M.I.T. Laboratory for Computer Science
Computer Systems Research Division

January 8, 1980
Request for Comments No. 182

DESIGN OF A DISTRIBUTED DATA STORAGE SYSTEM

by Liba Svobodova

This report introduces an ongoing project that involves a design and an implementation of a distributed data storage system for the users of the local network at the Laboratory for Computer Science at MIT.

Assumptions and goals

Our target is a reliable and secure data[1] storage system for a distributed environment consisting of many personal machines and one or more shared server machines. We are assuming distributed systems of the kind described in [SVOB 79]. The individual computers in such a system are highly autonomous. The data objects of any qualified user can be stored either on the storage device in his personal computer, or maintained by one of the server machines, but it is not possible to access directly the data objects in other than one's own personal machine.

The data storage server is not intended to be used directly by human users; it defines a functional interface for other programs, called here the clients. The components of the data storage system that actually store and retrieve data will be called storage servers. Each personal machine contains a local data storage server and communicates with one or more external data storage servers. The "personal" machines are dedicated to one user at a time, but they can be used by different people at different times. In such cases, a machine is personalized by changing the physical storage devices in the local server.

The most important reason for having a local server is availability: at least some work can be done even if the personal computer is completely detached from the network, or if the communication with other nodes in the network (in particular, the servers) is broken. In addition, the local storage servers provide more efficient access, both because of their physical proximity and because it is not necessary to use complex protocols to guarantee secure access and transfer. It can be also argued that the local servers provide more privacy, since the owner of the storage device is the only agent that has access to its contents.

The design goals for the data storage system include:

1. Uniform interface. The read and write operations provided to the data storage system clients should make the distributed nature of the system transparent, yet the clients should be able to control, through appropriate commands, where an object is to reside.

---

1. The terms "data" and "data object" are used throughout this memo as generic terms for stored information, that is, they include also programs and other control information.

2. Support for object oriented languages. The environment of the clients is assumed to be object-oriented; this means that the data storage system ought to be able to handle large quantities of rather small objects. Also, it should provide a realiable support for type-safe languages. However, this latter requirement should not make it impossible or difficult to use the data storage system from languages that do not embody such a strong notion of types.
3. Protection. It must be possible to specify and enforce access restrictions separately for each object or a group of objects maintained by an external server. Also, it should be possible to verify that the object returned to a client is of the type expected by the client.
4. Atomicity. It must be possible to write a group of objects in a single atomic operation without regards for the actual physical location of the individual objects.
5. Reliability. The data storage system must provide stable storage, at least for some classes of objects. Also, the atomicity of write operations must be satisfied in spite of failures of the servers and the communication lines.
6. Simplicity. The system should be sufficiently general to provide flexible and reliable service for a wide class of applications, yet as simple as possible. In particular, it is desirable that the shared server machines support only the minimum necessary set of functions.

Our data storage server will differ substantially from the existing file systems (servers) for similar environments that have been described in the literature [ISRA 78, SWIN 79, PAXT 79]:

1. it integrates the local storage server and the external servers,
2. it is object-oriented,
3. the degreee of reliability and protection provided for each individual object is controllable by the clients.

We will experiment in implementing the model of objects developed by Reed [REED 78]. In this model, a data object is thought of as a history of the states assumed by the object since its creation. Physically, each state is represented by a special immutable object called a version. The representation of the actual data object is an ordered list of the existing versions. A new version is created each time the object is updated. This model serves as the

basis for synchronization of accesses to shared data objects and for implementation of atomic actions that involve several distinct objects.

Stable storage

The storage system should quarantee that the data objects are stored reliably: neither the operations requested by the clients nor the internal operations (management functions) of the servers should ever damage them. In particular, the storage provided by the servers must survive failures of the hardware components used in the implementation of the servers, including failures of the storage devices. Such storage is called stable storage. Further, it is usually desired that an update operation on a data object in stable storage should either change the content of the object to the new desired value, or, if it fails, leave the object content as it was prior to the invocation of the update operation. That is, an object should never be left in an inconsistent state where the old value has been lost and the new one is incorrect. Thus the operations on stable storage should be atomic.

Since no physical device provides storage with these properties, the atomic stable storage must be implemented as an abstraction, using components with less desirable properties. To be able to use the techniques for implementation of atomic stable storage described in [LAMP 79], the following support is required:

i. It must be possible to detect that a physical copy of a stored object is incorrect.

ii. It is necessary to have at least two sets of physical storage (storage areas) that are completely independent from the point of view of failures. That is, a failure in one storage area should neither cause nor increase the probability of a failure in the other area.

Implementation of stable storage is quite expensive, both in terms of the amount of memory required and the processor and I/O time.[1] Thus atomic stable storage is usually assumed only for some special objects that are necessary for successful recovery from processor failures while other objects are only

---

1. The memory overhead is an extra copy of the object plus additional memory in the map or directory through which the physical location of the copies is determined. The processor and I/O overhead is the time needed to write the second copy; this has to be done every time the object is written. Read operations encounter this kind of overhead only if the primary copy is found to be incorrect, since then the second copy must be read.

periodically backed up in some offline storage. Then a recovery from device failures will restore an object to some, but not necessarily the latest, correct state. This means that some information may be lost. We postulate that some user data are so important that such a loss is intolerable and thus for the relevant data objects the current version, or possibly every version in the object's history must be stored in stable storage.

The following assumptions are made about the storage servers:

1. The external servers have the necessary capabilities to implement stable storage. These servers also can provide an extensive offline backup that includes the entire history of the (selected) objects maintained by the server.
2. The personal machines are expected to have quite limited amount of online storage and no offline storage. Also, the online devices may be less reliable than the devices used in the external servers (to keep the cost of the personal machines down). It is probably safe to say that even in the local server it will be possible to detect a bad copy, but it may not be possible to provide two sufficiently large storage areas that are failure-independent. Thus stable storage can be reasonably provided only for some special objects. Also, since the personal machines ordinarily will not have offline storage, local backup might be impractical.

Classes of stored data

The class of stored data is determined by several aspects: required protection, stability, availability and shareability. The data storage system will support the following classes of data:

1. Local and private local data. Data objects of this class are stored on the local storage device and are inaccessible to anybody but the owner, that is, the current user of the personal(ized) machine. The local storage server usually will not guarantee that such data will not be lost or damaged.
2. Private data in stable storage. Since the personal machines do not provide stable storage for client's objects, data of this kind have to be stored at one of the external server machines. The storage system has to ensure an owner-exclusive access to such objects.

3. Shareable data. Data objects that are to be directly accessible to several users must be stored at one of the external server machines. The storage system must support a secure and safe access to such objects by a selected changing set of users.
4. Public data. These are read-only data objects that are accessible to the entire user community. Such data can be maintained by the external servers less expensively than data of class 3.

If there is more than one external file server in the system, it is quite possible that each will have different reliability, protection and performance characteristics. Consequently, there may be several subclasses of classes 2 and 3 that reflect these distinctions in the file servers.

Private objects, both the local ones and those maintained by the external servers, can be shared at a higher level. That is, some user program which is the client of the data storage system can make them available to other users, but the storage system alone cannot do that. For the objects maintained by the external servers, the local storage device will be used as a cache. Alternatively, the system could support a class "private local data with external backup": the local server would maintain the data as in class 1, and an external server would maintain an encrypted copy. It is not clear whether this alternative view presents any advantage.

Protection issues

It is assumed that the data maintained by the external server will be stored and transfered to and from the server in an encrypted form. This should minimize the protection mechanisms that must be built into the external servers. A special component, the authentication server will be developed to control the distribution of the encryption keys.

The access to shareable data objects could be controlled primarily through the distribution of the encryption keys, however, such a scheme presents a number of problems. First, encryption alone is not sufficient to control write access. Thus the personal machine attempting to update an object in an external data storage server must present an additional piece of information to the server to ascertain its write rights. Alternatively, we could devise protocols where all write requests are supervised by the authentication server. In either case, the data storage server must be able to

authenticate the validity of the write requests. Second, since any personal machine could read any object stored at the external server, some information could easily leak out; for example, it would be possible to observe changes in the size of an object. An additional level of protection could be achieved by restricting the right to invoke an external data storage server. This could be accomplished by protocols such as those described by Needham and Schroeder; the authentication server would have to be invoked every time a connection is to be established between the personal machine and the data storage server.

The key distribution by the authentication server probably will be based on access control lists. But, since the encryption keys are distributed to the personal machines, the result is essentially a capability system. The capabilities (keys) can be revoked only by reencrypting an object under a different key, and modifying the access control list in the authentication server. The implications of long term encryption of shareable data are not fully understood yet; this is one area where more research is necessary.

Even if the data in the external server is stored in an encrypted form, the local server still provides greater privacy:

only the owner can invoke the server, and

only the owner can destroy an object maintained by the server.

Since it may be difficult to protect the local storage devices physically (that is, protect them from being "borrowed" by an unauthorized person), the private local data may have to be encrypted too. However, since only the owner can ever perform any operation on such objects, the key distribution/ revocation problem and the write access problem do not arise here.

Client interface

An interesting problem is how to divide the responsibility for data management betweeen the data storage system and its clients. The data storage system is not a data base system. Its function is merely to provide reliable storage and access to uninterpreted objects, vectors of bits. The main issue concerning the interface is the responsibility for maintaining integrity of typed objects. This problem has two aspects:

1. ensuring that updates of non-trivial data structures (structures that consist of several "storage" objects) are atomic,

ii. ensuring that only the proper type manager will be given access to an object retrieved from the data storage system.

The mechanisms for performing an update on several data objects as an atomic operation ought to be built into the data storage system, but the decision as to which objects are to be included in an atomic update must be made by the client. Thus the interface must include mechanisms through which the clients can specify such constraints.

The storage system has to interface with the environment where objects are strongly typed. Some magic conversion from an abstract type to the storage representation (and vice versa) is performed at the interface to the storage system. It is not clear how much run-time support is needed in this kind of distributed system to ensure that the objects retrieved by the storage system are indeed handled by the right type manager. Some protection mechanisms would have to be built into the personal machines. In particular, if the name used by the storage system to retrieve an object never appears outside of the type manager for that object, then the object can be retrieved only by that type manager. It may be sufficient to impose this restriction through compile-time type checking, or a mechanism such as the "object viewer" [LUNI 79] could be used at run time. A more difficult problem is how to ensure that the object retrieved by the storage system is of the correct type. This problem arises especially when retrieving a foreign object from an external server. One possibility is to seal objects by encrypting them under a key private to the type manager. The cost/benefit of this approach must be first investigated.

The client must be able to specify the class of the new object presented to (created by) the unified storage system, to change the class and to specify who has access (and what kind of access) to shareable data. In addition, it may be desirable to be able to specify that an object is to be replicated by the storage system for greater availability. With the exception of the operations that set/change the data class, the client does not have to supply the class; only the name of the data object should be required. For any operation, the client should not have to be concerned with the actual physical location of the object.

Finally, it is necessary to address the problem of naming. The servers do not understand human-oriented names: all objects are named by unique identifiers, which are strictly machine-oriented names. However, the client interface to the storage system ought to accept higher level names and look up

the appropriate uid, possibly by calling on an external name look-up server. (Alternatively, the name look-up server could be a client of the storage system.)

Internals of the storage system

The main components of the proposed data storage system are illustrated in Figure 1. The part contained in the personal computer has several levels. At the lowest level is the implementation of the local server. The next level contains all the necessary functions needed to find, update, recover, and protect typeless storage objects. Finally, there is the "user veneer" where typed objects are converted into typeless objects. Thus the implementation of the storage system interface is concentrated in the personal machine.

The external servers provide little more than stable storage. Their functional interface is very simple: it provides only operations for creating and deleting data objects, retrieving their values and updating (creating new versions of) data objects. The object must be specified by its uid. However, a novel storage organization and management will have to be developed to support efficiently Reed's model of objects.

One of the important research issues in this project is to explore the problems of supporting modularly composable atomic operations using Reed's update semantics. Thus the storage system will incorporate pseudo-times and possibilities, the mechanisms developed in Reed's thesis [REED 78]. These mechanisms are basic to the operation of the whole system; they have to be built into the servers, and properly set up by the storage system interface.

The average object in the proposed system is going to be quite small. Retrieving such objects from the external servers and protecting them individually could be quite expensive. In a system where protection is based almost entirely on encryption, support of each object as a completely independently controllable entity requires each object to be encrypted under a different key. The maintenance of such a large key data base and the associated access control lists is one problem. The other problem is that, unless the key for an external object is cached locally, the authentication server must be involved. For efficiency, it is desirable to group objects that are likely to be used together and that have the same access constraints. Such a set can be then encached at the local server with one authentication

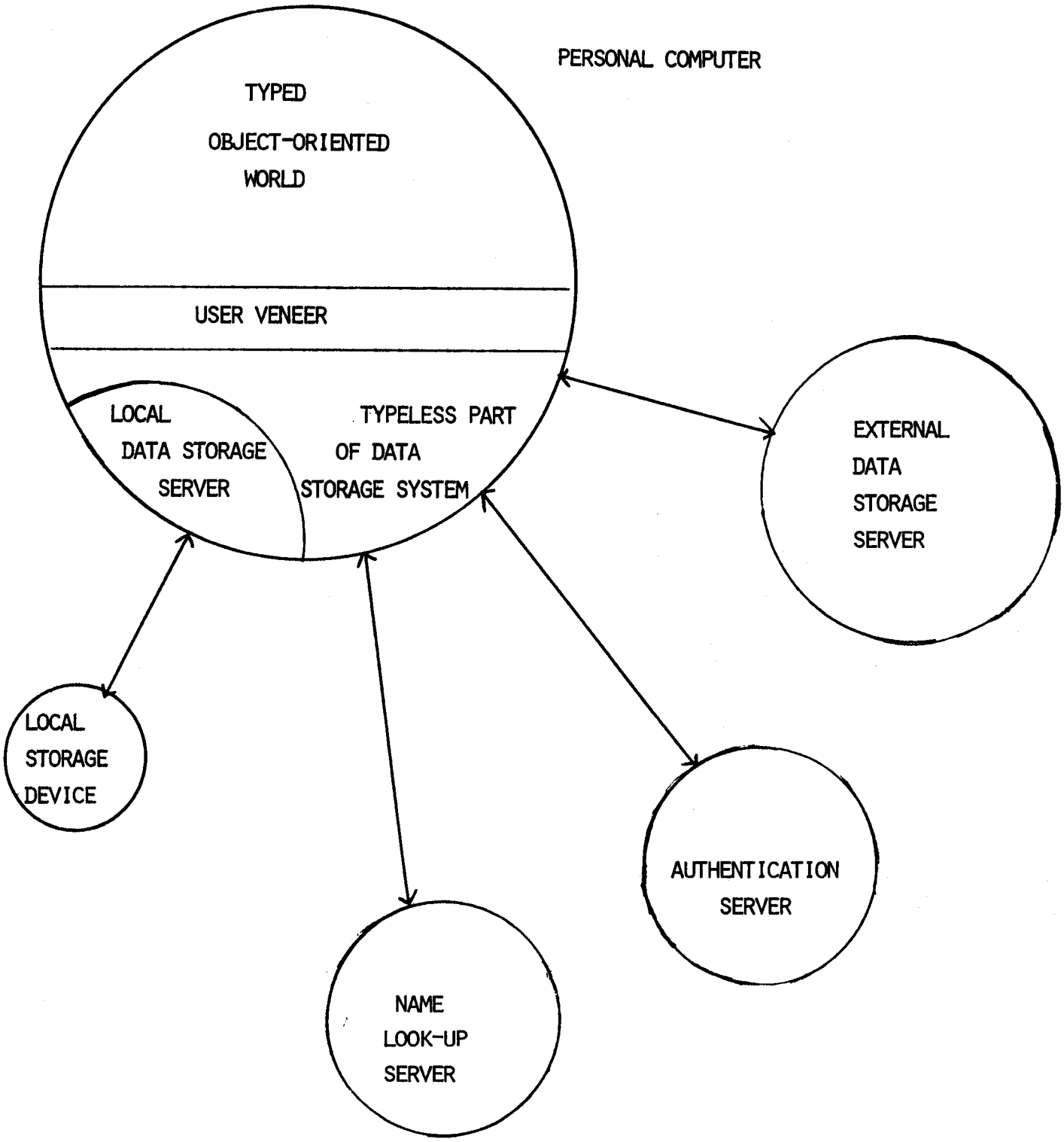

FIGURE 1: DATA STORAGE SYSTEM AND RELATED SERVERS

step and one retrieval step. This is a natural way to handle private stable data.

The system ought to be able to accomodate multiple external servers. As mentioned above, the external servers may have different performance, reliability and protection characteristics. Other reasons for supporting multiple servers are the extensibility (in terms of capacity) of the storage system and increased availability of stored data through replication on different servers. Since we already will support several different classes of data, experimenting with a system that utilizes the different characteristics of the multiple external servers probably would not provide any additional insight. However, experimentation with replicated shared objects is an interesting research project.

References

ISRA 78 Israel, J.E., Mitchell, J.G., Sturgis, H.E., "Separating Data from Function in a Distributed File System," Proc. of Second International Symposium on Operating Systems, IRIA, October 1978.

LAMP 79 Lampson, B.W., Sturgis, H.E., "Crash Recovery in a Distributed Data Storage System," XEROX Palo Alto Research Center, April 1979.

LUNI 79 Luniewski, A.W., "The Architecture of an Object Based Personal Computer," PhD. Thesis, Dept. of Electrical Engineering and Computer Science, M.I.T., December 1979.

PAXT 79 Paxton, W.H., "A Client-Based Transaction System to Maintain Data Integrity," Proc. of Seventh Symposium on Operating Systems Principles, December 1979, pp. 18-23.

REED 78 Reed, D.P., "Naming and Synchronization in a Decentralized Computer System," M.I.T. Laboratory for Computer Science Technical Report No.205, September 1978.

SVOB 79 Svobodova, L., Liskov, B., Clark, D., "Distributed Computer Systems: Stucture and Semantics," M.I.T. Laboratory for Computer Science Technical Report No.215, March 1979.

SWIN 79 Swinehart, D., McDaniel, G., Boggs, D., "WFS: A Simple File System for a Distributed Environment," Proc. of Seventh Symposium on Operating Systems Principles, December 1978, pp. 9-17.

Summary of the design issues

The design of the storage system can be divided into several parts. In addition to the design of the storage system proper, we have to consider two supplementary servers, the name look-up server and the authentication server.

1. Interface to the storage system from the clients

   issues:

   functions to be supported

   conversion from typed to untyped world

2. Internal organization of the private (local) server in the personal machines

   issues:

   encachement scheme for private objects

   encachement scheme for shareable objects

   relationship of the local data storage server to the local virtual memory

   storage management (storage used by the locally maintained data objects vs cache)

3. Internal organization of the external data storage servers

   issues:

   implementation of stable storage (type of storage devices to be used)

   mechanisms for synchronization and atomic update of groups of objects

   storage management

   replicating objects on several servers

4. Interface to the external data storage servers

   issues:

   connections to the external servers from the personal machines

   grouping objects for encachement and easier key management

   control of object creation/deletion/update

object naming

object sizes

objects that contain both private and shareable components

5. Interface to the authentication server

issues:

control of key distributions

the role of the authentication server in updating shareable objects

the role of the authentication server in the encachement protocols

6. Internal organization of the authentication server

issues:

maintenance and protection of the key data base

7. The name look-up server

issues:

interface to the data storage system

internal organization

relationship with the authentication server