Large-scale File System Design and Architecture

V. Dynda, P. Rydlo

This paper deals with design issues of a global file system, aiming to provide transparent data availability, security against loss and disclosure, and support for mobile and disconnected clients.

First, the paper surveys general challenges and requirements for large-scale file systems, and then the design of particular elementary parts of the proposed file system is presented. This includes the design of the raw system architecture, the design of dynamic file replication with appropriate data consistency, file location and data security.

Our proposed system is called Gaston, and will be referred further in the text under this name or its abbreviation GFS (Gaston File System).

Keywords: file system, replication schema, data protection, consistency, data locating.

1 Introduction

The amount of data stored in digital storage has been increasing at a very rapid rate. Since much of this data is of critical importance for its owners, it is necessary to ensure transparent data availability and to secure it against loss or disclosure. Till now, data backup has in many cases been performed manually, using various backup data storage types; in extreme cases the data has even been transformed into printed form. Unfortunately, these methods of data backup provide neither satisfactory protection against loss nor greater availability, and such a solution is clearly unsatisfactory. The probability of simultaneous hard drive failure and backup tape loss or damage remains high for many applications. Moreover, such backup is far from transparent for users. The cost of this approach is also much too high.

The Gaston File System is an experimental storage system that provides users with the standard services of ordinary file systems. Its major advantages are high data availability, support for mobile and disconnected users, and the ability to protect data from loss, damage or disclosure. One of the primary purposes of the system is to perform under many circumstances similar to existing LAN-based networked storage systems. These properties are to be achieved by involving thousands of computers spread across a large geographical area (continents or even the whole world) and connected by a computer network (Internet). High data availability and protection are managed by massive replication, and data protection in the generally insecure environment of the global network is achieved by suitable cryptographic mechanisms.

The GFS is still under development and is divided into several mutually connected areas. These are: the system architecture, data locating, shared data consistency, replication, caching, data security, authentication & authorization, request distribution and naming schema & structure. It is necessary to bind these research fields together into a single complex and consistent system specification useful for subsequent implementation and testing.

The essential architecture of the system is based on clients who deposit their data into the entire system and manipulate it, replica-managers managing stored (and replicated) data, and data servers that store data. Each replica-manager controls one or possibly more data servers for better performance and fault-tolerance. Clients access the data transparently using a specified interface. Since shared storage space is based on mutual reciprocity, client, replica-manager and data server can be installed on one machine, thus providing the system with shared resources in return for using its services.

An important means for protecting data in GFS is massive replication. Data is allowed to be replicated anytime/anywhere (even a copy in a client cache is considered to be a replica). To provide maximum performance, data location is of great importance. This is supported by separating data from its physical location (such a class of data is called nomadic data) and by replicas that can flow freely. The number and location of the replicas can vary in time, based both on client specification and actual usage patterns and network characteristics. These propositions are important for mobile client support, to deal with temporary network partitioning and especially to minimize communication costs.

One of most important tasks in the field of replication is to ensure the consistency of shared files. As the main aim of GFS is to achieve high availability, it is necessary to implement optimistic replica control protocol providing data availability even in a partitioned network. The protocol used in GFS is based on version vectors, transaction semantics and on automatic version conflict resolution. Conflict resolution is performed taking into account data type, or using the supplied application specific merge procedure. In order to ensure consistency it is also necessary to order update requests, which, unfortunately, cannot be performed globally on such a high number of replicas due to unacceptable time consumption. Hence ordering is specified by means of smaller variable groups of selected primary replica-managers, which eventually make information available to the other replica-managers. Because of primary replica-manager group variability, neither the system scalability nor fault-tolerance is negatively affected.

To improve GFS performance and decrease network usage costs, the system uses caching of data. In GFS whole files are cached to support mobile clients, who can be temporarily disconnected and therefore can use data even in this mode of operation. Since caching of data in distributed environments causes data consistency problems, cached data are seen as ordinary replicas and for this reason the replica management described above is also used for cached data.

One of the requirements on GFS is data security. As the system is open to many different clients, and as it uses Internet as its communication subsystem, it is exposed to many security threats. The system must protect the data not only when it is being exchanged, but also when it is stored at data servers. For these purposes well known and widely accepted cryptographic algorithms are used. For change detection of sensitive or non-encrypted data, digital signatures are used. Like almost every distributed system, GFS performs authentication of clients to check the user's identity, which is also based on cryptography. However, data is not only protected by cryptographic means. Every client who wants to perform any operation on the data must be authorized to perform such an operation. In this case access control lists are used, as in many other modern operating systems.

Since GFS is intended to provide users with a transparent file system with sharing capability, a global name space is used. This means that each user uses the same file name to access another's file. Moreover, the names of files are independent of the number of replicas and their locality, hence files in the system are transparently presented to the user. Another important benefit is that there is no need to remember file locations, because all replicas of one particular file are equivalent.

The design of the above mentioned challenging properties of GFS is further described below in several chapters dealing with the most important design issues.

2 Requirements to DFS

The proper design of a distributed file system should reflect several basic characteristics of the system, which arise from the essential features of a distributed environment and the typical data access pattern in DFSs. These include the following:

- *Read/write access*

A distributed file system usually stores data accessible in read and write mode (unlike WWW and other similar distributed services storing just read-only data objects). Furthermore, these data can be shared by more than one user. The system must efficiently support strong consistency models for shared objects that are often updated, which causes higher communication costs. The replica management should minimize these costs.

- *Dynamic access pattern*

The client usage patterns may change quickly and frequently. The replica management must maintain system stability while responding reasonably to these changes. The environment of a large-scale distributed file system, as wide as the Internet, contains many individual machines. The protocols and algorithms used must scale well with the number of replicas for each object and the total number of machines in the system. Any centralized authority or information source would quickly become a performance bottleneck.

- *Federal structure*

The replica managers of a large-scale distributed file system may reside in disjoint administrative domains, so the system must allow replica managers to make autonomous decisions. No replica manager can force others to perform any activity that is against their interest, and the protocols that are used must take into account the possibility of cooperation refusal by any party.

- *Unreliable environment*

It is not possible to eliminate all errors of individual components (communication lines, replica managers) of large-scale systems, so at least some replicas may not be available at any moment and the network may even become temporarily partitioned. Also in these cases the data must be as available as possible.

- *Insecure environment*

The replica managers, clients and network infrastructure cannot be trusted. To prevent the replica management system from becoming a security risk, it is necessary for all decisions to consider only authenticated information. The system must also prevent a small number of compromised replica managers from gaining control of the system.

3 System architecture

The Gaston file system consists of three main architecture components of the highest importance – *replica managers*, *clients* and *data servers*:

- *Replica managers*. These manage stored (and replicated) data and run data consistency and update protocols in order to be able to provide the correct data.
- *Clients*. These are the basic elements in the system. They deposit and manipulate the data in the system. If the client caches the data, it is at the same time the replica manager, considering cached data as ordinary replicas.
- *Data servers*, which physically store the data.

The elementary architecture is presented in Fig. 1.

The system supports three types of users – *ordinary users*, *dedicated servers* and *diskless terminals*:

- The *ordinary users* of the system are caching clients (thus also replica managers), who use the system and in turn provide the system with their resources. The data of other users can be replicated at these clients.

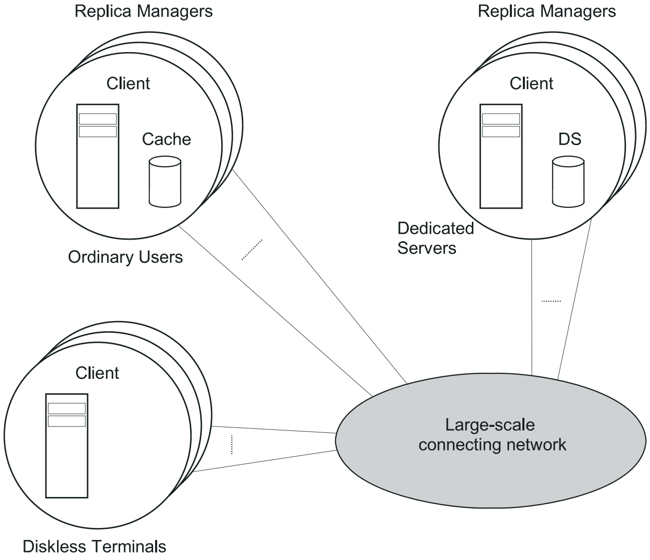

Fig. 1: Elementary system architecture

- The *dedicated servers* are machines intended just to store data. They act as relatively stable replica managers supposed to operate continuously.
- The *diskless terminals* are non-caching clients manipulating the data and communicating with replica managers. This is the case of small and possibly mobile and temporarily disconnected devices (cellular phones, small handheld computers, etc.).

4 Data consistency

The Gaston Global File System works under a transaction model of computation, which uses transactions for data manipulation. The system infers transactions by mapping the file operation call sequences to the transaction types. The goal of consistency and update protocols is to ensure one-copy serializability – data correctness. Since the means for achieving high data availability is massive data replication, it is not possible to assume a connected network of replica managers all the time. In such a large system, network partitioning will surely occur. Thus, the optimistic strategy for partitioned replica control is used. This protocol is based on precedence graphs [1].

A precedence graph models the necessary ordering between transactions and is used to check serializability across partitions. Each partition maintains a log file, from which read-sets, write-sets and serialization order can be deduced. The sequence of transactions in partition K_i is denoted as $T_1^i, T_2^i, \ldots, T_n^i$.

The nodes of the precedence graph represent transactions; the edges represent interactions between transactions. The first step in construction of the graph is to model interactions in each individual partition. Two types of edges are introduced:

1. *Data Dependency Edges* $T_j^i \rightarrow T_k^i$,
   if $\mathit{writeset}\left(T_j^i\right) \cap \mathit{readset}\left(T_k^i\right) \neq 0,\ j < k$.
2. *Precedence Edges* $T_j^i \rightarrow T_k^i$,
   if $\mathit{readset}\left(T_j^i\right) \cap \mathit{writeset}\left(T_k^i\right) \neq 0,\ j < k$.

Both types of edges demonstrate that the transaction processing order has influenced the computation result. The graph constructed in this way must be acyclic, since the orientation of an edge is always consistent with the serialization order, and within each partition serializability is always provided.

To complete the precedence graph (at reconnection), a conflict between partitions must be represented. The following type of edge is defined for this purpose:

Interference Edges $T_j^i \rightarrow T_k^\ell,\ i \neq \ell$,
if $\mathit{readset}\left(T_j^i\right) \cap \mathit{writeset}\left(T_k^\ell\right) \neq 0$.

An interference edge represents the dependency between transaction T_j^i that read the data in one partition and transaction T_k^ℓ that wrote the same data in another partition. This edge between T_j^i and T_k^ℓ indicates that T_j^i logically precedes T_k^ℓ since the value read by T_j^i could not be influenced by any update transaction in another partition. Thus, the interference edge signals read/write conflict; a pair of these edges indicates a write/write conflict.

If the resulting precedence graph is acyclic, then transactions from partitions can be represented by a single serial history that can be obtained by topologically sorting the graph. If the graph contains the cycles, then the computation is not serializable and detected inconsistencies are resolved by rolling back (undoing) transactions and their dependent transactions (connected by dependency edges) in the reverse order of execution until the resulting subgraph is acyclic. To merge partitions, the final value of each data object is then forwarded to other partitions. If the transaction cannot be rolled back, some compensating actions need to be performed to nullify the effect of that transaction.

In order to address the very high number of replicas, a kind of epidemic algorithm is used for update propagation among replicas that make the tentative commits of the transaction until the global stable commit is announced.

5 Replication

Data replication is an important means for achieving high data availability, fault tolerance and thus better data protection. To provide maximum performance, the data locality is of great importance. This is supported by separating data from its physical location (this kind of data is called nomadic data) and by replicas that can flow freely. The number and location of replicas can vary in time based both on client specification and actual usage patterns and network characteristics. The enhanced ADR algorithm [2] is used for dynamic replication scheme management.

The ADR algorithm constitutes the replication scheme that forms the tree subnetwork. The size and the location of this tree changes dynamically depending on the request locality and frequency. All requests for an object from clients are routed to the closest replica manager in the replication scheme (RS) along the shortest path. The following types of replica managers are defined:

- $\overline{\text{RS}}$ – *neighbor* – a replica manager that belongs to the replication scheme and has a neighbor that does not belong to RS,
- RS – *fringe* – a leaf replica manager of the subgraph induced by RS.

The replication scheme is modified through the following tests that are executed at the end of each predefined time period t:

- The *expansion test*. The test is executed by all $\overline{\text{RS}}$ – *neighbor* replica managers. Each $\overline{\text{RS}}$ – *neighbor* replica manager RM_i compares for each of its neighbors $\text{RM}_j \notin \text{RS}$ values of

$$\mathit{rcnt}_r^j(x_i) \text{ and } h_1 = \sum_{\substack{\text{RM}_k \in A^i \\ k \neq j}} \mathit{rcnt}_w^k(x_i),$$

- which is the total number of write requests received in the last time period t from RM_i itself or from a neighbor other than RM_j (A^i denotes a neighbor set of replica manager RM_i). If $\mathit{rcnt}_r^j(x_i) > h_1$, then replica manager RM_i sends to RM_j a copy of the file with an indication to save it. Thus RM_j joins RS and the replication scheme expands. The expansion test is successful when the (if) condition is satisfied for at least one neighbor of RM_i.

- The *contraction test*. The test is executed by all RS – *fringe* replica managers. Each RS – *fringe* replica manager RM_i compares for its only neighbor $RM_j \in RS$ the values of

$$rcnt_w^j(x_i) \text{ and } h_2 = \sum_{\substack{RM_k \in A^i \\ k \neq j}} rcnt_r^k(x_i)$$

– the total number of read requests received in the last time period t from RM_i itself or from a neighbor other than RM_j. If $rcnt_w^j(x_i) > h_2$, then RM_i requests permission from RM_j to leave a copy of the file. Thus replication scheme RS shrinks.

- The *switch test*. The test is executed by replica manager RM_i if RS = {RM_i} and the expansion test has failed. For each neighbor RM_j it compares the values of

$$rcnt^j(x_i) \text{ and } h_3 = \sum_{\substack{RM_k \in A^i \cap RM_i \\ k \neq j}} rcnt^k(x_i)$$

– the total number of all requests received in the last time period t from all replica managers apart from RM_j. If $rcnt^j(x_i) > h_3$, then RM_i sends a copy of the file to RM_j with an indication that RM_j becomes the new singleton replica manager in RS, and RM_i discards its copy. Thus the replica migrates towards the center of the request activity.

At the end of each time period t all $\overline{RS}$ – *neighbor* replica managers execute the expansion test and all RS - *fringe* replica managers execute the contraction test. Each replica manager that is both $\overline{RS}$ – *neighbor* and RS – *fringe* first executes the expansion test, and if it fails it executes the contraction test. A replica manager that does not have a neighbor in RS first executes the expansion test and if it fails it executes the switch test.

An example of the dynamic replication scheme is presented in Fig. 2.

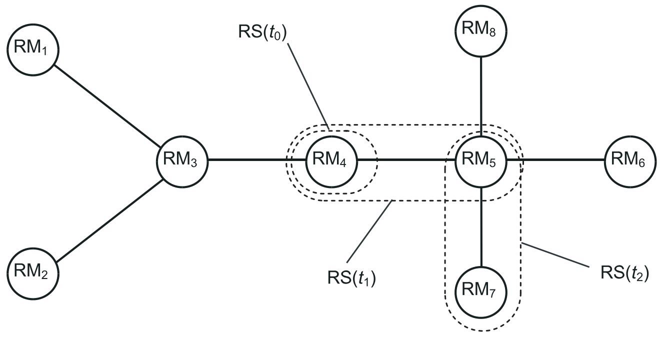

Fig. 2: Example of a dynamic replication scheme

The *initial replication scheme* $RS(t_0) = \{RM_4\}$.

Issued requests (in each time period *t*):

$RM_1 – RM_6$, RM_8: 4 read requests, 2 write requests
RM_7: 20 read requests, 12 write requests.

t_1: RM_4 (as an $\overline{RS}$ – neighbor of RM_3 and RM_5) executes the expansion test. Since the number of reads requested by RM_3 is 12 and the number of writes requested by RM_4 and RM_5 is 20, replica manager RM_3 does not enter the replication scheme. The number of reads requested by RM_5 is 32 and the number of writes requested by RM_4 and RM_3 is 8. Thus, $RS(t_1) = \{RM_4, RM_5\}$.

t_2: first, RM_4 performs the expansion test that fails, and then it executes the contraction test. It is successful since RM_4 receives 18 write requests from RM_5 and 16 read requests from RM_4 and RM_3. At the same time RM_5 performs the expansion test and successfully includes RM_7 in the replication scheme (the number of reads from RM_7 is 20 and the number of write requests from the other replica managers is 14). The resulting $RS(t_2)=\{RM_5, RM_7\}$.

t_3: RS stabilizes at $RS(t_3) = \{RM_5, RM_7\}$ and it will not change further.

Extension of the ADR algorithm to an arbitrary network topology is relatively straightforward using the network spanning tree. The modification that will be used in the Gaston System implementation concerns the connectivity of the replication scheme, and is able to create *pseudoreplicas* to address the problem of extremely distant clients.

6 File addressing

To locate data in the large system we use two distinct algorithms. The first one is a probabilistic algorithm that uses modified Bloom filters. It can be characterized as a fast algorithm that adheres to the principle of locality. Every node in the network keeps a filter (union) of objects that are accessible at this node and every edge also keeps filters of objects that are accessible using this edge. These filters (edge filters) define objects that are accessible at different distances (filter for distance one, two, etc.), thus forming a kind of routing table. An example of the algorithm is shown in Fig. 3.

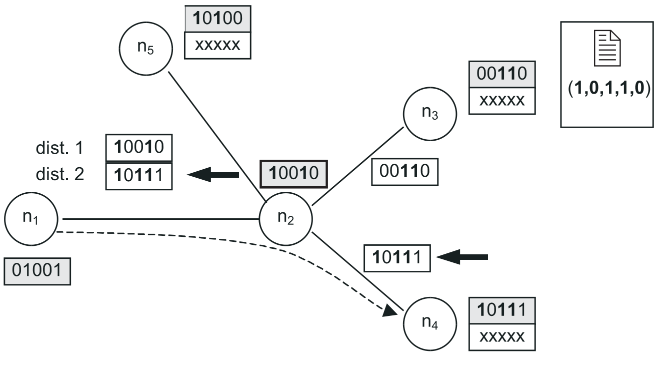

Fig. 3: Example of probabilistic file addressing

Here, a document characterized as (1,0,1,1,0) stored at node n_4 is being accessed from node n_1. The filters (the filter of the document and filters at nodes and edges) are compared to match and sequently this means, that the document is not stored at node n_1, but it can be reached on a node at distance 2. At node n_2 a comparison of the filters shows that the document is not stored at the node, but can be reached in distance 1 and in the direction to node n_4. At node n_4 we have perfect match, so the document can be reached. However, we must confirm this result with a complete scan of the documents stored at node n_4, due to the simple fact that the filters are unions.

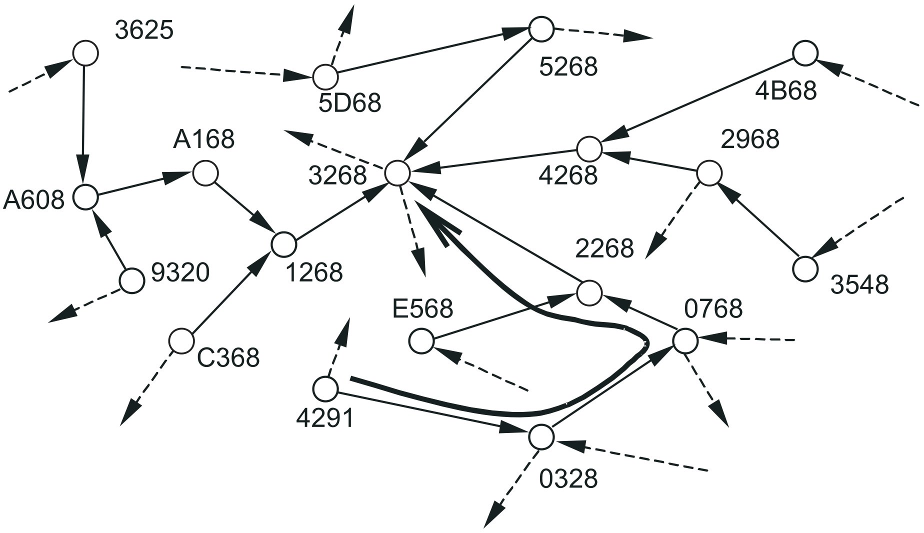

Fig. 4: Example of a deterministic file addressing mechanism

The second algorithm is used when the first one fails. It is a modified algorithm originally developed by Plaxton et al [3]. The algorithm uses embedded trees rooted in every node, which are assigned numbers that correspond to the object IDs. Routing to the node means routing to the information where physical data is stored. This mechanism is extended with information about the object that might be found during routing to the root, thus increasing fault tolerance. An example of this algorithm is shown in Fig. 4. The mechanism is similar to routing on an n-dimensional hypercube. Simply said, given a node from which we access data we compare particular parts of its ID and climb the tree in the direction corresponding to the matching parts. In the example, we access the node identified as 3268 from node 4291. We compare particular elements of the ID from right to left, thus passing through nodes 0328, 0768, 2268 and finally to node 3268. As can be seen, the nodes we are passing through become more similar to the node as we come closer to it.

7 Authorization and data protection

Distributed systems nowadays require far more protection of data than ever before. This is due to the simple fact that more people can easily access the distributed world than in the early days of the Internet. Protection using only well guarded servers is not sufficient, indeed it is not possible in the architecture that we present here. Data can be stored at potentially dangerous servers, which may not be guarded as necessary. Using them just as storage is sufficient from the view of the resistance against "physical" attacks. We propose to use cryptography to protect data content from restricted users (those who are not allowed to know the content, not only those who are not allowed to access the system). Data is therefore encrypted at the client nodes, which ensures that only the client knows the data and controls who can read the content. This approach not only provides good security, but also gives the system the property of good scalability. The system is designed to adapt to any appropriate cipher algorithm for this purpose. Of course, the system is open to several other improvements on the server and/or communication side of the system, where the underlying layers can use their own protection mechanisms (IPSEC in the communication layer, hw encryption of data at servers etc.)

Protection of new data is straightforward – data is encrypted by a user and then sent to the replica manager. Protection of changed data is a more complex problem, which is solved by a modified mechanism presented in [4]. A changed block of data is encrypted and attached to the end of the file, but the old block is also preserved to decrypt all data that consequently follow this changed block (we proposed to use the CBC mode of a cipher algorithm). This enables all the data of the file not to be reciphered, so that performance is not severely affected. As this solution preserves old blocks of data, the old versions of the file are also saved. The update situation is shown in Fig. 5.

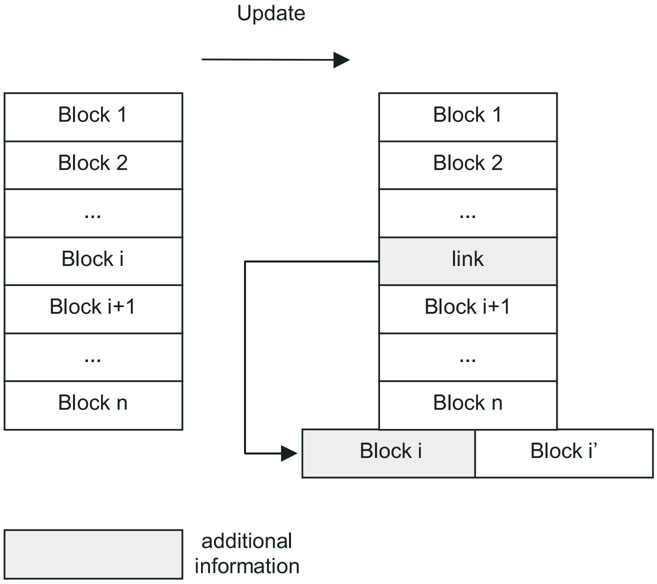

Fig. 5: Update of a file

A distributed file system is accessed by many users, so users' data must somehow be protected against access by other users. On the other hand, some people may want to publish their data to be read by others so some authorization to the data has to be introduced. In Gaston we use the following general principle [5].

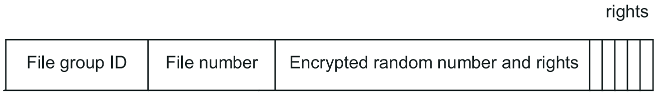

Fig. 6: Unique file identifier

The file identifier is extended with an encrypted random number and effective rights to the file for a given user. Unencrypted rights are attached for a user to know what rights he has been assigned at the moment. As he does not know cryptographic key to decrypt the encrypted part, he is unable to change the rights. The encryption key is known only to the issuer of the identifier and may be stored in a special part of the file system. A random number is used to prevent guessing of file identifiers. Changing the rights that are attached to the identifier has almost no effect on the resulting access. All user rights are stored in access control lists (ACL) that are associated with every file.

8 Conclusion

This paper has introduced the general architecture and key design issues of a large-scale file system. Our proposal of the Gaston file system is designed to provide high data availability, security and user mobility. These goals are achieved particularly by deploying file replication with reasonable data consistency and fast file location and also by proper cipher and authorization techniques that allow unencrypted data to be completely hidden from the replica managers.

Our design is based on the following basic characteristics, reflecting the essential requirements for DFSs:

- File replication providing high data availability and support for mobile and disconnected users.
- Transactional processing solving multiple read/write accesses and allowing achievement of data consistency corresponding to the requirements of DFSs.
- A dynamic replication scheme reacting to a frequently changing access pattern.
- Cryptography for securing user data and support for any appropriate cipher algorithm to protect data.
- Advanced authorization which helps in dealing with access rights.

Future work on the Gaston file system project will include advanced remote data modification techniques at distrusted replica managers, and also improving replica management in the field of load-dependent request distribution.

Symbols

A^i	Set of neighbor nodes to node n_i
ACL	Access Control List
ADR	Adaptive Data Replication
CBC	Cipher Block Chaining
DS	Data Server
DFS	Distributed File System
GFS	Gaston File System
K_i	Graph partition i
n_i	Node i
RM_i	Replica Manager i
$RS(t_i)$	Replication Scheme in the time t_i
$\overline{RS}(t_i)$	Graph complement to Replication Scheme RS in the time t_i
$rcnt_r^j(x_i)$	Read request count to x_i initiated by node n_j
$rcnt_w^j(x_i)$	Write request count to x_i initiated by node n_j
$readset\,(T_j^i)$	Set of data read by transaction T_j^i
t_i	Time i
T_j^i	Transaction with sequential number j performed in partition K_i
$writeset\,(T_j^i)$	Set of data written by transaction T_j^i
x_i	Data item stored at node n_i

References

[1] Davidson, S. B.: *An Optimistic Protocol for Partitioned Distributed Database Systems*. Ph.D. Thesis, Department of Electrical Engineering and Computer Science, Princeton University, Princeton, NJ, 1982

[2] Wolfson, O., Jajodia, S., Huang, Y.: *An adaptive replication algorithm*. ACM Trans. on Database Systems, 1997, Vol. 22, No. 2, pp. 255–314

[3] Plaxton, C., Rajaraman, R., Richa, A.: *Accessing nearby copies of replicated objects in a distributed environment*. In. Proc. of the 9th ACM Symp. on Parallel Algorithms and Architectures, 1997, pp. 311–320

[4] Kubiatovicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., Zhao, B.: *Oceanstore: An architecture for global-scale persistent storage*. In Proc. of the 9th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems, 2000

[5] Couloris, G., Dollimore, J., Kindberg, T.: *Distributed Systems: The Concepts and Design*. Addison-Wesley Pub. Co., 1994, ISBN 0-20162-433-8

Ing. Vladimír Dynda
e-mail: xdynda@fel.cvut.cz

Ing. Pavel Rydlo
e-mail: xrydlo@fel.cvut.cz

phone: +420 2 2492 3325
fax: +420 2 2492 3325

Department of Computer Science and Engineering
Czech Technical University in Prague
Faculty of Electrical Engineering
Karlovo náměstí 13
121 35 Praha 2, Czech Republic