

An Architecture for Lifecycle Management in Very Large File Systems

Akshat Verma*
IBM India Research Lab
akshatverma@in.ibm.com

Upendra Sharma
IBM India Research Lab
supendra@in.ibm.com

Jim Rubas
IBM Watson Research
rubas@us.ibm.com

David Pease
IBM Almaden Research
pease@almaden.ibm.com

Marc Kaplan
IBM Watson Research
makaplan@us.ibm.com

Rohit Jain
IBM India Research Lab
rohitjain@in.ibm.com

Murthy Devarakonda
IBM Watson Research
mdev@us.ibm.com

Mandis Beigi
IBM Watson Research
mandis@us.ibm.com

Abstract

We present a policy-based architecture STEPS for lifecycle management (LCM) in a mass scale distributed file system. The STEPS architecture is designed in the context of IBM's SAN File System (SFS) and leverages the parallelism and scalability offered by SFS, while providing a centralized point of control for policy-based management. The architecture uses novel concepts like Policy Cache and Rate-Controlled Migration for efficient and non-intrusive execution of the LCM functions, while ensuring that the architecture scales with very large number of files.

The architecture has been implemented and used for lifecycle management in a distributed deployment of SFS with heterogeneous data. We conduct experiments on the implementation to study the performance of the architecture. We observed that STEPS is highly scalable with increase in the number as well as the size of the file objects hosted by SFS. The performance study also demonstrated that most of the efficiency of policy execution is derived from Policy Cache. Further, a rate-control mechanism is necessary to ensure that users are isolated from LCM operations.

1. Introduction

Migrating data between different tiers of a multi-tiered storage system according to the changing quality of service (QoS) needs of data during its lifecycle is a

well known problem[8]. Several commercially available products offer such hierarchical storage management [2, 4, 11]. However, increased governmental regulatory requirements on electronic data places new emphasis on the need for more complete policy control of data lifecycle management. Furthermore, the explosive growth of data in recent years requires lifecycle management solutions that are scalable to very large file systems with billions of files and that address not only the migration of data but also unchangeable retention and timely deletion.

The utility functions that comprise the lifecycle management solution must also be aware of their impact on the system since 24x7 service is now a common requirement. Therefore, there is a renewed interest in scalable, adaptive, and comprehensive lifecycle management solutions. In this paper, we present an architecture and implementation for lifecycle management (LCM) in a large-scale distributed file system that addresses these challenges.

1.1. Motivation for a comprehensive LCM Solution

The increased interest in scalable lifecycle management solutions stems from some new technologies that have emerged. The evolution of file level location independence supported in the new file systems like IBM SAN File System (SFS) [6] enables transparent movement of a file without affecting its users and (even running) applications. Another technological change comes in the form of Serial ATA drives that offer inexpensive, higher capacity, but lower performance storage alterna-

*Authors are listed in reverse-alphabetical order.

tive to enterprise-class SCSI drives. Taking advantage of this new class of online storage, storage administrators deliver differentiated QoS by migrating data between the storage classes as the performance requirements of data changes over time. To take the simple example of an email server, the recent messages may be hosted on SCSI drives when they arrive and moved to cheaper SATA drives as they become older (and their access probability goes down). These technology trends have led to an environment where the lifecycle of a file may involve many migrations. This is a stark contrast to the typical file lifecycle model for which existing lifecycle management solutions are designed; i.e., creation followed by backup and archival, a model where the file data would move only once, usually from a SCSI device to a tape pool. These additional complexities in file lifecycle require us to develop efficient infrastructure for large-scale lifecycle management that can deal with a large number of migrations.

Also, in existing lifecycle management solutions, there often are multiple tools designed to control the various aspects of the file lifecycle. For example, backup and archival to tape, migration to near-line storage, and migration to WORM storage are under the control of different tools. Managing multiple tools to provide an integrated lifecycle strategy is complex, error-prone and can overburden the job of an administrator. Further, current lifecycle management solutions are not integrated within the file system but instead exist outside the file system thereby limiting their ability to take advantage of file system internal structures for scalability. In large scale file systems with billions of files, selecting candidate files for lifecycle management operations can be extremely resource intensive. Naive implementations scan the entire file system namespace for candidate files on every policy invocation. Studies [3] show that less than 1% of files are modified every day in a file system with 200,000 files, and the percentage reduces with increasing size of the file system. Our measurements show that roughly 5000 files can be scanned per second, which implies that a naive implementation requires 27 hours to scan one billion files on every policy invocation.

Hence, one needs to design file lifecycle management solutions that are scalable, provide an integrated control for all management functions, and has mechanisms to ensure that the large number of migrations do not disrupt regular client traffic.

1.2. Contribution

We present the Storage Tank Enhanced Policy Service (*STEPS*) architecture for policy-based file lifecycle management that provides an integrated and central

management control for all lifecycle management functions in a non-disruptive manner. In order to capture the diverse lifecycle management functions, we have provided a powerful yet simple policy language that storage administrators can use to specify policies to control all aspects of file lifecycle management. The architecture provides centralized policy-based management for controlling the placement of files in different storage tiers, from their initial creation to their eventual deletion or secure archival. The policy execution is designed to scale to billions of files while minimizing the impact on ongoing client workloads.

To avoid the scanning of the entire filesystem metadata for each policy execution, we propose a novel concept of a Policy Cache in which the entire metadata is scanned at policy initialization time and a cache is built of future policy actions applicable to each file. The cache is updated subsequently, using a lazy and batched approach, as a result of modifications to files, file creations and deletions. This significantly reduces the cost of policy execution as the problem of candidate file selection is now reduced to cache lookup. In our example of a file system with one billion files, our policy cache will reduce the scanning to the actual set of files modified, which is likely to be less than 1%; thus it will take less than 15 minutes to find candidates for LCM actions even in this huge file system.

We also provide a resource arbitration mechanism for controlling the rate of LCM initiated data movement in order to minimize the impact of LCM operations on normal client operations. LCM operations often require migration of large amounts of data between storage tiers. For very large scale file systems, the time required to complete data migration may be several hours during which client access to the system could be severely affected. For those large 24x7 service providers, such an impact would have a negative affect on the business. We provide a control-theoretic mechanism to adapt the rate of migration according to changes in client workloads so as to minimize the impact on them.

We have implemented this architecture in The IBM SAN File System (SFS) also known as Storage Tank [6]. SFS is a distributed file system for SAN-attached storage that supports heterogeneous clients. Our implementation leverages advanced management concepts provided in SFS such as separate centralized metadata management, namespace partitioning into containers, and storage pools. However, the architecture is general enough to be applicable for implementation in other distributed file systems. Experimental results validate that the *STEPS* architecture is scalable and the rate-control mechanism is effective in ensuring that LCM activities do not disrupt regular client traffic.

The rest of this paper is organized as follows. In section 2, we present a detailed design of our system along with a discussion on the design choice that we made. In section 3, we present the details of our prototype implementation, our performance study and results regarding the scalability and efficiency of the implementation. Finally, we conclude with our key observations in section 4.

2. Framework and Architecture for SAN File System Policy Service

We describe a policy-based lifecycle management architecture, Storage Tank Enhanced Policy Service (STEPS), in the context of the IBM SAN File System [6], a distributed filesystem that can support a large number of file objects. We first provide a brief overview of the IBM SAN File System (SFS).

2.1. SFS Overview

The IBM SAN File System is architected to be a highly available file system for SAN-attached storage; it is designed to provide a network-based, heterogeneous file system for data sharing and centralized policy-based storage management in an open systems environment. SFS is designed to enable host systems to plug into a common SAN-wide file structure (Fig. 1). With SFS, files and file systems are no longer managed by individual computers; instead, they are viewed and managed as a centralized IT resource with a single point of administrative control. SFS provides a common file system for UNIX, Windows and Linux servers, with a single global namespace to help provide data sharing across servers.

SFS maintains all the file system metadata on a dedicated metadata server (MDS) cluster. SFS clients retrieve the physical location of a particular file segment from the MDS, and directly fetch the data from the disks attached to the SAN. SFS also uses dedicated SFS clients to perform any bulk data copy for LCM functions, thus ensuring that data access operations never flow through the metadata servers. The server-free data path along with the aggressive caching of metadata supported by SFS ensures that data access is not affected greatly by server congestion, thus making the system highly scalable without the limitations normally associated with Network File System (NFS) or Common Internet File System (CIFS) implementations.

Unlike many file systems, the name space in SFS is completely decoupled from the storage space; that is, a file's location in the name tree has no connection with its location in the storage subsystem. The name space is

subdivided into segments called "containers"; containers are used for various management purposes in SFS, including server load assignment. The storage space is subdivided into "pools"; pools are named collections of storage volumes; at any point in time, a file's data is stored in one pool.

The metadata servers designate one of their number as a master node with all the other servers being subordinate to it. At a file system level, the master MDS assigns containers to individual subordinate nodes, which are responsible for managing the assigned container. The subordinate node that manages a specific container allocates space for the files belonging to that container; a file's data is allocated on one or more of the volumes of the appropriate pool during its lifetime, as specified by the LCM policies. The distributed server cluster and random master election provide a high level of concurrency, that in turn ensures scalability of SFS with large number of files, and fault tolerance in the presence of server failures.

However, the distributed nature of the MDS cluster presents challenges in designing an LCM policy infrastructure that can be controlled centrally without affecting the scalability and the high degree of concurrency in the SFS. Moreover, the policy infrastructure should not violate the intrinsic design principle that management components do not sit in the datapath. Further, LCM operations like backup and archival that deal with tape or optical pools require us to integrate the policy architecture efficiently with a (possibly) third party tertiary storage manager. We now present an architecture that achieves the above goals.

2.2. The STEPS Architecture

The STEPS architecture has a policy user interface that is used by an administrator to create "high level" lifecycle management policies. A policy is a tuple of a condition and an LCM action (migration, replication, deletion, backup or archival) on a datagroup. A datagroup is a collection of files that share certain common properties; it is specified by a boolean expression of file attributes. An example of a common policy used for LCM is

*If <space utilization of PREMIUM_POOL is greater than 80% > then <MIGRATE *.tmp files greater than 1 MB ordered by size to IDE_POOL until the space utilization of PREMIUM_POOL is 60% >.*

The datagroup is defined as the set of files whose name matches the pattern *.tmp and whose size is greater than 1 MB, ordered by size.

The central control in our architecture (Fig. 2) lies within a File Policy Scheduler and Orchestrator (FPSO) that determines if a particular policy is active at any point

