

# Object Store Based SAN File Systems

**J. Satran**

*Julian\_Satran@il.ibm.com*

**A. Teperman**

*teperman@il.ibm.com*

**IBM Labs, Haifa University, Mount Carmel, Haifa 31905, Israel**

## Abstract

*SAN file systems today allow clients direct access to block devices for data storage and retrieval without going through a server. This however poses new challenges to file system designers such as security, scalability and management. The newly developed Object Stores (Obs) [5][6][9] enable applications to create and delete objects and to write and read byte ranges to/from objects. Obs provide space management abstraction, late binding, security, safe writes and other capabilities. Building a SAN file system using Obs as storage devices alleviates the challenges mentioned above. In this article we briefly detail these challenges and describe how Obs help in solving them. We then describe zFS, a scalable distributed file system which uses Obs.*

## 1 Introduction

The advent of high speed networks enables hooking up many storage devices and many computers together in what is called a *Storage Area Network (SAN)*. In such an environment clients interact directly with the storage devices for data storage and retrieval without going through a server. This direct interaction and the high speed network enables building a high performance distributed file system; thus such systems are also referred to as *SAN file systems*. For such a system to work properly, a coordinating agent is required to coordinate clients' access to the shared data. This agent is usually called the *Meta Data Server*.

In the past where SANs were being used in private Fibre Channel networks to emulate direct attached storage (i.e., no sharing), there was no real problem of security, scalability and management. The reason is that the file system was private thus the clients were trusted, and the file system itself was relatively small, so scalability and management were not a big problem. Once SANs started to be used as enterprise and Wide Area Network (WAN) file system allowing all clients direct access to the storage devices without server mediation, these issues became acute.

Since clients can directly access storage devices to store and retrieve data, special care must be taken to provide security, so that one client will not accidentally corrupt the data of another or read its data. Furthermore, storage allocation is a problem; in case a client needs more space not available on the disk, how and who is responsible for allocating more space on another storage device. End-to-end

management at a meaningful semantic level is also a problem.

The newly introduced Object Stores [5] [6] [9] are developed to address these issues. Building a SAN file system using object stores as storage devices overcomes these problems and provides us with a scalable secure distributed file system which is comparatively easy to manage.

In the next section we describe the components of a SAN file system. Section 3 describes the relevant characteristics of Obs followed by section 4 which describes how object stores can be used to build SAN file systems free of the problems mentioned earlier. A description of *zFS*, a scalable distributed file system using object stores, follows in section 5. Section 6 describes two file systems which use Object Stores followed by summary remarks in section 7.

## 2 SAN File System Components

Figure 1 shows a typical configuration of SAN file system [1]. The configuration consists of a set of *hosts* (also referred to as *nodes*), a set of storage devices and networks that interconnect them. The configuration consists of the following components:

- A set of hosts/clients which read or generate data on the file system. These can be heterogeneous hosts running different operating systems.
- A cluster of Meta Data Servers to coordinate clients' access to the file system. There must be at least one metadata server, but for high availability several should be setup.
- A set of storage devices. These devices can be block devices, or a more elaborated storage sub-system which provides RAID virtualization over physical disks. In the remainder of the article we use the term *Logical Unit (LU)* to refer to such storage volumes.
- A network which connects the clients, metadata server and the block devices. This can be either Fibre Channel, iSCSI or IP network.
- One or more *administrative consoles* that control the metadata servers.
- An IP network connecting the administrative console to clients and metadata servers. Note that the two networks can be merged into one.

Although SAN file system is a distributed file system, it provides applications with a *single system image* of the file system. In other words, data consistency for applications

Protection is always needed when there is shared access to data. It is intended to address buggy clients, administrative errors, etc. One example of why protection is needed is that

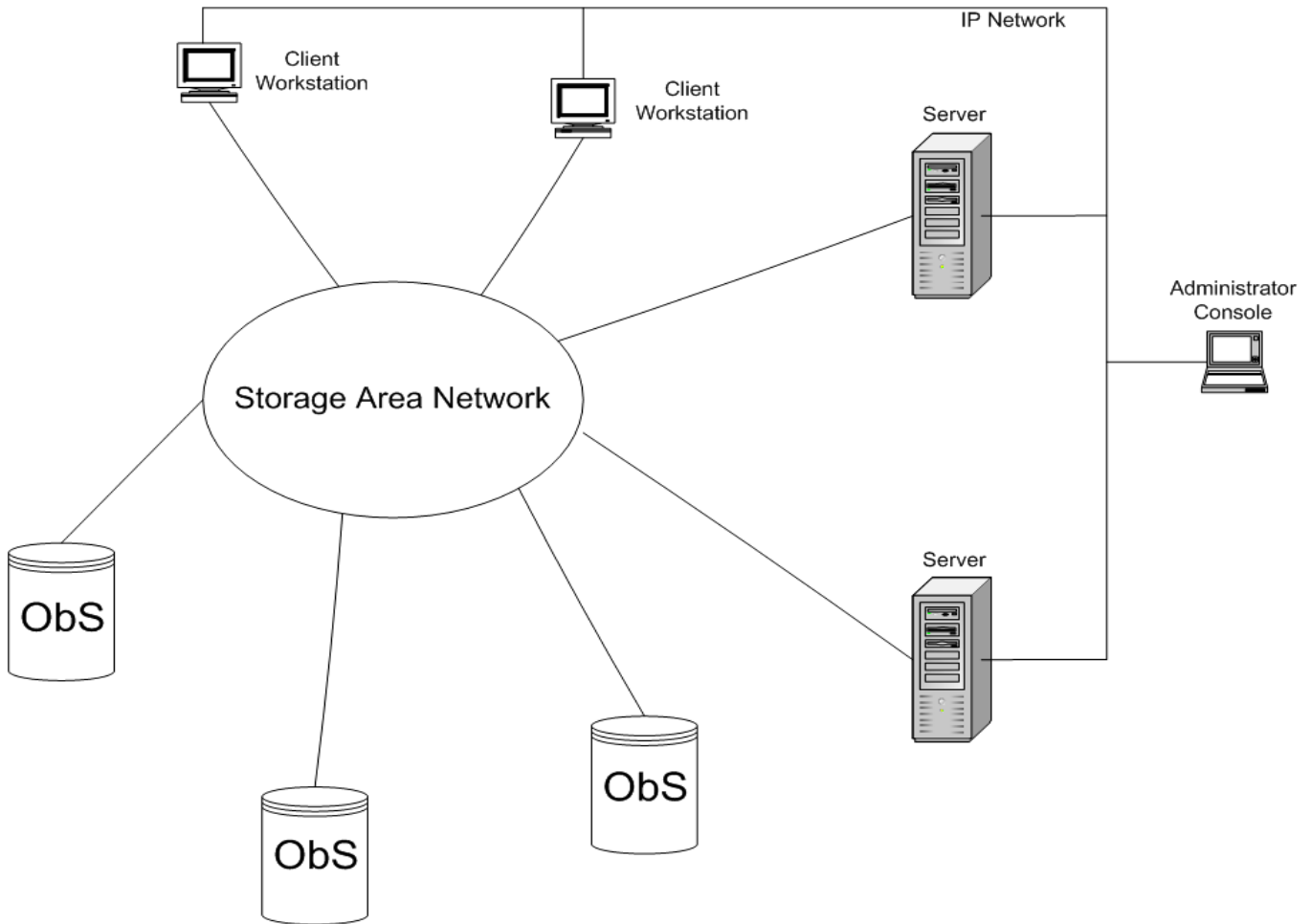


Figure 1. SAN File System Components

running on different nodes and accessing the same file is the same as all applications running on a single node accessing the same file on a local file system.

### 3 Object Storage

As mentioned before, SAN filesystems promise to improve storage access performance by providing non-mediated, shared access to storage. In a SAN Filesystem the data and control paths are separated. But the use of Block Storage in SAN's raises several issues.

The most significant issue raised is probably security on a SAN. This issue exists even on closed networks but it will be exacerbated by the expected adoption of IP-based storage.

In discussing SAN security we find it useful to distinguish between two concepts: security and protection.

if an administrator incorrectly configures Logical Unit Number (LUN) masking<sup>1</sup>, a Windows NT client that discovers a Logical Unit (LU) will assume it owns the LU, writing a signature on the LU, thereby causing a data integrity problem. Protection, as a defense against errors is thus needed even if we have a completely secured and trusted infrastructure.

Security goes beyond protection in that it addresses intentional attempts at unauthorized access. Security is thus essential if the infrastructure, (e.g., storage network), is untrusted.

---

<sup>1</sup> *Masking* is a common way of protecting whole disk units by making them *invisible* to clients.

Today, only minimal and very coarse grain support for security in a storage area network environment exists. In general, one must assume that the storage clients are trusted and the mechanisms that do exist such as zoning, LUN masking, etc., are hard to use and related to the physical structure of the storage. Thus, at best it is possible to provide all or nothing access to a LU for a given host.

Doing significantly better than this very coarse level of access control in the context of today's block storage devices, is probably not practical. There are too many actively used blocks to provide block-level security and extent-level security as proposed in [2] is foreign to the way higher level software (in this case file systems) are organized to provide access control.

For block-level security the storage unit cannot effectively take part in enforcing security decisions since it does not know which blocks need to have the same security attributes, i.e., the storage unit does not know which blocks are related. It would, thus, be necessary to tell the storage unit who is allowed to access each block which clearly would not make sense from the perspective of performance. For extent-level security the coordination required between a managing entity, client and the storage controller would be extensive.

It is for this reason that most of the research on object storage has been driven by the need to provide SAN security [3].

A second issue that arises when trying to fully leverage a SAN is scalability. Scalability is not an issue unless hosts share access to volumes. However, shared access is one of the touted benefits of a SAN filesystem. Shared access requires coordination, and coordination can lead to scalability problems. For example, file systems must coordinate allocation of blocks to files and for shared read-write access clients must coordinate usage of data blocks with other clients.

In SAN file systems built upon block storage, space allocation is managed by the metadata server, typically in concert with smart client involvement. By having the metadata server run on a cluster and partitioning responsibility between the nodes of the cluster, a good degree of scalability can be achieved. However, there are limits. This coordination can incur several costs including false contention between clients allocating space from different logical units and additional communication. In addition, since a metadata server runs on a traditional computer platform, i.e., one without a non-volatile RAM, there is either additional overhead to harden metadata updates, or a risk of (meta) data loss; since storage control units (typically) have some form of non-volatile RAM (e.g., to support fast writes) this support can be leveraged to harden metadata if the control unit was to perform space allocation.

Unlike a traditional block-oriented control unit which provides access to data organized as an array of unrelated blocks, an object store allows access to data via storage-

objects. A storage-object is a virtual entity that groups data considered by a client to be related. It is similar to a byte-stream file in a flat file-system with a conceptually unlimited size. Space for a storage-object is allocated by the object store control unit on demand; in other words, sparse allocation is supported.

The collection of storage-objects, i.e., an object store, forms what is essentially a flat file system. There is no name space -- just a flat ID space. The object store provides security enforcement for access to the storage-objects it contains but it does not provide security management, i.e., it is not the role of the object store to determine who is allowed to access an object -- it is the role of the object store to enforce access rights determined by some external security administrator.

While different proposals for object storage vary in the details of the functions they provide, in almost all proposals an object store provides (at least) the following basic functionality:

- Create an object
- Delete an object
- Read from or write to a byte range within an object
- Format
- Get object storage info, ...

An object store provides a level of virtualization and aggregation; more significantly it provides data path security. Thus it is only natural for an object store to be in the data path. A good place to realize an object store is on a control unit.

An object store control unit is an element of a scalable, networked storage infrastructure; each such control unit may export multiple object stores, just as a traditional block control unit exports multiple logical units.

As described above, an object store is a paradigm shift that places more intelligence in the control unit and aims to address issues of security, scalability and management.

An object store addresses security by securing all operations with a credential. Simply providing a credential on each operation, even if the credential is not cryptographically protected, provides protection since it is not realistically possible to accidentally present a valid credential for an operation. To provide security, however, some form of cryptographic protection on the credential is required.

The goals of object store security are to provide increased protection/security at the level of objects rather than whole LUs, allow non-trusted hosts to sit in the SAN and allow shared access to storage without giving hosts access to all data on volume. In addition, since allocation metadata is not directly processed by hosts, an additional level of protection is provided by the cooperation of:

1. A security administrator -- a process that runs outside of the object store and which is responsible for authenticating users, authorizing their access and generating the credential
2. An object store client which must request the credential from the administrator and pass this credential to the object store on all operations
3. The object store itself which is responsible for validating the credential presented by the client. The metadata server in a SAN filesystem in a "natural" residence for a security administration component.

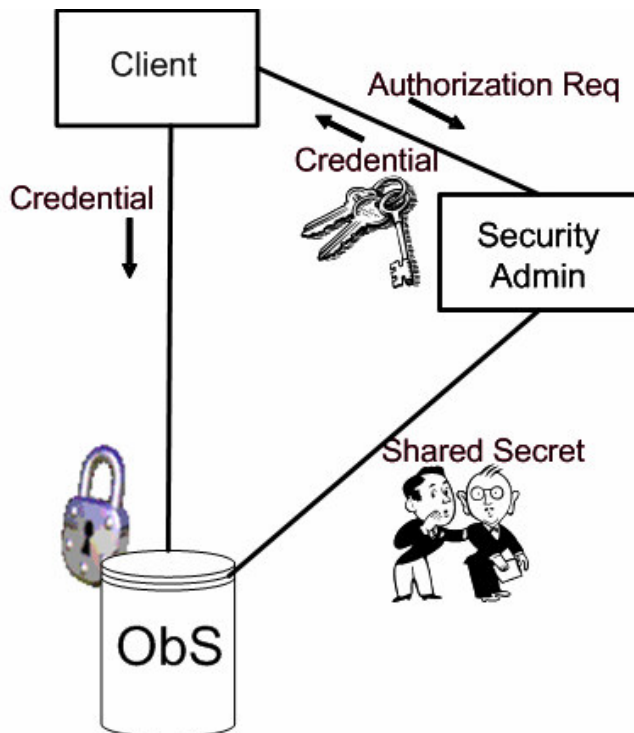


Figure 2. Object Store Security

Object stores improve scalability by distributing space allocation among storage controllers rather than doing this at the level of a metadata server. This means that no extra messages are required to coordinate allocation and there is no false contention for allocation to different object stores. In addition, since metadata recoverability is the storage controller's responsibility, we can leverage the non-volatile RAM typically found in a control unit.

#### 4 ObS Based SAN File Systems

Using Object Storage to build a SAN filesystem enhances considerably the filesystem scalability – and some filesystem builders have chosen Object Storage as their building block mainly because of the enhanced scalability ([4]).

Using object storage also considerably enhances security in a shared environment.

Building a file system based on Object Storage however raises a new set of challenges.

Object Storage is not built to support transactional operations. Even operations on a single object are not guaranteed to be atomic and stable. A write operation on an object might end with the object containing neither the old content nor the new content. Object Storage makes only very limited guarantees regarding its operations. Even more so operations that involve more than one object do not carry any guarantees.

All modern file systems however provide strong guarantees on some operations (e.g., operations on directories) and recovery from many failures is very efficient.

When using conventional file system operations in conjunction with an appropriate locking mechanism, users may provide atomicity guarantees to application data. When building an Object Store based file system we are faced with the challenge of providing this type of mechanisms in a form that keeps the scaling promise of Object Storage.

Most of the local and distributed filesystems built based on block-storage have developed block based caching mechanisms. Extending caching to support Object Storage and make effective use of it is another challenge that has to be taken by the Object Storage builder. In addition the security mechanisms employed to insure data integrity on its way from storage to clients could possibly be extended to enable cooperative caching. For some classes of applications using the collective caching capability of a large number of clients might result in high gains in performance.

With the extreme scaling enabled by Object Storage, active file management will have to be distributed to a larger number of machines. A file-system built on Object Storage may be able to use distribution techniques that are more efficient than the conventional clustering software and need no or very little group communication support. A Serverless structure will go a long way in improving file-system scalability.

#### 5 zFS: An ObS Based Distributed File System

zFS was designed to be a scalable distributed file system by separating storage-space management from file management. Storage-space management is carried out using ObSs, and file management is distributed over a set of cooperative machines [8]. These machines are commodity, off-the-shelf components such as PCs, running existing operating systems and high speed networks. While the current version of zFS does not address the issues of deployment policies and security management (access control), we believe that they can be distributed similarly to the way simple file management is distrusted today, and we plan to investigate these issues in the next version of zFS.

Files and directories in *zFS* are mapped to objects on object stores. The ObS does not distinguish between a file and a directory. Each file and directory in the file system is identified by a *file pointer*. The file pointer is a tuple  $\langle obs-id, oid \rangle$  where the *obs-id* identifies the object store where the object resides and the *oid* identifies the object within that object store.

Figure 3 shows the overall architecture of *zFS*.

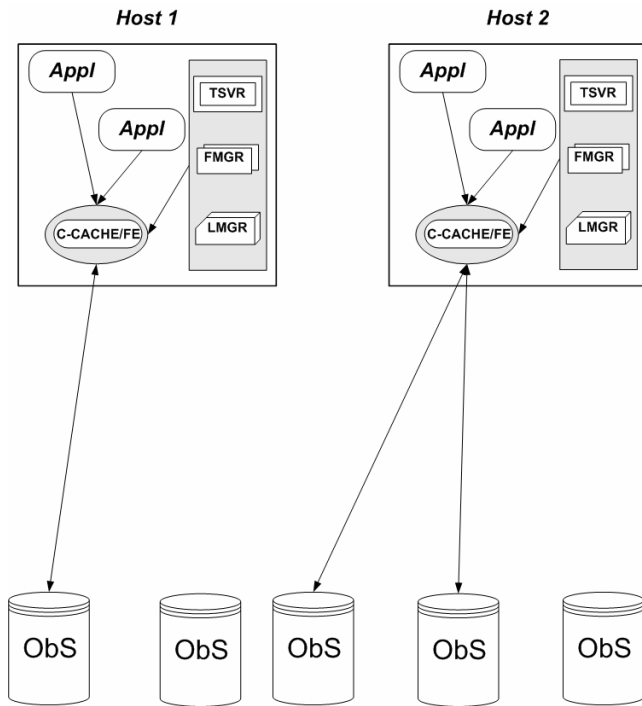


Figure 3. Overall architecture of *zFS*<sup>2</sup>.

All metadata operations: create file, delete file, create directory etc., are distributed transactions spanning several object stores. For example, create file may involve two object stores. One which holds the directory in which the file is created and other is the object store where the file itself is created. To ensure consistency in the presence of failures, such metadata operations are executed as distributed transaction by the *Transaction Server (TSVR)* component.

All storage allocation and management in *zFS* is delegated to the Object Store devices. When a file is created and written to, the data blocks are sent to the ObS, which allocates space on the physical disk and writes the data on the allocated space. Usually, this is done asynchronously for performance reasons.

<sup>2</sup> Although the figure shows *TSVR*, *FMGR* and *LMGR* running on each host, this is an extreme case. The other extreme is where non of the above runs on a host; only *C-CACHE/FE* must run on a host for it to participate in *zFS*. For details see [8].

The two most prominent features of *zFS* are its cooperative cache and distributed transactions.

The cooperative cache of *zFS* (*C-CACHE*) integrates the memory of all participating machines into one coherent cache. Therefore, instead of going to the ObS for a block of data which already resides in the memory of one or more of the participating machines, the *zFS* cooperative cache retrieves the data block from the memory of another machine.

The *zFS* front-end (*FE*) runs on every node participating in *zFS*. It presents the user with a standard file system API and provides access to *zFS* files and directories.

To ensure data integrity, file systems use one form or another of a locking mechanism. *zFS* use leases rather than locks. The management of leases is split between two components: the lease manager (*LMGR*) and the file manager (*FMGR*). The lease manager acquires an ObS lease from the ObS and grants exclusive file leases on whole files to the file managers. The file manager, after getting the exclusive file lease, manages the range leases which it grants the front-ends. When the front-end reads or writes data blocks from/to the ObS, it uses the range lease which is verified by the ObS. The file managers monitors each read lease and write lease it grants, and keeps track of where each file's blocks and leases reside in an internal data structure. This information is used by the cooperative caching algorithm as described below. The file manager also handles lease conflicts and issues the downgrade/revoke commands accordingly.

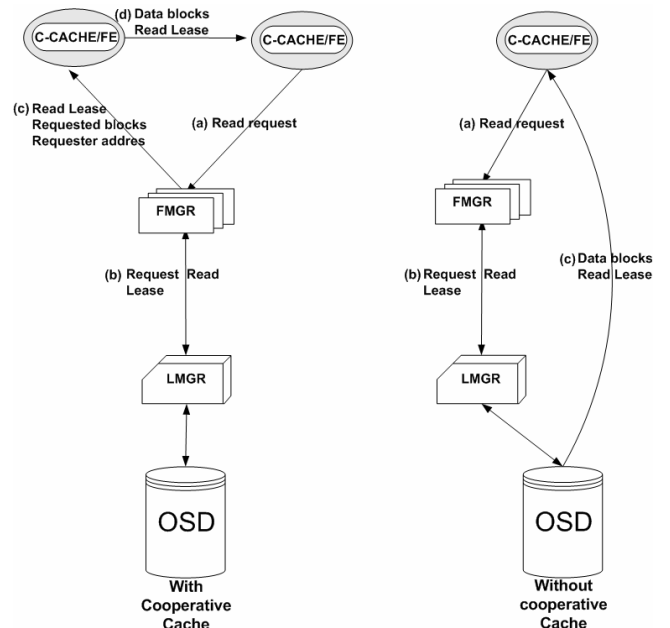


Figure 4. Read request with/without Cooperative Cache

Figure 4 shows the flow of control for a read operation in case the requested data resides in the memory of another

host, (which invokes the cooperative cache mechanism), and in case the data is not cached in the local memory of another host.

Each opened file in *zFS* is managed by a single file manager assigned to it when the file is opened. The set of all currently active file managers manage all opened *zFS* files. Initially, no file has an associated file manager. The first machine to perform an `open()` on file *F* creates a *local* instance of the file manager for that file. Henceforth, and until that file manager is shut down, each lease request for any part of the file *F*, and from any participating machine is handled by this file manager<sup>3</sup>.

Building *zFS* from the components described above is expected to achieve high performance and scalability due to the following reasons:

- Separation of storage from file management  
Caching and metadata management are done on a different machine from the one storing the data, the ObS. Dynamic distribution of file and directory management across multiple machines is done when files and directory are opened. This offers superior performance and scalability compared to traditional server-based file systems.
- Use of cooperative caching  
Local cache miss, and load on ObS is reduced due to the use of the collective memory of all participating machines as a global cache.
- Lack of dedicated machines  
Any machine mounting *zFS* can run the file manager and lease manager. Hence, machines can automatically get exclusive access to files and directories when they are the sole users. Moreover, any machine in the system can assume the responsibility of failed components. For more details on the recovery mechanism see [8].
- Use of Object Stores  
The use of ObS relieves the file system from handling metadata chores of allocating/removing and keeping track of disk blocks, thus reducing the overhead of the file system management.

## 6 Related Work

There are very few SAN file systems which use Object Store and we give a short description of each here.

Lustre [4] is a SAN file-system based on of three components: *clients*, *Cluster control-system*, and *Storage Targets*, connected together by a SAN. The clients see a cluster file system with standard POSIX semantics.

The *Cluster Control Systems* does not handle files data; rather it manages name-space, metadata coherence, security

---

<sup>3</sup> How other nodes find the corresponding file manager for *F* is described in [8].

and cluster recovery while directing clients to perform file I/O directly with storage targets.

Target stores are Object Stores which are programmable allowing execution of modules downloaded to them.

Lustre is an open-source project which is currently under development.

StorageTank [7], is a SAN file system built in IBM and composed of *clients*, *metadata servers* and *Storage*. In the first version it will use SAN connected conventional block storage and will operate with trusted clients. Later versions will use Object Stores to enable Storage Tank operation with any client. Storage Tank is commercially available under the name “SAN-FS”.

Clients and Object Stores are connected directly by a SAN, allowing efficient movement of large amounts of data. Metadata servers are connected to the other components by an IP network. The roles of the metadata servers are to maintain the file systems metadata state, and are responsible for all coherency and management chores. Metadata servers may be clustered for scalability and fault-tolerance.

## 7 Summary

SAN file systems based on block devices poses security, scalability and management challenges which can hardly be answered by the existing technologies. The emerging Object Store technology alleviates these problems and allows large SAN file systems to share data among many clients in a secure and manageable way. While Object Stores address the current challenges they also introduce new ones, like the need to find new ways to execute transactional operations. Object Stores also create the opportunity to use cooperative caching between object stores and (with some additional provisions) between clients and thus further improve performance. These issues are subjects for our future research.

## 8 Acknowledgements

We wish to thank Alain Azagury, Michael Factor, Kalman Meth, Ohad Rodeh, Uri Schonfeld and Avi Weit from IBM Research Labs in Haifa for their useful discussions on this subject.

## 9 References

- [1] IBM TotalStorage™ SAN File System – Draft Protocol Specification.
- [2] M. K. Aguilera, M. Ji, M. Lillibridge, J. MacCormick, E. Oertli, D. Anderson, M. Burrows, T. Mann and C. A. Thekkath, Block-Level Security for Network-Attached Disks, *Proceedings of FAST '03: @nd USENIX Conference on File and Storage Technologies*, San Francisco, CA, USA, March 31-April 2, 2003
- [3] A. Azagury, V. Dreizin, M. Factor, E. Henis, D. Naor, N. Rinetzky, O. Rodeh, J. Satran, A. Tavori and L. Yerushalmi, Toward an Object Store, *proceeding 20th*

*IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies*, April 2003, 165-176

- [4] P. J. Braam, The Lustre Storage Architecture, *Technical Report*, Cluster File Systems, Inc., 2002, <http://www.lustre.org/docs/lustre.pdf>
- [5] G. A. Gibson, D. P. Nagle, K. Amiri, F. W. Chang, H. Gobioff, E. Reidel, D. Rochberg and J. Zelenka, *File-systems for network-attached secure disks*, 1997
- [6] G. A. Gibson, D. P. Nagle, K. Amiri, F. W. Chang, E. Feinberg, H. G. C. Lee, B. Ozceri, E. Reidel and D. Rochberg, A case for network-attached secure disks. *Technical Report*, CMU-CS-96-142, CMU 1996
- [7] J. Menon, D. A. Pease, R. Rees, L. Duyanovich and B. Hillsberg, IBM Storage Tank – A heterogenous scalable SAN file system, *IBM SYSTEMS JOURNAL*, Vol. 42, No. 2, 2003
- [8] O. Rodeh and A. Teperman. zFS – A Scalable Distributed File System Using Object Disks, *proceeding 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies*, April 2003, 207-218
- [9] J. Satran and M. Factor, Intelligent Storage – Object Storage and Beyond, *IPSI-2003*