Distributed Data Storage Based on Web Access and IBP Infrastructure

Lukáš Hejtmánek
Faculty of Informatics, Masaryk University Brno,
Botanická 68a, 602 00 Brno,
Czech Republic
xhejtman@mail.muni.cz

Abstract – **This paper introduces an environment for distributed data storage based on Web access and an Internet Backplane Protocol storage infrastructure as well as a model for distributed data storage and load balanced Web front-ends. We describe combining a Web load balancing system with an IBP storage. Integrating Web access with IBP through `libxio` provides an access interface to the large storage capacities. Security aspects and solution for highly secure data storage based on users' certificates is discussed as well as integration of IBP storage with files versioning. Mention is also made of four pilot groups that uses this system for storing digital pictures and videos.**

Key-Words – **Distributed data storage, IBP, load balancing, `libxio`, web access**

1 Introduction

Recently it has become obvious that from the hardware perspective, both available network and storage capacities are growing faster than processing capacity exceeding Moore's law prediction [1]. Storage facilities are frequently built as distributed systems to achieve better scalability, higher availability and performance while often keeping costs lower when compared with centralized storage. Current high performance applications are thus often designed with the presumption that plenty of storage is available at users' disposal. However, the research community is still seeking for the optimal system that would enable applications to access distributed storage in an uniform, easy to use and yet very efficient manner. There is a number of projects aimed at exploring distributed storage in context of Grid computing [2], [3], [4] since the Grid environments and applications are perceived as best candidates for utilizing the distributed storage.

The Grid related projects in the Czech Republic are covered by the MetaCenter project [5] which provides parallel supercomputing facilities as well as powerful processing capacity in the form of distributed PC clusters. The MetaCenter project was enhanced during year 2003 by a new project called Distributed Data Storage (DiDaS) aimed at complementing the processing capacity with sufficient distributed storage capacity. In this paper we describe our approach to creating scalable and secure distributed storage which is to have a global name space that is easily accessible to end users and that can be integrated into the Grid environments but that is capable of stand-alone operation as well. We also briefly describe several data-intensive pilot applications that are the key drivers of our effort because they are nearly impossible to implement without some scalable distributed storage.

The rest of this paper is organized as follows: Section 2 gives a short introduction to state of art of the distributed storage, Section 3 gives an overview of the directory services and the I-Node representation used for building the file system, and Section 4 details the security aspects of our approach. Section 5 details the implementation of the user interface and directory services, Section 6 gives some performance evaluation results, Section 7 introduces pilot applications and user groups, and Section 8 gives some direction for the planned future development of the whole system.

2 State of Art and Related Work

Two basic approaches are being adopted in distributed storage today: first group contains systems that provide full-featured file system while the other group are systems that provide basic block storage only. The systems from the first group (AFS [6], CODA [7], etc.) provide very advanced semantics and features and high degree of robustness while sacrificing overall performance.

The systems from the latter group like Internet Backplane Protocol (IBP) [8], GridFTP [9], iSCSI [10], and Google File System [11] are more low-level oriented usually providing block storage comparable to UNIX I-Node concept with rather limited semantics. The points of focus for these approaches are usually scalability and performance. These systems can be also seen as a bridge toward robust and high-performance peer-to-peer networks. However, such systems need some higher-level developer interfaces to facilitate their adoption by applications developers.

Because of performance demands of our pilot applications, we have decided to choose the latter approach and build our infrastructure on top of block-oriented distributed data storage. The block oriented approach also allows us to create per-file replicas and to stripe even single replica of a file across multiple hosts thus enabling load balancing over all available depots for large files without need for creating as many replicas as the number of available depots.

The file-oriented GridFTP is an extension of FTP protocol designed to support requirements by Grid environments like data transmission over parallel TCP channels and integration with Grid Security Infrastructure (GSI) [12]. iSCSI standing for "Internet SCSI" is an Internet Protocol based storage networking standard for linking data storage facilities. By carrying SCSI commands over IP networks, iSCSI is used to facilitate data transfers over intranets and to manage storage over long distances.

The IBP developed at the University of Tennessee is a new promising light-weight protocol for storing data blocks over the networks. It is designed from scratch to provide high performance best effort storage service similar to best effort IP service in Internet. It allows multiple TCP connections for both uploading and downloading data, time-limited and volatile allocations and due to its purely block nature, it can be used for stripping large files across multiple servers. As the IBP has become more popular, its designers have begun to implement directory services for the data stored in IBP. Preliminary applications were IBPvo [13] and IBPmail [14] where users shared their XML metadata by sending it as e-mails. The second step was to develop a basic directory service called the XDN site [15] where users could apply for an account and then publish their data though XML metadata. Nowadays the XDN site is deprecated by the LoDN site [16]. The LoDN site offers more functionality then the XDN site. Users can store their data that they want to publish and to keep private. Downloading is performed via a Java applet, otherwise users can download XML metadata and then download a particular file on their own. The LoDN site runs on a single web server with CGI scripts that offer common manipulation with directories and files. Users can manipulate their files only and can download published files by other users. However this project does not have a load-balanced architecture without a single point of failure and file versioning. Also it has just a simple authorization permitting only public and private data. Downloading through a Java applet is similar to downloading through the web server. The advantage of the Java applet does not increase the load on the server and can use multiple TCP streams but it does not achieve the speed of binary tools that are also available as a part of IBP. Downloading through CGI binary is faster, but a greater load is placed on the web server and it always uses a single TCP stream from the web server to the client.

Both the IBP and the GridFTP can run in user-space only while the iSCSI requires support in hosting operating system kernel. The iSCSI also requires rather complicated coordination of parallel access to the distributed storage. We perceive the hard-coded integration of GridFTP with GSI as a disadvantage of this system as it requires rather complicated infrastructure to be set up even if it is not needed by target application. Thus we have chosen to base our implementation of distributed storage on IBP protocol and infrastructure which will be described in more detail later on in this paper.

Probably the closest related project to ours is the Google File System (GFS) [11] which however does not have any open-source implementation according to our knowledge. It is based on autonomous servers storing blocks of data od fixed size (IBP uses variable size). There is a master server that keeps metadata and provides directory services. A client receives metadata from this master server which keeps them cached for limited period of time and then the client communicates directly with the storage servers. Due to its block nature, it also provides data replication on block level and it provides just API instead of native filesystem in a way similar to our system. It contains very sophisticated garbage collector that it however not needed in IBP because it is implicitly built in in form of time limited allocations. Quotas implemented in GFS do not need to exist in IBP because of time limited allocations. Another difference between GFS and IBP based infrastructures is that the IBP can provide secure anonymous data storage (the users are not allowed to read/rewrite/delete data of other users unless explicitly allowed to do so).

3 Model Overview

Target of our model is a system, that can be run in the user space of the hosting operating system only, that can perform efficient garbage collection either explicitly or implicitly, and that has very decentralized architecture. We also want to have per-file replication setup and possibility to stripe files across several storage servers. The distributed storage should be accessible via library that mimics standard UNIX I/O functions to allow application developers to incorporate this functionality into their applications easily. We have targeted our work at applications that upload the data into the storage infrastructure once, download them many times, and change them rarely or not at all allowing us to implement just subset of full UNIX I/O functionality.

After testing various distributed storage systems (some of them being mentioned in the previous section) we have found that none of them completely satisfies our needs and we have opted for basing our development effort on the Internet Backplane Protocol (IBP). This consists of low-level storage infrastructure building blocks on top of which we create our services, most importantly the directory services and user access services. We are proposing an architecture that will not have a single point of failure. Also as the IBP doesn't provide secure data transfer mechanisms and we deploy the whole infrastructure in an Internet environment, we need to secure the underlying network infrastructure in order to protect data from unauthorized and malicious users.

Our directory service provides semantics weaker then that of the standard UNIX semantics, as it is mainly used for storing large collections of data that are written once and then read many times. This design is suitable for applications such as large video and image archives that are also among our primary pilot applications.

A common problem with distributed high-performance filesystems is that no applications support them directly. One approach might be in emulating a local filesystem in a transparent way which usually requires some modification to the kernel of operating system. We have therefore developed a `libxio` (Sec. 5.3) library that provides a straightforward interface resembling traditional UNIX file operations.

Our model of distributed storage uses architecture composed of two layers. The lower layer used for file storage and representation is called *I-Node* abstraction and is based on the IBP infrastructure. The higher layer provides *directory services* and *user access services*.

3.1 I-Nodes

The I-Nodes layer is entirely based on the IBP infrastructure. The IBP works in a soft consistency mode offering time-limited allocation. The basic atomic unit of the IBP is a byte array which can be used in a way similar to the traditional UNIX I-Node concept. The byte arrays are collected as serializable metadata information (called exNode) which describes particular files. An IBP depot (server) is a basic building block of the IBP infrastructure offering disc capacity in the form of the byte arrays. The IBP depots are registered with an L-Bone server that tries to perform load balancing and minimize the network utilization based on optimizing the location of the data across the depots. All the depots registered with one L-Bone server can be seen as a single depot providing large storage capacity.

After the data are uploaded into some depots of the infrastructure, an XML serialization of metadata is created and usually stored as a file. Metadata consists of the identification of byte arrays in which the data are stored; this is similar to the UNIX I-Nodes concept except that the IBP blocks can be of any size, can contain duplicates (file replicas), and can use various end-to-end services [17]. The soft consistency of the IBP model is ensured by the fact that each block (file) allocation has time-limited duration[1].

3.2 Directory and User Access Services

The I-Nodes need to be collected by some directory services if they are to be available via a common directory structure. We decided to create a Web based directory service. To provide a more robust and scalable service without a single point of failure we decided to use load balancing strategies on the web server level.

When using setup with multiple web servers, we need to ensure consistency by distributing files with a serialized representation of metadata to all the web servers. The most important goal is to prevent users from creating files with the same name stored under the same path name.

Our semantics is weaker than that of UNIX. Our semantics allows files to be read and written. We allow multiple clients reading the same file whereas multiple clients writing are forbidden (with the exception of writing different versions of the file as shown below). Files are removed immediately and there is no reference counting performed unless versioning is performed. Files have limited duration in time but there are special persistent directories where expiration time is refreshed by a specialized daemon.

[1]The IBP allocation blocks have unique sequential numbers to ensure that the same block is not allocated by another user when previous allocation expires and thus even if the previous user still has the serialized metadata, he is unable to read the original blocks that now contains new data belonging to new allocation.

3.3 Directory Services Versioning

Due to the nature of the IBP byte arrays, the CVS-like versioning system can be offered: the IBP byte array may not be overwritten and the IBP depot allows the byte array to be appended only. When the user alters a file, a new byte array is allocated and a new file with serialized metadata representation is created. Thus we can keep track of old metadata files and hold all the IBP allocations. By default, the user has access to the latest version of the file although any available version can be retrieved on request. When modifying part of a file only, the unmodified parts of the file are shared among all the versions of the particular file. This means that versioning is optimized for space at the expense of a non-trivial file being removed while all the versions must be checked to see if a particular byte array is being shared or not (e. g. by reference counting technique).

When the user opens a file, a particular version is accessed until it is closed. It is possible that new versions of the file appear while the file is being accessed in which case the user keeps accessing the version that was used when the file was first opened.

4 Security Model

Security aspects differ between local storage and network storage. For local storage, only the administrator has access to low level media and it is therefore enough to use only a common UNIX ACL system of security having users, groups and three kinds of access (read, write, execute/directory access). The administrator is a trusted person who will not compromise nor tamper with the data of users. Users are protected against one another by access rights while no ordinary user has access to low level media to bypass the protection mechanism provided by the kernel of the operating system.

In the case of network storage, we are dealing with the presumption of an evil Internet. Many users can have access to low level media on the network level and therefore it is necessary to prevent man-in-the-middle attacks to keep users' data safe. This requires secure communication channels between the server and client as well as among the servers. We must ensure that no one can fake user's identity and no unauthorized person can get access to the data during network transfer.

The IBP has its own security model. When the user allocates a byte array at some IBP depot, the IBP depot returns an array of IBP capabilities—the capability for reading, writing and managing. Any user possessing these capabilities has access to that byte array which can read and write (in append only mode) and can change the duration and destroy the array. However those capabilities are transferred through insecure TCP sockets both from the server to the client and from the client to the server and stored in an XML serialization of the metadata.

Besides these capabilities, the IBP offers an encryption as a part of end-to-end services. The user can choose the AES or DES encryption of byte arrays. Each byte array can have its own encryption key. This key is also stored in an XML metadata file but it is not transferred to the IBP depot.

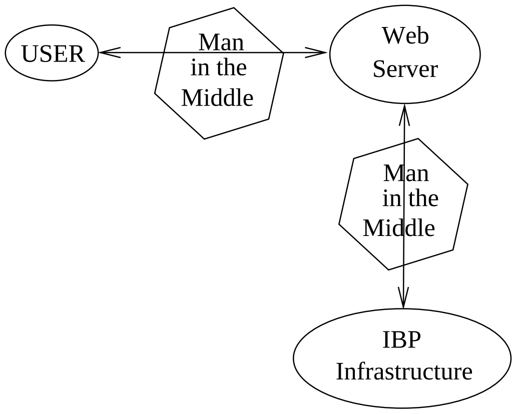

Fig. 1. Man-in-the-middle attack.

To keep the low level media secure, we are forcing users to use HTTPS to communicate with the Web Server. Almost any user is able to establish HTTPS connection to the Web Server but not many users are able to establish an IPsec channel to the Web Server or to use other secure methods to make the connection secure. On the other hand we are able to control our Web Servers and the IBP infrastructure so that the Web Servers and IBP depots communicate via IPsec channels as shown in the Fig. 2.

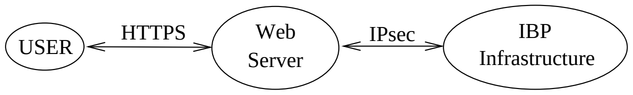

Fig. 2. Secure channels.

It is obvious that XML metadata should be transferred only via secure channels. We store XML metadata on the shared AFS volume at each Web Server using IPsec tunnels among Web Servers and AFS servers in order to keep the metadata safe. Also web servers and the IBP depots are connected via IPsec tunnels as shown in the Fig. 2 to prevent capabilities disclosure.

Since AES/DES encryption keys are stored directly in the XML metadata file, the administrators can decrypt a user's data and users need to use their own methods of encryption such as PGP if they need to store highly sensitive data.

Users can upload files directly to the IBP depots using native IBP tools and only then upload XML metadata files. In such a case the data can be protected using end-to-end AES/DES encryption; however, the capabilities can be eavesdropped and the attacker can use them only to remove a user's data since reading content is protected by encryption. However, there is a workaround for this issue: When user uploads an XML metadata file to the web server, it may be requested to create a copy of the file. The copying process is performed by the web server in the secure environment so the eavesdropper does not gain access to the capabilities thus he cannot remove user's data. The user can the download the new copy to ensure it is the same as the original local file. An exploitable copy can be kept for download optimization, as the download utility can perform load balancing among the IBP depots or it can be removed to optimize the storage capacity usage.

Although the Directory Services are based on the web technology implemented by CGI scripts as shown below, we cannot use underlying the filesystem ACL system of security unless CGI scripts are *suid* executables and each user and group exists in the system. We are using authorization services for matching users, groups, and file privileges. File privileges are only applied to metadata files since without them it is impossible to access the files themselves.

5 Prototype Implementation

By the first quarter of 2004, the DiDaS project has built several IBP depots distributed across various locations in the Czech Republic with a total of more than 7 TB of storage. Various hardware is used in order to evaluate the performance of different systems ranging from software-based disk arrays to either internal or external hardware based solutions (using ATA, SATA, and SCSI disks)[2]. All arrays run in RAID5 setup and host computers are equipped with Intel Pentium 4 Xeon with 1 GB of RAM and Gigabit Ethernet interface and run Linux operating system. Each server hosts an IBP depot and web server and the server, with the SCSI disks also hosts the L-Bone Server, an LDAP Server, and an AFS Server.

The IBP layer consists of the IBP depots, L-Bone Server, and LDAP server. Currently only one L-Bone server and one LDAP server is used but it is possible to have a backup L-Bone Server and backup LDAP server as well. Each server has access to an AFS shared volume that stores XML metadata files.

5.1 User Interface

After successful authentication and authorization at the web server, the user is able to see a shared directory and his/her own private directory. These directories are browsable in the usual way by the web browsing. User can upload and download files or XML metadata if they have the appropriate access rights. When uploading a file users must specify their desired duration as the IBP byte array allocations are time-limited. Unlimited duration is possible only for files located in the special directory whose content is periodically refreshed by a refresh daemon. Users can also specify the number of copies and whether the content of the file should be encrypted.

As already discussed in Sec. 4, users can also use IBP native *lors tools* to directly access the IBP depots and then upload the XML metadata to directories on the web server.

5.2 Software Implementation

The software implementation can be divided into two parts: storage service and directory service that provides metadata handling.

a) Storage service.: This service consists of the IBP depots. The IBP depots are interconnected via IPsec

[2]Preliminary performance evaluation of internal hardware solutions can be found in [17].

tunnels to provide a high level of data security transferred among the servers. Users can access depots through the web servers then all the communication is encrypted (SSL/TLS secured HTTP protocol) or they can access depots directly using *lors tools* but in such a case the security is weakened as discussed in Sec. 4. The IBP depots are registered with the L-Bone server. We use a modified L-Bone server which monitors disk capacity usage on the particular IBP depot. The L-Bone server also runs a *prediction* server that collects information about disk usage on other IBP depots and instructs the L-Bone server to offer less loaded depots if possible.

b) Directory service.: This part consists of a metadata repository, authority services and load-balanced Apache web servers as shown in Fig. 3.

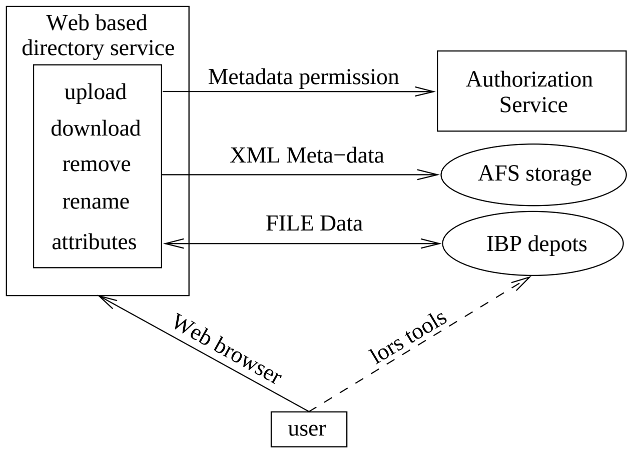

Fig. 3. Directory Service overview.

CGI scripts run on web servers to offer basic manipulation with files. Users can *upload*, *download*, *remove*, *rename*, and *change attributes*. The attributes allow users to set the level of redundancy, expiration period of files, and access permissions for files. The scripts use `libxio` library (Sec. 5.3) to access files in the IBP depots and they store the resulting serialized XML metadata file onto the shared AFS volume.

Authentication of users is based either on basic HTTP authentication or on user certificates. Currently the authorization service is implemented directly on the servers but we intend to either create an autonomous *authorization service* or use an existing one that performs ACL matching for stored files.

Metadata are stored on the local disk and are made accessible via the web server. However as one web server would be a bottleneck for the whole system and would create a single point of failure in the architecture as discussed above, we have set up several web servers coordinated via `mod_backhand` [18] that performs load balancing based on monitoring the system load on each machine. The `mod_backhand` can be used in two basic modes: it can be configured as a proxy or it can perform HTTP redirects to other web servers. When working in proxy mode, it contacts the web server with the lowest load and then proxies data to the client. The redirection mode uses the HTTP redirection request to another web server. Fig. 4 shows the basic setup with redirecting where all web servers are equal in their functions and Fig. 5 shows a possible two tier setup where one layer of web servers is used solely for redirecting. It is also necessary to combine web server load-balancing with a DNS round robin in order to avoid all clients overloading one server with initial requests.

Currently our implementation uses the model shown in Fig. 4 where all the web servers run `mod_backhand` and redirect to other server if needed. However because of the stateless architecture of HTTP protocol, there might be a whole sequence of redirects and it might take quite a long time for some server to accept the client. Therefore we also use URL rewriting to send information to the server that the request has already been redirected, so that the request must not be redirected any more.

5.3 `libxio` IBP Access Library

In order to simplify adoption of distributed storage in end user applications we have developed the `libxio` abstraction library that closely mimics standard UNIX I/O interface. It can access both IBP stored data when IBP URI is used and local files when normal filename is provided instead of IBP URI. The library provides equivalents for following standard I/O functions: `open`, `close`, `read`, `write`, `fttruncate`, `lseek`, `stat`, `fstat`, and `lstat`. The functions can be used in common way except for that IBP stored data can not be opened in `O_RDWR` mode at the moment.

The IBP URI has the following format:

```
lors://host:port/local_path/file
  ?bs=number&duration=number
  &copies=number&threads=number
  &timeout=number&servers=number
  &size=number
```

where the `host` parameter is a specification of a L-Bone server to be used, the `port` is a specification of L-Bone server port (default is 6767), the `bs` is a specification of block-size for transfer in megabytes (default value is 10 MB), the `duration` specifies allocation duration in seconds (default is 3600 s), requested number of replicas is specified by the `copies` (defaulting to 1), the `threads` specifies number of threads (concurrent TCP streams) to be used (default is 1), the `timeout` parameter is specification of timeout in seconds (defaulting to 100 s), the `servers` parameter specifies number of different IBP depots to be used (default is 1), and the `size` specifies projected size of file to ensure that IBP depot has enough free storage. It is possible to override default values using environment variables, too. If the given filename doesn't start with the `lors://` prefix, the `local_path/file` is accessed as local file instead.

When writing a file into the IBP infrastructure, the `local_path/file` specifies the local file where a serialized XML representation of the file in IBP will be stored. We advise users to use `.xnd` suffix for metadata files. At least an L-Bone server must be specified when writing a file

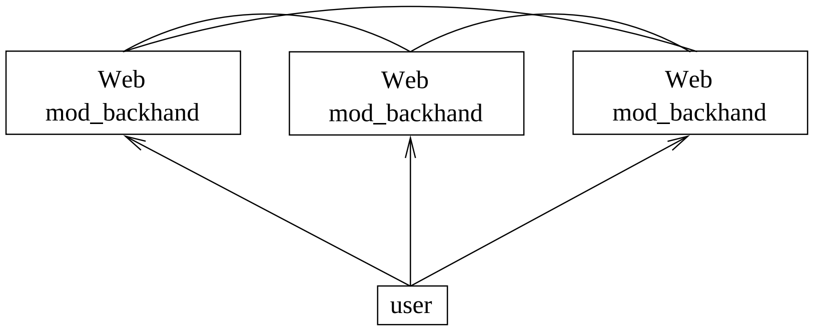

Fig. 4. `mod_backhand` redirect configuration with all servers having the same role.

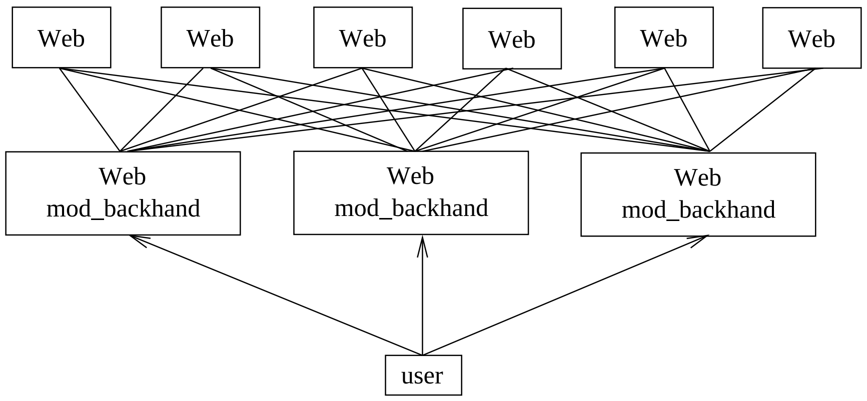

Fig. 5. Two tier `mod_backhand` redirect configuration.

into IBP. In our experience the metadata file occupies approximately 1/10th of the actual data size in IBP on average.

When a file stored in IBP is read, the `local_path/file` specifies the local file containing a serialized XML representation of the IBP file. User can also use a short form URI `lors:///local_path/file` as the servers are already specified in local XML representation.

Based on `libxio` library we have enabled IBP capabilities in several pilot applications. More information on `libxio` with detailed description of both user and developer interfaces can be found in [17].

6 Performance Evaluation of Prototype Implementation

For verifying our storage infrastructure we have performed measurements of bandwidth achievable for varying number of clients. We wanted to provide values that are close to real-world results of our applications and thus we used part of DiDaS production infrastructure as a testbed for the measurements. As parallel reading performance is critical for our applications, we have focused our measurements on reading performance with multiple parallel clients. We used following three storage servers:

- a server equipped Intel Pentium 4, 1 GB RAM, internal hardware RAID-5 array with SCSI disks, and GE network interface connected directly to client cluster in Brno
- a server equipped Intel Pentium 4, 1 GB RAM, internal hardware RAID-5 with ATA disks, and GE network interface connected via a router with bandwidth of approx. 400 Mbps
- a server equipped Intel Pentium 4, 1 GB RAM, external hardware RAID-5 with ATA disks, and GE network interface located in Prague connected with 2.5 Gbps WAN link (the bottleneck is local GE interface)

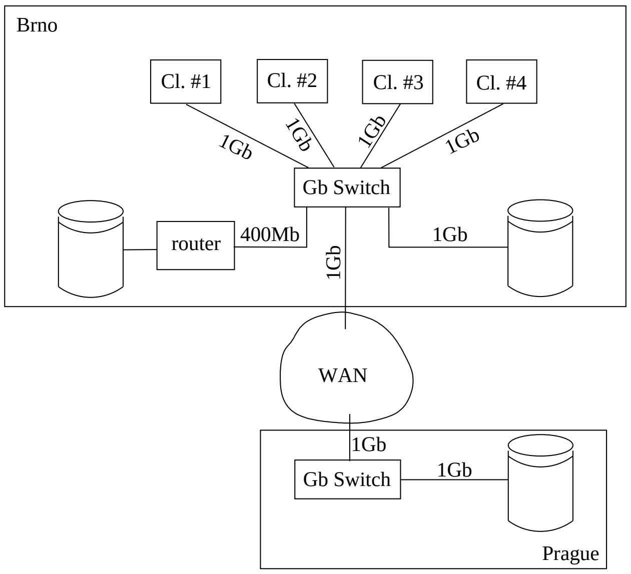

Fig. 6. Measurement testbed setup.

as shown in Fig. 6. The data for measurement were uploaded into IBP so that all the data were evenly distributed across all the depots.

We selected up to 20 nodes from MetaCenter PC cluster infrastructure to work as clients, all located in the same cluster in Brno. All clients were equipped with 2× Intel Pentium 4 at 2.5 GHz, 1 GB of RAM and a Gigabit Ethernet LAN connection. The client jobs were distributed evenly across the machines.

The results of the experiment are summarized in Fig. 7. It can be seen that the system scales well for at least 64 simultaneous clients downloading the data. For 128 clients the performance suffers from sudden drop which is subject to further investigation.

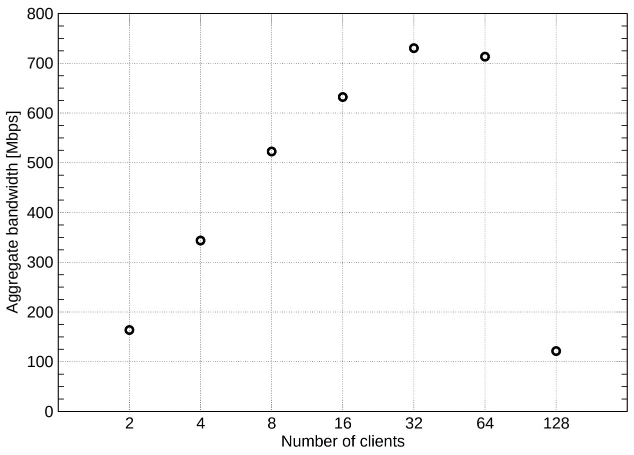

Fig. 7. Aggregate bandwidth achieved when reading from storage testbed for varying number of clients.

7 Pilot Applications

There are several user groups used as pilot applications by our project. One group is at the Faculty of Informatics, Masaryk University, which has been recording lectures at the faculty for the last two years [19]. This group needs to record, process, and store more than 20 hours of video per week. Each lecture is recorded as raw video in Digital Video (DV) format [20] (requiring approximately 14 GB per hour), decoded to raw video (requiring temporarily even 7× more capacity), de-interlaced, cleaned and finally encoded into target RealMedia format used for streaming in two qualities and DivX format used for downloading [21]. The size required for one lecture is approximately 700 MB in total. Due to trends of inter-faculty or even inter-university lecturing, we expect to be able to publish the content of recorded lectures to all the students at the participating universities.

The second group is at the National Library in Prague which has vast archives of digital books and maps. The archives consist of digitally scanned pages of books and maps in several resolutions (qualities). Users have an overview using files with small resolution while more details can be obtained with higher resolution scans. It consists of about two million files totaling about 1 TB of data. These digital materials should be made available for academic use. Potential users are spread across the whole Czech Republic so the distributed solution is very suitable.

The third group is also at the National Library in Prague. They have digital scans stored in MrSID [22] format. We aim to store corresponding open format of that data into our depots and then to offer an application to MrSID in the form of the browser plug-in.

The fourth pilot application is a storage capacity that can be used by all the Czech academic community so as to enable participants to store and share their large data sets.

8 Future Work

We would like to focus our attention on integrating our web server access to the IBP infrastructure with a real filesystem in the Linux operating system via LUFS project [23]. The goal here is to develop a distributed filesystem with reduced semantics close to FTP semantics with a CVS-like versioning. Authorization in this filesystem will be done through certificates for the authorization service or through Kerberos tokens.

We would also like to offer some kind of generic service interface on the web server level. For example when user upload a video file, a change of encoding might be requested during or just after the upload process. The CGI scripts will set up a transcoding job to run on the MetaCenter cluster computing facilities to perform transcoding to the desired format.

Another goal is to offer different semantics for the file access. Most significantly we would like to allow concurrent writing to the same file as some cluster computations require very large shared output files. This file can be read even while writing is taking place.

Acknowledgments

The work has been supported by CESNET Development Fund project 018/2002. We would also like to thank to Luděk Matyska and Petr Holub for kindly supporting our work and for stimulating discussions, and to Miroslav Ruda and the rest of MetaCenter staff for helping us with getting our storage infrastructure integrated with the MetaCenter grid infrastructure.

References

[1] M. D. Brown *et al.*. “Blueprint for the future of high-performance networking.” Communications of the ACM, 46(11):30–77, 2003. Special issue on High-Performance Networking.

[2] *The DataGrid Project*. `http://eu-datagrid.web.cern.ch/eu-datagrid/`.

[3] R. Buyya. “The virtual laboratory project.” IEEE Distributed Systems Online, 2(5), 2001. `http://computer.org/dsonline/0105/features/tvl0105.htm`.

[4] *Information Power Grid: NASA’s Computing and Storage Grid*. `http://www.ipg.nasa.gov/`.

[5] *MetaCenter Project, CESNET*. `http://meta.cesnet.cz/`.

[6] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. J. West. “Scale and performance in a distributed file system.” ACM Transactions on Computer Systems, 6(1):51–81, 1988. `http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/coda/Web/docdir/s%11.pdf`.

[7] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere. “Coda: A highly available file system for a distributed workstation environment.” IEEE Transactions on Computers, 39(4):447–459, 1990.

[8] M. Beck, T. Moore, and J. S. Planck. “An end-to-end approach to globally scalable network storage.” In *SIGCOMM’02*, 2002. `http://loci.cs.utk.edu/ibp/files/pdf/SIGCOMM02p1783-beck.pdf`.

[9] B. Allcock, J. Bester, J. Bresnahan, A. N. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. “Data management and transfer in high performance computational grid environments.” Parallel Computing Journal, 28(5):749–771, May 2002. `http://www.globus.org/research/papers/dataMgmt.pdf`.

[10] J. Satran, K. Meth, C. Sapuntzakis, M. Chadalapaka, and E. Zeidner. *iSCSI*. IETF Draft, IP Storage Working Group, Internet Engineering Task Force, January 2003. `http://www.ietf.org/internet-drafts/draft-ietf-ips-iscsi-20.txt`.

[11] S. Ghemawat, H. Gobioff, and S.-T. Leung. “The google file system.” In *19th ACM Symposium on Operating Systems Principles*, October 2003. `http://www.cs.rochester.edu/sosp2003/papers/p125-ghemawat.pdf`.

[12] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. “A security architecture for computational grids.” In *Proc. 5th ACM Conference on Computer and Communications Security Conference*, pages 83–92, 1998.

[13] M. Beck and T. Moore. “Logistical networking for digital video on internet2.” In *Fall 2003 Internet2 Member Meeting*, Indianapolis, IN, USA, October 2003. `http://loci.cs.utk.edu/modules.php?name=Publications&lid=213`.

[14] W. R. Elwasif, J. S. Plank, M. Beck, and R. Wolski. “IBP-mail: Controlled delivery of large mail files.” In *NetStore ’99: Network Storage Symposium*, Seattle, WA, USA, October 1999. `http://loci.cs.utk.edu/modules.php?name=Publications&lid=221`.

[15] *XDN site*. `http://promise.sinrg.cs.utk.edu/xdn/login.html`.

[16] *LoDN site*. `http://promise.sinrg.cs.utk.edu/lodn`.

[17] L. Hejtm´anek and P. Holub. *IBP deployment tests and integration with DiDaS project*. Technical Report 20/2003, CESNET, 2003.

[18] Y. Amir, T. Schlossnagle, *et al.*. *mod_backhand*. `http://www.backhand.org/mod_backhand/`.

[19] E. Hladk´a and M. Liška. “Předn´ašky ze z´aznamu (Lecture Recording).” In *Širokopásmové sítě a jejich aplikace (Broadband Networks and Their Applications)*, pages 130–133, CVP UP Olomouc, 2003. ISBN 80-244-0642-X. Available in Czech only.

[20] Internation Electrotechnical Commission. *IEC 61834: Recording – Helical-scan digital video cassette recording system using 6,35 mm magnetic tape for consumer use (525-60, 625-50, 1125-60 and 1250-50 systems)*, 1998, 1999, 2001. Parts 1–10, `http://www.iec.ch`.

[21] P. Holub and L. Hejtm´anek. “Distributed encoding environment based on grids and IBP infrastructure.” In *Terena Networking Conference ’04*, Rhodes, Greece, July 2004. Accepted submission.

[22] *MrSID: A Dynamic Image Format*. `http://www.lizardtech.com/solutions/geo/mrsid_overview.php`.

[23] F. Malita. *LUFS: Linux Userland FileSystem*. `http://lufs.sourceforge.net`.