GiSK: Making Secure, Reliable and Scalable VO Repository Virtualizing Generic Disks in the Grid

Eunsung Kim Hyeong S. Kim Heon Y. Yeom
Department of Computer Science and Engineering
Seoul National University
Seoul, Korea
Email: {eskim,hskim,yeom}@dcslab.snu.ac.kr

Jongsook Lee
Supercomputing Center
Korea Institute of Science and Technology Information
Daejeon, Korea
Email: jsruthlee@kisti.re.kr

Abstract— **Recently, the Grid has been recognized in industry as an important means of addressing issues of virtualization and distributed system management. Storage virtualization is the logical abstraction of physical storage and has tremendous potential for simplifying storage administration and reducing costs for managing diverse storage assets.**

In this paper, we study ways to virtualize heterogeneous storage systems in Grid and introduce a new secure, reliable, and scalable VO repository system, referred as GiSK, using storage virtualization for generic disks in the Grid. Through the lessons from GiSK, we identify a few important issues required to realize a storage virtualization system in the Grid and suggest the "Lassoing a stake in order" strategy for data reliability and the fine-grained access control mechanism using the GSI certificate.

I. Introduction

The Grid is a collection of computing resources which are distributed geographically over a network But it appears as a single computer system. These resources can be computers, databases, or computer-controlled instruments. The Grid makes these resources available to everyone connected to the Grid without sacrificing their local autonomy in order to share resources and perform some collaborative work. In other words, the Grid provides an infrastructure for federated resource sharing across trust domains. The trust domain is referred as "virtual organization (VO)" [1] [2].

Grid technologies have evolved in three generations: early ad hoc solutions, de facto standards based on the Globus Toolkit (GT), and, at present, more formal Web services (WS)-based standards within the context of the Open Grid Services Architecture (OGSA) [3]. OGSA adopts WS standards such as Web Services Description Language (WSDL) as basis for a service-oriented architecture within which arbitrary services can be defined, discovered, and invoked in terms of their interfaces rather than their implementations. This approach provides a basis for virtualization, interoperability, and composition [4].

The Grid community like the Global Grid Forum (GGF) has led or participated in the development of WS specifications that address other Grid requirements. The WS-Resource Framework (WSRF) [5] defines uniform mechanisms for defining, inspecting, and managing a remote state, a crucial concern in many settings. WSRF mechanisms underlie work on service management (WSDM, in OASIS) and negotiation (WS-Agreement, in GGF). These efforts are crucial to the Grid vision of large-scale, reliable, and interoperable Grid applications and services. Other relevant efforts are aimed at standardizing interfaces to data, computers, and other classes of resources.

Work on Grid-related standards is driven by the work of a vibrant open source community. Its most recent instantiation GT4, which is Web services-based and WSRF-compliant, provides basic middleware to create VO's, addressing such issues as specification and enforcement of VO-wide policy, discovery, provisioning and management of services and resources, and federation, replication, discovery, and movement of data. At deployment, depending on available resources and planned applications, specific service implementations can be chosen and deployed, often in conjunction with other GT-based components.

Grid technology R&D has produced specifications and technologies for realizing service-oriented architectures according to robust distributed system principles. Global control mechanisms are able to deal reliably with failure and adapt to changing environmental conditions, so application concerns have been a lesser concern [3].

Early application drivers were largely from scientific computing and included large-scale distributed computing (federation of computers), integration of large-scale data repositories (data grids), collaboration, and tele-instrumentation. More recently, the technology has been used in industry as a means of addressing issues of virtualization and distributed system management.

The data storage industry is one of the most dynamic fields in information technology today. Due largely to the advent of high-performance networking between servers and storage assets, storage technology has undergone a rapid transformation as new innovations have pushed storage solutions forward. At the same time, the viability of new storage technologies is repeatedly affirmed by the rapid adoption of networked storage by virtually every large enterprise and institution. However, with development comes liability.

Since the early 1990s, storage innovation has produced a steady stream of new technology solutions, including Fiber Channel, network-attached storage (NAS), server clustering, serverless backup, high-availability dual-pathing, point-in-time

Proceedings of the Eighth International Conference on High-Performance Computing in Asia-Pacific Region (HPCASIA'05)
0-7695-2486-9/05 $20.00 © 2005 IEEE

data copy (snapshots), shared tape access, storage over distance, iSCSI, CIM (common information model)-based management of storage assets and transports, and storage virtualization. Each of these successive waves of technical advance has been accompanied by disruption to previous practices, vendor contention, over-hyping of what the new solution could actually do, and confusion among customers.

No storage networking innovation has caused more confusion in today's market than storage virtualization. Storage virtualization is the logical abstraction of physical storage systems and aims to hide the complexity of physical storage devices and their specific requirements from management view. Storage virtualization has tremendous potential for simplifying storage administration and reducing costs for managing diverse storage assets [6].

In this paper, we study methods to virtualize storage systems in Grid, and introduce a new secure and reliable VO repository system, referred as GiSK, using storage virtualization for generic disks in the Grid.

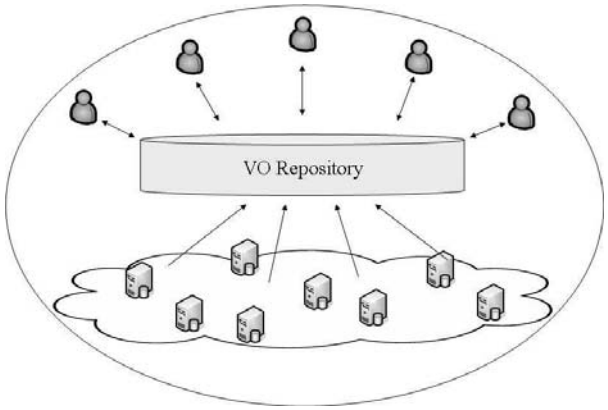

Fig. 1. The virtualization of heterogeneous storage system makes up a VO repository. The repository appears as a shared large storage system to VO users. Users use it to share their data or to allocate their required space

As seen in figure 1, GiSK acts as a common storage for all users in VO and gives the users a consistent view of the same set of files in the storage. VO users can perform generic file system operations such as file/directory management and ownership/permission management on it. They can also write their application codes using the API of the readily available repository system which offers storage space that are rarely limited.

GiSK addresses a few obstacles caused by the properties of the Grid when it virtualizes heterogeneous storage systems in the Grid as a single large storage system.

The main focuses of our system's design against the obstacles are as follows:

1) *Fine-grained access control for files and directories in Grid*
   The access control of files and directories is one of the most important functionality of generic file systems. To do this in detail, the file systems need to classify users of the system. The file system of GiSK gets this information by extending the semantic of subject name of the user's certificate in the Grid Security Infrastructure (GSI) and governs the fine-grained access control for files and directories with it. Therefore GiSK doesn't require an additional user management subsystem to classify users.
2) *Increasing the reliability of data under the autonomous administrative domains*
   The Grid integrates and coordinates resources that are not subject to centralized control, which is one of the three main grid checklists [7]. In these settings, substorage systems that compose GiSK appear dynamically according to their policy. The autonomous administration of the Grid makes some data stored in GiSK unavailable. For reliability, the volume controller of GiSK solves the problem using the "Lassoing a stake in order" strategy.

The rest of this paper is organized as follows: In Section II, we explain distributed file system, storage virtualization, and our motivation. Section III introduces the structure and subcomponents of GiSK. In section IV and section V, our system and its implementation are described in detail. The last section provides a summary and indicates our future research plans.

II. Background and Motivation

A. Distributed File System

A distributed file system stores files on one or more computers called servers, and makes them accessible to other computers called clients, where they appear as normal files. The servers can provide large storage space, which might be costly or impractical to supply to every client. There are many problems facing the design of a good distributed file system. Transporting many files over the network can easily create sluggish performance and latency [8]. The security of data is another important issue. Two further problems facing the design are related to failures. Often client computers are more reliable than the network connecting them and network failures can render a client useless. Similarly a server failure can be very unpleasant, since it can disable clients from accessing crucial information [9].

B. Storage Virtualization

Virtualization is a framework or methodology of dividing the resources of a computer into multiple execution environments by applying one or more concepts or technologies such as hardware and software partitioning, time-sharing, partial or complete machine simulation, emulation, and quality of service. In the context of storage systems, virtualization means an additional layer of software or hardware exists between the physical storage device and the user. In its simplest form, virtualization is just another form of aggregation of available block storage space that is then partitioned into logical volumes independent of the physical drives behind it

Logical volumes are mutually exclusive and dedicated to one host. Virtualization allows multiple storage controllers possibly heterogeneous and from multiple vendors to be connected and their storage to be aggregated.

Virtualization enables additional functions, including remote mirroring (making a copy of written data to a distant site, usually a disaster recovery site), caching (with additional memory for read or write caching), interoperability (allowing devices from two different vendors to be seen as a single device), global namespace (providing a single system view for either end users or system administrators), and scaling (allowing multiple devices to produce higher aggregate performance). [10]

C. Motivation

Until now, in the view of virtualization technology, the Grid has emphasized the virtualization for computing power and file access. Each of these is realized as a computational grid and a data grid. A computational grid [11] is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities. A data grid [12] is the next generation computing infrastructure providing intensive computation and analysis of shared large-scale databases, from hundreds of terabytes to petabytes, across widely distributed scientific communities. It enables access to geographically distributed computing power and storage facilities belonging to different institutions.

However, grid computing requires storage I/O and capacity that is typically orders of magnitude greater than most storage systems can provide today. This means that aggregate storage must be capable of maintaining a single image while being geographically distributed. That storage must also be able to scale both capacity and performance linearly up to tens of petabytes, exabytes, and even yottabytes in a single image. These are the reason that we need storage virtualization in the Grid.

Here we present GiSK, a VO repository system, as a realization of storage virtualization for the Grid, and address principle considerations in building a grid-based storage virtualization system.

III. GISK FRAMEWORK

This section describes the concept and structure of the GiSK framework. Subsection A provides a bird's eye view of the GiSK framework. Subsection B depicts main components, utilities, and API composing GiSK.

A. Overview

Current grid computing solutions typically employ file staging techniques to transfer files between user accounts in the absence of a common file system. File staging approaches require the user to explicitly specify the files that need to be transferred and are often not suitable for session-based or database-type applications. Also, as indicated earlier, current solutions don't provide virtualization for heterogeneous storage systems in the Grid. Not supporting virtualization may increase the complexity of physical storage devices and their administration and also escalate costs for managing storage assets.

GiSK is a repository system for VO that provides secure, unified, and enhanced access of VO's content and gives almost unlimited storage space to users that require a large amount of space to store their data.

GiSK is built within a collection of coordinated generic disk storages in the VO. Its capacity and performance may have changed in dynamics because the participating disks are controlled by their local policy. It gives a consistent view of the same set of files/directories to the VO users. It provides a fine-grained access control of files/directories, without having to operate specific user management functionalities, by utilizing the authentication mechanism of the GSI. It improves data reliability using the "Lassoing a stake in order" strategy. It provides API, which enables users to develop an application on the GiSK framework, and it facilitates its use and management with various utilities. Finally, it implements the Storage Resource Management (SRM) [13] specifications of the GGF, which make various storage management systems interoperable in Grid, and serves as a Grid-compliant storage system.

In the context of a GiSK user, GiSK provides the following:

1) He recognizes GiSK as highly available large storage space.
2) He is authenticated and his access privilege is restricted according to his certificate.
3) He creates, deletes, and modifies files or directories in the storage under his rights, and also configures a fine-grained access control for them.
4) He copies his local data to the storage for sharing in the VO and fetches shared data to his local storage.
5) He searches data he is interested in through various query options.
6) He utilizes the high volume storage by modifying his application using GiSK API.

In the context of OGSA, GiSK acts like the following:

1) Other OGSA services identify it as a service which serves the management functionalities for a large volume storage.
2) It takes advantage of the proxy delegation capability of GSI to control the access for files or directories.
3) They call it to get/put desired data.

B. GiSK Structure

GiSK is composed of three main components, GiSK utilities, and GiSK API. The main components are the GiSK File System (GiSK-FS), the GiSK Volume Controller (GiSK-VC), and the GiSK Volume (GiSK-V). The following figure 2 describes the structure of GiSK.

GiSK-V is a real repository of GiSK, and its space contribution depends on the service's local policy. It allocates a certain amount of local disk on its own machine to share space. The property of the space is specified in a configuration

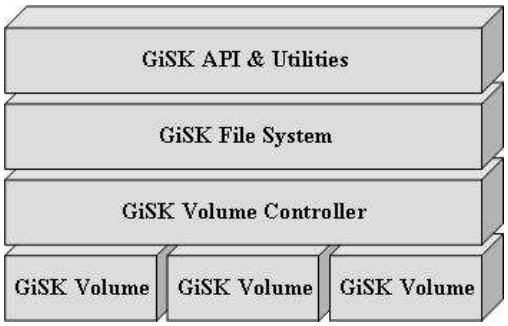

Fig. 2. The structure of GiSK. Each component communicates with each other using the SOAP mechanism

file of the service in the following format.

<gisk:volume path="/var/gisk" amount="10G"/>

Here, the path attribute represents a directory for space sharing and the amount attribute shows the maximum size of the space. It stores data in the unit of `Bricks`, which is a fixed size space for storing data, and its size is determined by the VO policy. A file may be stored over several `Bricks`. It is responsible for managing `Bricks` residing at the space.

GiSK-VC improves the flexibility of VO storage infrastructure by dynamically supervising the participation of GiSK volumes and effectively laying out data to the appropriate GiSK-V's. It accumulates the capacity of multiple GiSK volumes into a single storage pool, which is simpler to manage and helps increase utilization via central administration. It also governs the information for `Bricks` and `Cells`, pairs of Bricks (described later).

GiSK-FS abstracts the underlying complicated storage system. There are not only a bevy of volumes which provide actual space but also a great number of files and data. Therefore, we have GiSK-FS designed to simplify file and data management in the Grid storage system. Since the volumes themselves are managed by the local operating system, more specifically the file system, GiSK-FS has to consolidate diverse file systems across multiple platforms, such as Unix and Windows. From the user's view, GiSK-FS is a well-known gateway to the Grid storage system, and internal files and individual file systems are exposed to the user in a transparent manner through GiSK-FS.

GiSK-FS provides generic file system calls, such as open, read, write, close, rename, and unlink on the storage pool virtualized by GiSK-VC. It processes meta information for files and directories. It also handles access control of files and directories such as ownership and permission.

GiSK Utilities are a collection of tools to allow users to efficiently manage or utilize GiSK. These utilities provide similar functionalities as those for the Unix file system and are implemented as a client of a Web Service.

GiSK API offers all the building blocks necessary for users to easily write applications consistent with the system and lets users utilize GiSK using the SOAP-based RPC mechanism. It supports various languages including C, C++, and Java.

C. GiSK in Action

In the initialization of GiSK, GiSK-V organizes `Bricks` based on the size of allocated storage. Each GiSK-V registers to the GiSK-VC with its own `Brick` configuration. GiSK-VC makes up the `Brick` registry using these information. The registry is dynamically updated when a GiSK-V registers to or unregisters from GiSK-VC according to its policy, or it alters the `Brick` configuration due to the change in size of shared storage.

Users execute file operations through GiSK API in their codes. The API passes a user's request to GiSK-FS. GiSK-FS then authenticates the user, writes the metadata for the requested file, and consults GiSK-VC to get `Bricks` storing the content of the file. GiSK-VC selects the optimal `Bricks` for the file by referencing its `Brick` registry and returns them. GiSK-FS then uses the `Bricks` to operate files. The figure 3 shows the interaction among the GiSK components.

IV. Details of the GiSK Approach

This section goes into details about a few important issues of GiSK required to realize a storage virtualization system in the Grid.

A. Storage Management

Each GiSK-V divides its space into `Bricks` and then turns over the information to the GiSK-VC. GiSK-VC makes up a virtual storage pool synthesizing this information.

Whenever GiSK-VC receives the space allocation request from GiSK-FS, the controller selects a pair of `Bricks`, referred as a `Cell`, from the virtual pool. One is a primary `Brick` for storing data and the other is a secondary for a backup. This helps the system increase reliability.

In the case of organizing a `Cell`, we must choose a different GiSK-V from the GiSK-V with the primary `Brick` since replicating a `Brick` on a volume where the primary is stored is meaningless. Also, scattering the primary `Bricks` of a single file on multiple volumes allows striped transfer of the file to enhance performance.

A previous study, the "Chained declustering [14] [15]" strategy, suggests a solution for these problems in environments where the disks are organized into many identical disk arrays. However, we are in a situation where each GiSK volume shares different storage space depending on its local policy.

We introduce the "**Lassoing a stake in order**" strategy so that we avoid these problems. The steps of the strategy are as follows (see figure 4):

1) GiSK-FC requires space from GiSK-VC.
2) GiSK-VC arranges registered GiSK-V's in ascending order, according to the number of `Bricks`.
3) GiSK-VC elects the volume with the most `Bricks` as the "stake." The stake provides a `Brick` whenever a `Cell` is constructed. To make a `Cell`, GiSK-VC takes

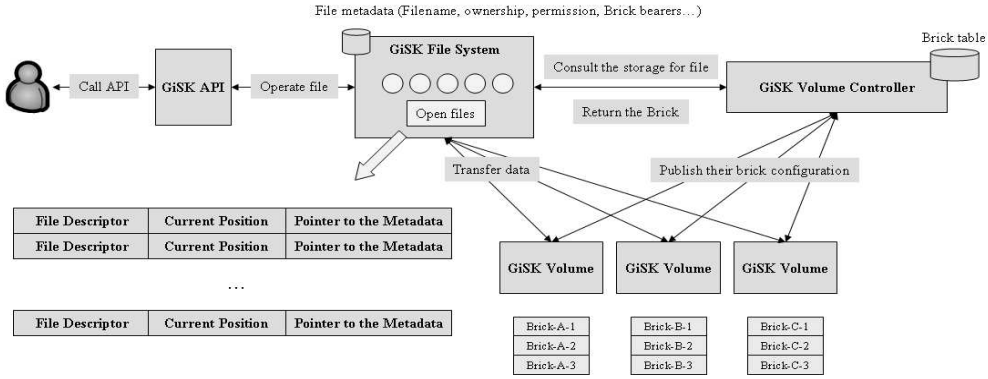

Fig. 3. A user accesses the GiSK system with the GiSK API. GiSK-FS has all the metadata information about the files so that it may handle the I/O requests and maintain the table for the open files as described. The current position is kept to record the user's read/write activity. As you can see, a single file can be stored in several GiSK-V's. If it is necessary to obtain a new `Brick`, GiSK-FS asks the GiSK-VC for an empty `Brick`.

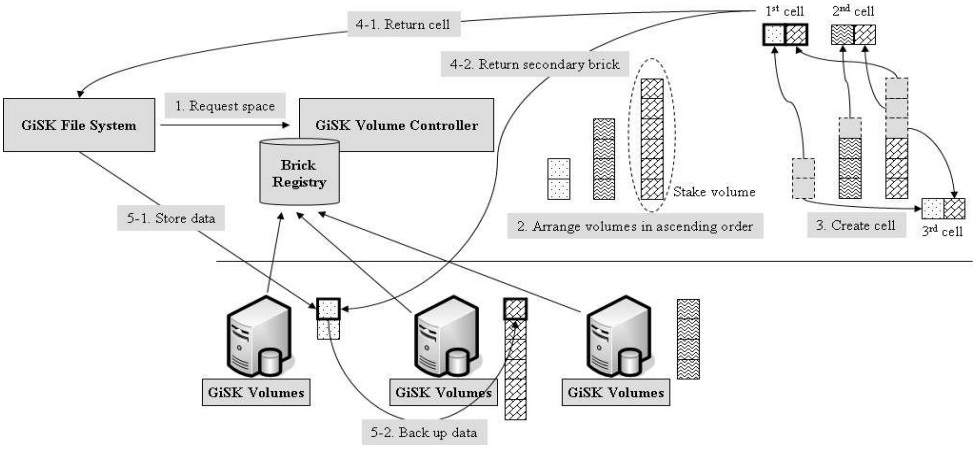

Fig. 4. Process of "**Lassoing a Stake in Order**" Strategy.

one `Brick` from the stake and binds it to a `Brick` from a non-stake. To make the next `Cell`, GiSK-VC takes the next `Brick` from the stake and binds it to a `Brick` from the next non-stake. When the stake volume runs out of `Bricks`, the second GiSK-V takes the responsibility of the stake.

4) A formed `Cell` is delivered to the GiSK-FS for file operations. GiSK-FS utilizes the primary `Brick` of the `Cell` to store data, whereas the secondary `Brick` is used if the primary fails. Also, the secondary `Brick`'s information, which is contained in the formed `Cell`, is passed to the GiSK-V with the primary `Brick`.
5) GiSK-FS writes data to the primary GiSK-V. The volume backs up its own data to the secondary GiSK-V using the secondary's information whenever its data is updated.

This approach assures that the replication of a `Brick` takes place at a different GiSK-V from the primary GiSK-V, and minimizes the number of `Bricks` that does not form a `Cell`. It also helps primary `Bricks` organizing a file to be distributed suitably. The weakness of this approach is that the failure of the stake GiSK-V decreases the reliability of primary `Bricks`.

B. File Management

Our goal is to hide the complicated underlying file management from the user under the Grid environment just as conventional file systems deal with the files on the disks. Though both of them look to the same goal - providing users with a consolidated view of the files - they have

several distinctions when dealing with the files. The obvious difference between the two systems is that the former must handle complex networking but the latter does not. This makes it harder for the GiSK system to handle the files in the aspects of reliability, scalability, etc. Our approach is to hide this complexity and provide the developers with familiar-looking operations - system calls. Through this, we not only can abstract the underlying storage system clearly but also can reduce the developers' burden that is concerned about making themselves familiar with new API's.

As can be seen from figure 3, a single file is stored in a series of fixed-sized units called `Bricks`. The `Bricks` are distributed over several GiSK-V's rather than on a single GiSK-V. This flexibility, though it may increase the managing cost of the `Bricks`, greatly improves the reliability of the files with the use of "Lassoing a stake in order" replication strategy and performance by realizing striped transfer.

Now we take a closer look at the GiSK operations one by one.

1) open: Conventional file systems maintain data structures for the open files on a per-open basis. Any subsequent requests on that file are processed via the file descriptor returned at the time of open. We follow this semantic on the open request for the GiSK file.

When a user tries to open a file through the GiSK API, the API first checks whether the name of the file is destined to the Grid or not. If it is identified as a Grid file, then the request is forwarded to the GiSK-FS by the GiSK API. Since GiSK-FS contains only the physical location of the files (remember that GiSK-FS does not have any physical files), GiSK-FS has to look up the passed file on its directory and instantiate an open-file handler, which includes the current position of the file. Open is just a means of making the GiSK-FS construct a corresponding data structure and prepare for the subsequent read/write request. During this setup, permission and ownership information should be matched.

Now GiSK-FS is ready to serve the actual read/write request. In order to allow the user to access the file in an efficient manner, GISK-FS assigns a unique identifier, file descriptor, to the open file and returns it to the requestor.

2) write: There are two options in designing the operations of our system, depicted in figure 3, that involve the movement of data depending on whether the data travels via GiSK-FS or not. The first approach lets the data directly travel from the user to the destination `Brick` bearer. This resembles the Direct Memory Access (DMA) of the modern I/O system and therefore, will exhibit a relatively short transfer time. However, its distributed architecture increases the degree of the complexity in managing the `Bricks`. The second approach makes the data stop over at the GISK-FS during the transfer. In this way, we can increase the manageability and reliability of the `Bricks` at the cost of doubling the data transfer.

Between these two methods, we have chosen the second approach as our data transfer method because our system is supposed to have numerous `Bricks`, which in turn makes managing the `Bricks` much more difficult.

Now let's go back to the write process. The user passes the file descriptor and the data to the GiSK-FS through the GiSK API. The GiSK-FS first has to check whether the requested region already exists or not. This will lead to two different behaviors. First, if it fails to find the requested region, the GiSK-FS asks the GiSK-VC to allocate an empty `Brick`. GiSK-VC returns the locations of a pair of `Bricks` to the GiSK-FS according to the local or global `Brick` allocation policy. Second, if the block already exists, GiSK-FS can retrieve the physical location of the requested region from the metadata information of the corresponding file.

Now the GiSK-FS has the physical location of the destination machine and the unique number of the `Brick`. GiSK-FS then hands out the given data to the selected machine.

3) read: When a user wishes to read data from a file, he specifies a file descriptor and destination buffer to the GiSK-FS through the GiSK API. Then the GiSK-FS computes the number of the `Bricks` that are needed by calculating the difference between the current position of the file and the length of the data that the user requests. GiSK-FS now actually gathers the `Bricks` from the `Brick` bearers and transfers them to the user. During this process, GiSK-FS caches the `Bricks` in the form of files. In order to do this, a cache manager is integrated into the GiSK-FS in a separate module. This module manages the `Bricks` cached by the GiSK-FS and retains a caching table according to the local policy.

4) seek: The inherent structure of the GiSK system offers a trivial method in processing the seek request. We have mentioned earlier that per-open data structure is instantiated and that it deals with the subsequent requests heading to the opened file. We can have the GiSK-FS handle the current position of the opened file since every request is passed through the GiSK-FS in a somewhat centralized manner.

5) close: Close frees the open file descriptor.

C. Access Control Management

Conventional file systems provide the protection of the file by controlling access. This is usually implemented by using the condensed access list which consists of owner, group, and others. Appropriate access privileges are assigned to each of them according to the protection level. These privileges include the right to read, write, and execute. Classifying the users into three categories is well-suited for Unix-like operating systems since their user management function is accomplished based on the user and the group identifier. We follow this approach because it has been used for several decades as a representative means to protect files and can be easily adopted, especially when the target system basically processes the requests per-user basis. Grid is one such system and so is the GiSK system.

Since the Grid deals with the authentication and authorization based on the GSI, there should be an appropriate GSI-integrated method to allow or restrict the user's access. The core functionality of the GSI is represented by its utilization of X.509 certificates and Public Key Infrastructure (PKI). We

have designed a secure file access mechanism based on the subject name of the certificate.

GiSK users have to be first authenticated by GiSK-FS to handle files and directories. GISK-FS utilizes the subject name, which identifies the person or object that the certificate represents, of a GSI certificate to manage the ownership and permission of files and directories. The subject name of a GSI certificate is composed of the following three components:

- CN: Represents "common name" and identifies this particular certificate.
- OU: Represents "organizational unit" and identifies this CA from other CA's.
- O: Represents "organization" and identifies the Grid.

GISK-FS extends the semantics of the above subject names and models ownership as follows (See figure 5).

- Owner ownership is mapped to the combination of O, OU and CN.
- Group ownership is mapped to the combination of O and OU.
- Others ownership are mapped to O.

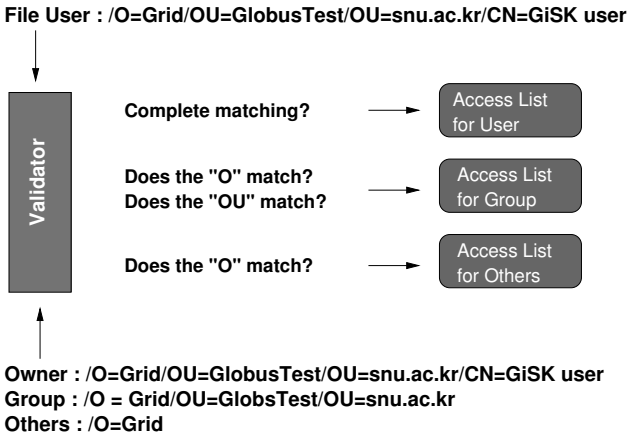

Fig. 5. How the permission is inspected

Figure 5 describes how the controlled access is integrated into the GiSK system. It is decided at the time of open whether a user has sufficient priviliges or not. The validator is responsible for this decision. The validator accepts two parameters, one is the ownership information of the file and the other is the subject name of the user. The validator checks whether the user is permitted to read or write the file according to the two supplied information and the match between them.

If the user's subject name completely matches the owner of the file, GiSK-FS regards the user as the owner of the file. A bit more relaxed match tells us that the user is a member of the group related to the file. Figure 5 shows an example that defines a group as a partial match. The *others* ownership is easily defined by a similar process. Once a user falls into one of the ownerships above, then the access privileges are easily determined since we can obtain the access control list from the file's metadata according to the ownership results from the previous match.

V. Implementation

We have developed the prototype of the GiSK framework based on GT4. GT4 is a realization of the OGSA requirements and a sort of de facto standard for the Grid community. It also includes a complete implementation of the WSRF specification, and we can program stateful Web Services using GT4.

In these setting, GiSK-FS and GiSK-VC are implemented by stateful web services with singleton resources, whereas GiSK-V is implemented by a stateful web service with multiple resources. GiSK API currently supports only the Java language.

Currently, GiSK-FS provides only the following basic file operations: gisk_open(), gisk_close(), gisk_read(), and gisk_write(). GiSK-VC doesn't yet simulate the full "Lassoing a stake in order" strategy and executes the strategy in a round-robin fashion. The brick size of GiSK-V is now 16MB. It is from a commercial SAN-based storage virtualization system such as the IBM storage virtualization system [16].

VI. Conclusion

Grid is emerging as a means of providing a consolidated view of distributed resources such as computing power and storage. A lot of work have been performed in order to establish a robust and concrete Grid infrastructure. However, most of the efforts have been made in constructing a computational grid, data grid, and access grid. Current works lack storage integration.

We borrowed the architecture of the virtualization from the storage virtualization system. It enables us to gather the magnetic disks scattered all around the Grid system and make use of them to provide a unified view of the underlying disk storage system. We have designed a secure, reliable, and scalable storage virtualization system for the Grid called GiSK. Basically, GiSK utilizes the GSI in order to control user access and is equipped with the "Lassoing a stake in order" strategy to improve reliability. In addition, GiSK follows the semantics of the conventional operating systems so that it might make operations and data structures clear and easy to understand.

We have developed a prototype of GiSK based on GT4. Currently, it is equipped with primitive operations and is running on simple configurations.

We try to find a mechanism that produces the appropriate `Brick` size considering the property or policy of VO. We plan to evaluate the "Lassoing a stake in order" strategy in diverse Grid environments. For implementation, we will implement missing functionalities and SRM interfaces.

Acknowledgment

The ICT at Seoul National University provides research facilities for this study.

References

[1] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "The physiology of the grid: An open grid services architecture for distributed systems integration," 2002, http://www.globus.org/research/papers/ogsa.pdf.

[2] K. Czajkowski, I. Foster, and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, 2nd ed. Morgan Kaufmann Publishers, 2004.
[3] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "Grid services for distributed systems integration," *IEEE Computers*, vol. 35, no. 6, pp. 37–46, 2002.
[4] I. Foster, N. R. Jennings, and C. Kesselman, "Brain meets brawn: Why grid and agents need each other," in *Third Internaional Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04)*, 2004.
[5] "The WS-Resource Framework," http://www.globus.org/wsrf.
[6] T. Clark, *Storage Virtualization: Technologies for Simplifying Data Storage and Management*. Addison Wesley Professional, 2005.
[7] I. Foster, "What is the grid? a three point checklist," *GridToday*, 2002.
[8] A.-J. Peters, P. Saiz, and P. Buncic, "AliEnFS - a linux file system for the AliEn grid services," in *CHEP03*, 2003.
[9] R. J. Figueiredo, N. H. Kapadia, and J. A. B. Fortes, "The PUNCH virtual file system: Seamless access to decentralized storage services in a computaional grid," in *Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC 01)*, 2001.
[10] E. Riedel, "Storage systems - not just a buch of disks anymore," *QUEUE*, pp. 32–41, 2003.
[11] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," *International Journal of Supercomputer Applications*, vol. 15, no. 3, pp. 200–222, 2001.
[12] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke, "The data grid: Towards an architecture for the distributed management and analysis of large scientific data sets," *Journal of Network and Computer Applications*, vol. 23, no. 3, pp. 187–200, 2000.
[13] "Grid storage management workgin group," https://forge.gridforum.org/projects/gsm-wg.
[14] H.-I. Hsiao and D. J. DeWitt, "Chained declustering: A new availability strategy for multiprocssor database machines," in *Proceedings of the Sixth International Conference on Data Engineering*. IEEE Computer Society, 1990.
[15] L. Golubchik, J. C. S. Lui, and R. R. Muntz, "Chained declustering: Load banalcing and robustness to skew and failures," in *Proceedings of the 2nd International Workshop on Research Issues in Data Engineering*, 1992.
[16] "IBM virtualization overview," http://www-1.ibm.com/servers/storage/software/virtualization.