If it is a FILE DELETE operation, it sends a delete request to the destination storage pool to delete the file.

5 Experimental evaluation

5.1 Benchmarking tool

A tool for benchmarking a data management system is required to create realistic file-system state and to generate representative workloads. We implemented DMstone, a new benchmarking tool to evaluate the performance of AIP. DMstone is composed of three parts. The constructer is a module used to reconstruct the scenario of a large scale file system from a snapshot of plan 9 file system [21]. The generator generates the file access traces based on the changes of different snapshots. The replayer replays the I/O traces on the scenario we reconstructed. Plan 9 file system retains daily snapshots of its contents from 1990 until 2002.

DMstone opens a file, sends the designated read/write requests to the file in terms of the timestamp and request type of each I/O event in the trace, waits for the completion of the I/O requests, and gathers the corresponding I/O results, such as response time and throughput.

5.2 Experimental setup

In our experiments, we used three machines—running DMAgent, DMServer and DMBack. These machines are connected over a 1000 Mb Ethernet network. The configurations of these machines are listed in Table 1.

The file system scenario we used is reconstructed from the snapshot of May 5th 1993. The total number of files is 577243 and the total size is 22.7 GB. The files that have been accessed in the past 10 days account for only 2.8% of all the files. Most of the files have not been accessed recently.

Table 1 Configurations of machines

	CPU	Memory (GB)	Hard Disk(RPM)
DMAgent	Intel(R) Xeon(R) 2.50GHz	3	15000
DMServer	Intel(R) Xeon(R) 2.40GHz	4	7200
DMBack	Intel(R) Xeon(R) 1.60GHz	6	7200

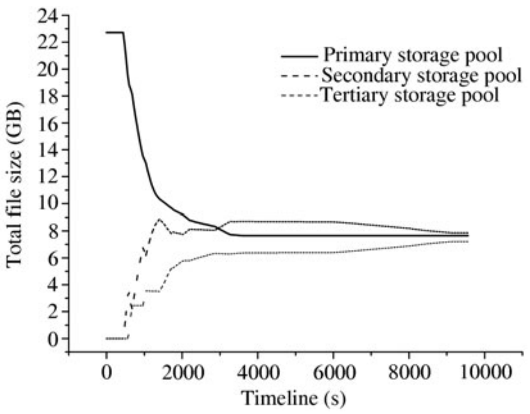

Figure 5 Space utilization rate changes of the three storage pools.

5.3 Effect of data hierarchical storage

We used three storage tiers in this experiment to evaluate AIP's ability of migrating data adaptively. The capacity of the primary storage is 9 GB. We use the following policies to migrate files between different tiers:

• Policy A: When the utilization rate of the primary storage is over 95%, AIP moves those files whose last access time is 10 days ago to the secondary storage in a descending order of file size, until the utilization rate of the primary storage becomes lower than 85%.

• Policy B: AIP moves those files whose last access time is 150 days ago from the secondary storage to the tertiary storage.

The evaluation is composed of two stages. The first stage is activating the policies to move files. Once the utilization rate of the primary storage gets lower than 85%, DMstone replays traces. During the test, we record the utilization rate of the three storage pools every 40 seconds.

The result in Figure 5 shows that the space utilization rate of the three storage pools changes with time. In the first stage, the space utilization rate of the primary storage pool decreases with time. This is the effect of Policy A that is migrating files to the secondary storage. Policy A moves files in a descending order of the file size. Since transferring larger files presents a larger throughput, the decrease of the space utilization rate slows down gradually.

As to the secondary storage pool, the space utilization rate is affected by both Policys A and B. Policy A moves files to it, so the space utilization rate of the secondary storage keeps on growing as a whole. Policy B moves files from it to the tertiary storage every 6 minutes, and the utilization rate of the secondary storage also presents some reduction locally.

In the second stage, DMstone replays the traces. FILE CREATE and FILE DELETE requests are included in these traces. The space utilization rate of the primary storage fluctuates as the CREATE and DELETE requests are performed. Since these requests do not affect the secondary and tertiary storage pools, their space utilization rate keeps stable. It can be seen that AIP can redistribute files among multiple different storage pools according to the customized policies.

5.4 Effect of optimizing collections of file states

We used two policies in this experiment to evaluate the mechanism used to optimize the collections of file states.

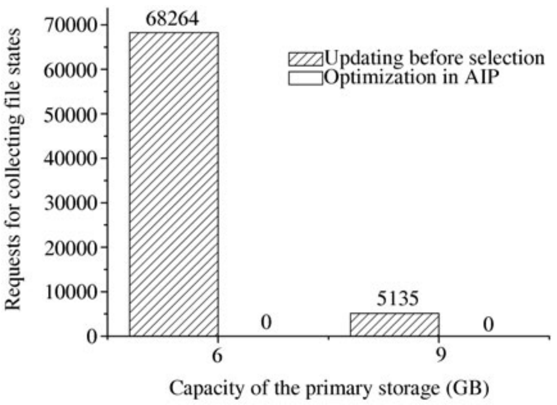

Figure 6 The number of requests for collecting file states when executing Policy B.

• Policy A: When the utilization rate of the primary storage reaches 95%, AIP moves those files whose last access time is 10 days ago to the secondary storage in a descending order of file size, until the utilization rate of the primary storage becomes lower than 85%.

• Policy B: AIP moves those files whose last access time is 150 days ago and whose file sizes are over 1 MB from the secondary storage to the tertiary storage.

The baseline method does not store file states in the database. Once AIP selects candidate files for a policy action, the baseline method collects file states, including last access time, last modification time, and file sizes. In two experiments, the capacity of the primary storage is set to 9 GB and 6 GB respectively. We collected the number of requests for collecting file states when executing Policy B, using the baseline method and the AIP method.

Figure 6 compares the numbers of requests for collecting file states using the two methods. As far as the baseline method is concerned, the collection number is 5135 when the capacity of the primary storage is 9 GB; the collection number is 68264 for 6 GB. This difference originates from the fact that the smaller the primary storage is, the more files are migrated to the secondary storage. As a result, states of more files are required to be collected in executing Policy B.

When the optimization in AIP is used, the collection number is 0, either for the 9 GB capacity or for the 6 GB capacity. This indicates that AIP's optimization in collecting file states can eliminate the state collections of files on the secondary storage. We can claim that the differentiated collections of file states in AIP will benefit more as the managed file system gets larger.

5.5 Effect of resource arbitration

To perform resource arbitration for data migration, AIP measures the length of I/O queue to assess the foreground workload. From the queuing theory, the response time of an I/O request is proportional to the length of I/O queue. To validate this relation using the real-system experiment, we wrote data to a disk continuously, and recorded the I/O queue length via reading the Linux /proc file system. At the same time, we sent a request for reading 4 KB data each second, and collected its response time. As shown in Figure 7, the response time is sensitive to the length of the I/O queue. Also, we can see that the increase of response time lags slightly behind the increase of the length of I/O queue. This indicates that the length of the I/O queue can be used to predict the foreground workload.

In this experiment, we replayed traces for 20 minutes with and without rate control respectively. During the experiment, we record the response time for each request and then get the average for all requests. Figure 8 shows the measured response time with and without rate control of data migration. Compared to the performance without rate control, the rate control in AIP reduces the average response time by about 9.63%.

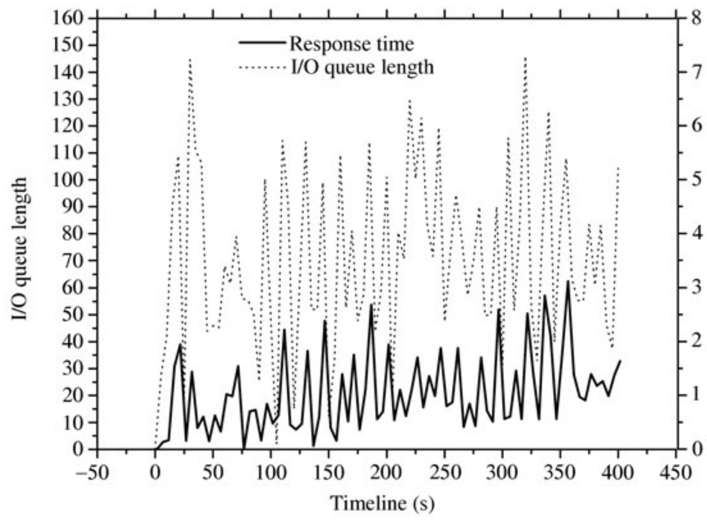

Figure 7 Relation between the I/O queue and the response time.

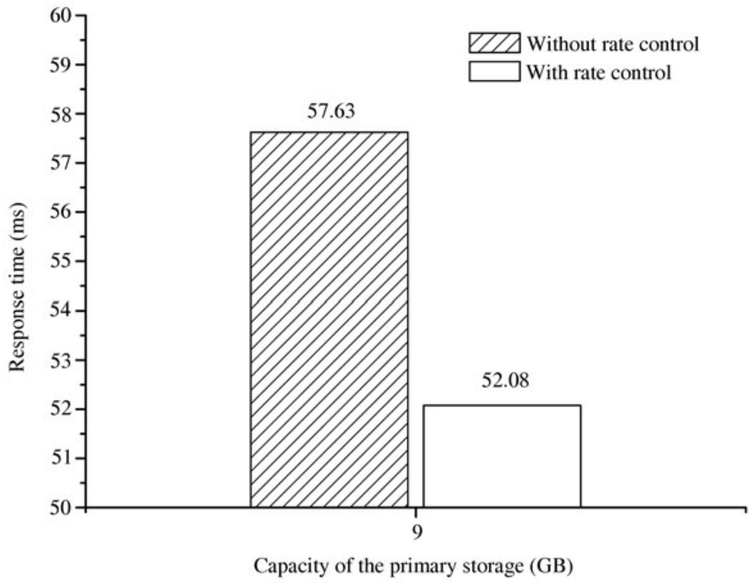

Figure 8 Average response time with and without rate control.

6 Conclusions and future work

In this paper, we present AIP, a tool for flexible and transparent data management. It is implemented through the standard data management API called DMAPI. Hence, AIP supports multiple file systems and makes data management operations transparent to applications. Furthermore, it supports multiple policy-based data management operations, such as data migration, data backup, data compression, and data deletion.

By reconstructing the state of a real file system and replaying its corresponding I/O traces, we evaluated the performance of AIP. Some conclusions can be drawn from our experimental results. First, AIP can move user data among different storage tires according to a customized policy. Second, AIP uses differentiated collections of file states to reduce the execution overhead of data management operations. Third, AIP can control the rate of initiated data movement to reduce the negative impact on normal client operations.

AIP moves data from low tier to high tier only when it is accessed, which results in a noticeable latency for accessing a staged-out file. In the future, we will develop a method for identifying and prefetching data that will be accessed with more possibility. Our preliminary idea is to use near-line data mining techniques to extract access patterns for data prefetching.

Acknowledgements

This work was supported by National Natural Science Foundation of China (Grant Nos. 60903183, 61170008), National High Technology Research and Development Program of China (Grant No. 2009AA01A403), and National Basic Research Program of China (Grant No. 2007CB311100).

References

1 Smith A J. Long term file migration: development and evaluation of algorithms. Commun ACM, 1981, 24: 521–532
2 Douceur J R, Bolosky W J. A large-scale study of file system contents. In: Proceedings of the 1999 ACM SIGMETRICS Conference. New York: ACM, 1999. 59–70
3 Vogels W. File system usage in Windows NT 4.0. In: Proceedings of the 17th ACM Symposium on Operating Systems Principles. New York: ACM, 1999. 93–109
4 Wang F, Xin Q, Hong B, et al. File system workload analysis for large scale scientific computing applications. In: Proceedings of the 12th NASA Goddard, 21st IEEE Conference on Mass Storage Systems and Technologies(MSST 2004). Washington DC: IEEE, 2004. 139–152
5 Gibson T J, Miller E L, Long D D E. Long-term file activity and inter-reference patterns. In: Proceedings of 24th International Conference on Technology Management and Performance Evaluation of Enterprise-Wide Information Systems. California: Computer Measurement Group, 1998. 976–987
6 Gibson T J, Miller E L. Long-term file activity patterns in a UNIX workstation environment. In: 15th IEEE Symposium on Mass Storage Systems. Washington DC: IEEE, 1998. 355–371
7 Gribble S, Manku G, Roselli E, et al. Self-similarity in file systems. In: SIGMETRICS98. New York: ACM, 1998. 141–150
8 Miroshnichenko A. Data management API: the standard and implementation experiences. In: Proceedings of AUUG 96 & Asia Pacific World Wide Web. NSW: AUUG, 1996. 271–282
9 Jin H, Xiong M Z, Wu S. Information value evaluation model for ILM. In: ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing. Washington DC, 2008. 543–548
10 Zhao X N, Li Z H, Zeng L J. A hierarchical storage strategy based on block-level data valuation. In: 4th International Conference on Networked Computing and Advanced Information Management. Washington DC: IEEE, 2008. 36–41
11 Vengerov D. A reinforcement learning framework for online data migration in hierarchical storage systems. J Supercomput, 2008, 43: 1–19
12 Verma A, Pease D, Sharma U, et al. An architecture for lifecycle management in very large file systems. In: Proceedings of the 22nd IEEE / 13th NASA Goddard Conference on Mass Storage Systems and Technologies. Washington DC: IEEE, 2005. 160–168
13 Menon J, Pease D A, Rees R, et al. IBM storage tank-a heterogeneous scalable SAN file system. IBM Syst J, 2003, 42: 250–267
14 Beigi M, Devarakonda M V, Jain R, et al. Akshat verma: policy-based information lifecycle management in a large-scale file system. In: POLICY '05 Proceedings of the 6th IEEE International Workshop on Policies for Distributed Systems and Networks. Washington DC: IEEE, 2005. 139–148
15 He D S, Zhang X B, Du D H C, et al. Coordinating parallel hierarchical storage management in object-base cluster file system. In: Proceeding of 23nd IEEE–14th NASA Goddard Conference on Mass Storage Systems and Technologies. Washington DC: IEEE, 2006. 219–234
16 Gelb J P. System-managed storage. IBM Syst J, 1989, 28: 77–103
17 Kaczmarski M, Jiang T, Pease D. Beyond backup towards storage management. IBM Syst J, 2003, 42: 322–338
18 Anonymous. Veritas data protection products. 2004. http://veritas.com
19 Brooks C, McFarlane P, Pott N, et al. IBM tivoli storage management concepts. http://www.redbooks.ibm.com/redbooks/pdfs/sg244877.pdf
20 EMC Corporation. A better approach to managing file system data, lowering costs, reducing risk, and managing data growth. EMC White Paper. 2006
21 Pike R, Presotto D, Dorward S, et al. Plan 9 from Bell Labs. Comput Syst, 1995, 8: 221254