A Highly Scalable and Efficient Distributed File Storage System

Fawad Riasat Raja[†] **Dr. Adeel Akram**[††]

Faculty of Telecommunication & Information Engineering
University of Engineering & Technology Taxila, Pakistan

ABSTRACT

The need and use of large scale distributed storage has rapidly increased in last few years. Organizations need Terabytes of storage for their operational data and backups. Large storage systems are the ultimate solution, but they are very expensive and require higher degree of skills for their operation and maintenance e.g. Storage Area Network (SAN). We propose "A Highly Scalable and Efficient Distributed File Storage System" that is reliable, inexpensive and easy to maintain. Our system is based on peer-to-peer network architecture. To ensure the reliability of the system we use a technique of erasure codes known as Luby Transform (LT). The System is designed for deployment in Local Area Networks (LAN) but with minimal changes it can be extended for Wide Area Networks (WAN) and Internet.

Key words:
Distributed Storage Systems, Peer-to-Peer Networks, Consistent Hashing, Data Blocks.

1. INTRODUCTION

With the passage of time, storage space requirements of small businesses to large enterprises increased by many folds for archival of their operational data and backups. Information in the form of e-mails, documents, presentations, databases, images and multimedia contents etc., require Terabytes of storage space. Storing information and managing its storage in a limited budget is a critical issue for small businesses as well as for large enterprises.

Vendors come up with different solutions day by day but these solutions are very expensive and hard to maintain. Some organizations uses file servers to overcome their storage requirements and when the need of storage grows, they add more hard disks or tape drives in their storage servers' farm to increase their storage capacity. For reliability, replication is used between the dedicated servers while their disk drives are organized in the form of RAID arrays e.g. RAID 1+0 or RAID 5. These types of storage solutions are not scalable and their management is another important issue [1]. Some of the storage systems use clustering technology [2] [3].

In Cluster technology, many computers or storage nodes are connected together using a SAN. But storage nodes connected in a cluster can share same account information with each other that may results in obvious security issues. Another problem with this solution is its cost and management.

We come up with a solution that addresses the above mentioned problems. Nowadays, a standard desktop PC has enormous computing and storage capacity. Usually a standard PC contains more than 100 GB Hard Disk Drive (HDD), 1 GB RAM and 2GHz or higher processor. A typical installation of an operating system and other required application software do not consume more than 20 to 30 GB of HDD storage. This leaves on the average about 70% of the storage space to be unused, especially in case of computers used in Laboratories and office environment. A small organization has more than 20 PCs. A University LAB for example, may contain on average around 30 PCs with above mentioned specifications. If the available storage capacity of these PCs is combined together, then a single LAB can provide 30 x 70 = 2100 GB of storage capacity. This surplus multi-Terabytes storage capacity remains unused in most of these LABs and can be utilized if combined to form a Large Virtual Storage Space to store huge amount of data. This motivation guided us in developing a Large Distributed File Storage System based on available storage capacities of existing PCs.

Our proposed system utilizes unused storage capacity of desktop machines (PCs) operating in small businesses, large enterprises or universities. Our design is based on completely decentralized (peer-to-peer) architecture. Main reasons behind using the peer-to peer architecture instead of client server architecture are:

- Resilience to failure
- Load Balancing
- Higher availability of resources

Manuscript received December 5, 2009
Manuscript revised December 20, 2009

This solution is also very cost effective because any organization whether small or large, can utilize this system over their existing resources (desktop machines) without purchasing any extra hardware or software components. We intend to make our system Open Source for Public use. The reliability of the system is achieved by using erasure codes for encoding and decoding of data blocks.

2. OVERVIEW

To ensure the data integrity and reliability we use a technique of erasure coding known as Luby Transform codes. Because of this encoding and decoding scheme our system can work even when some nodes (PCs) are offline. Section 3 describes the system architecture in detail. Section 4 states the system implementation. Section 5 compares our system with other Distributed Storage Systems. Section 6 concludes our work with discussion of future directions.

3. SYSTEM ARCHITECTURE

Our system is divided into the five modules as shown in [Figure.1].

- Graphical User Interface (GUI)
- Directory Manager
- Forward Error Correcting (FEC)
- Block Manager
- Node Look-up Service

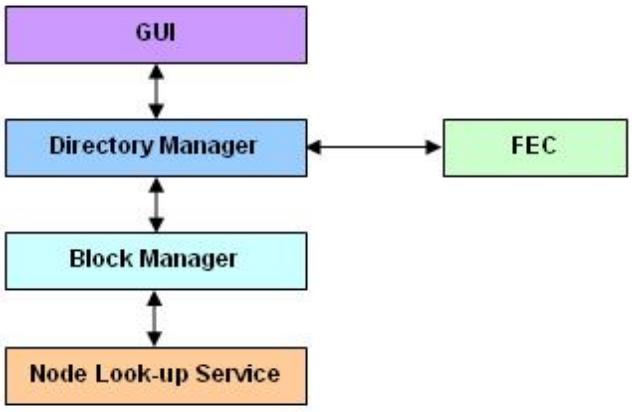

Figure.1 System Architecture

Before getting into the details of the system, let's have a brief overview of the high level working of our system. GUI module provides an interface for end users to interact with the system e.g. to upload, download and delete files etc. Directory Manager is responsible to split each file into smaller chunks also known as data blocks and forward these data blocks to FEC module for encoding and decoding. Directory Manager is also responsible for maintaining and updating the directory information. FEC module is responsible for receiving these data blocks from Directory Manager and encoding or decoding them for uploading and downloading as files respectively. FEC module replicates the data blocks to ensure the data reliability. Block Manager is responsible to receive the data blocks from Directory Manager and computes content hash of each data block to generate 160 bit hash key for each block. Each Data block is identified by this 160 bit hash key. Node Look-Up Service interacts with Block Manager and receives 160 bit hash key of each data block. On the basis of hash key, Node Look-Up Service identifies a node in the network that is responsible for storage/retrieval of that particular data block.

3.1 Graphical User Interface (GUI)

GUI gives the visual appearance of the virtual file system to the end user. GUI color schemes, layout, working and behavior are quite similar to Windows Explorer. Windows XP style task pane provides easy access to common operations and gives appealing look. Standard Toolbars, popup menus and shortcut keys make operation of software easy for all type of users. Easy to Use, Easy accessibility to functions and Appealing appearance are the main features of GUI.

3.2 Directory Manager

Directory Manager communicates with FEC and Block Manager Modules. Using the primitive functions provided by these modules Directory Manager provides more required features. Directory Manager also handles the virtual movements (move) and replication (copy paste) functions. This module also performs necessary transformations on data to make it understandable for a user or another module that uses it. Two major functions of Directory Manager are File Operations and Directory Navigation as shown in [Figure. 2].

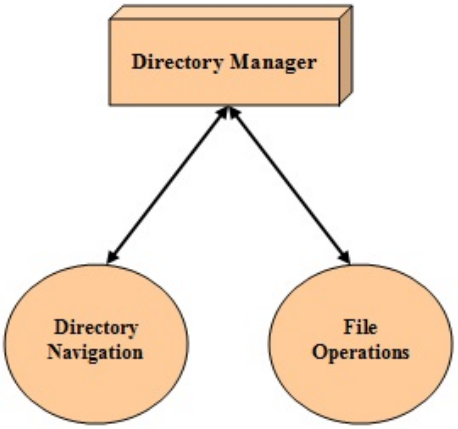

Figure.2 Basic Functions of Directory Manager

3.2.1 File Operations

Files and directories creation are core functions of any storage system. File and directory creation process is a bit complicated in our system as compared to a local real file system. We handled the complexity in the way so that for the users, it is similar to creating files/directories on local file system.
File Operations consists of File Creation/Uploading, File Downloading and Deletion of a File.

File Uploading is performed by reading the file from local file system. Then file is split into number of smaller chunks and these chunks are passed to FEC module to encode them. Encoded chunks are forwarded to Block Manager for saving them on the nodes connected through a peer to peer network. On success file information is added to XML File that provides storage file and directory manager for all users. Finally GUI is updated to reflect current changes.

Downloading a file is achieved by reading the file information from XML File and passing the information to Block Manager by calling the download function. Block Manager returns encoded file chunks those are then decoded by FEC module. On success, Directory Manager assembles the decoded chunks into a complete file and then save the file on local File system.

To delete a file, the file information is read from XML File and this information (Hash key of INODE block) is passed to Block Manager by calling the delete function. On success, file information is deleted from XML File and update GUI to reflect current changes.

3.2.2 Directory Navigation

Directory Navigation contains methods to perform navigation within the directory structure of a user. Navigation is visually same as in Windows explorer. Technically it is quite different from usual navigation operations as there is no real file system available and no built in methods could be used. Information for Directory structure of each user is maintained in an XML File. Directory navigation provides methods to navigate through this XML File and provides required data structures.

3.3 Forward Error Correcting (FEC Module)

FEC module interacts with Directory Manager Module. Core functionality of this module is to perform encoding and decoding of data blocks (file chunks) provided by the Directory Manager Module, to ensure the availability and reliability of the system.

As in Distributed Storage Systems, a file is stored in the form of chunks over different nodes so it may possible that some nodes are unavailable while retrieving the file, which may results in corruption of original file. This is the major problem in distributed storage systems those are based on peer to peer networks. To overcome this Problem, FEC module is based on Luby Transform codes.

Luby Transform (LT) codes are a class of erasure codes, called universal erasure codes. Length of symbol can be arbitrary for the codes, from one-bit binary symbols to general l-bit symbols. If the original data consists of n input symbols then each encoding symbol can be generated, independently of all other encoding symbols, on average by $O(\ln(n/\delta))$ symbol operations, and the k original input symbols can be recovered from any $n+O(\sqrt{n}\ln^2(n/\delta))$ of the encoding symbols with probability $1-\delta$ by on average $O(n \cdot \ln(n/\delta))$ symbol operations [4].

If encoded blocks are lost then entire file must be reconstructed or re-encoded to the network and because of this, all the previous encoded blocks are useless. The reason behind this is that most of the erasure codes have a set rate (see E.q. 1).

$$\text{Rate} = f_k / (k+l) \quad (1)$$

Where k is number of original data blocks and l is number of encoded data blocks [5].

Reed-Solomon and traditional LDPC codes have a set rate. Because of this they have same major drawback as mentioned above. This is the reason to choose Luby Transform Codes, a type of erasure codes but rate-less.

LT codes are rate-less in the sense that the number of encoding symbols that can be generated from the data is potentially immeasurable. Encoding symbols can be generated as many as needed. Decoder can recover an exact copy of the data from any set of the generated encoding symbols that are only slightly longer in length than the data. Decoder can recover the data from minimum number of encoding symbols. Encoding and decoding times of LT codes are asymptotically very efficient as a function of the data length. LT codes are referred as universal codes in the sense that they are simultaneously near optimal for every erasure channel and they are very efficient as the data length grows [4].

3.4 Block Manager

Block Manager provides functionality for reliably uploading, downloading and deleting of files. The basic storage unit is a data block (file chunk), which is any kind of binary data represented by variable length, byte sequence. Each block is identified by Hash Key which is a 160 bit key. The Hash Key is computed by using SHA-1 hashing algorithm. [Figure.3] represents three basic functions of Block Manager.

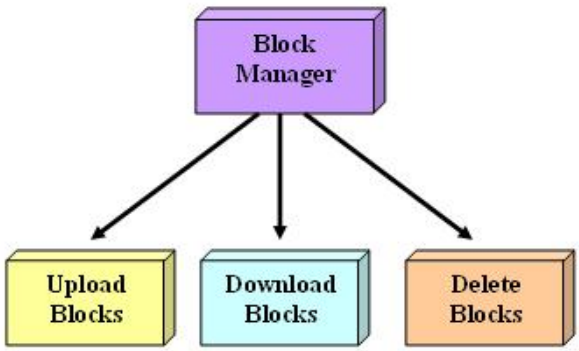

Figure.3 Basic Functions of Block Manager

Block Manager interacts with "Node Look-Up Service" Module by providing it 160 bit hash key of each block (chunk) of a file that is to be uploaded. In return "Node Look-Up Service" module provides addresses (IP Addresses) of all nodes connected in peer to peer network, those are responsible for storing these blocks (chunks). Block Manager creates a TCP connection with each node and store the particular data block (file chunk) over that node. TCP Protocol is used for these connections with each node to ensure the reliability. After successfully storing/uploading all data blocks of a file over particular nodes, Block Manager generates an INODE Block which contains keys of each data block and addresses of all nodes over which these blocks are stored. Block Manager then computes hash key of this INODE Block and forward this key to "Node Look-Up Service". Node look-Up Service returns the address of a node in the network that is responsible for storing this INODE Block. Block Manager then creates TCP Connection with this node and stores the INODE Block over it. Block Manager then sends this key to Directory Manager Module. Directory Manager maintains this key for downloading or deletion of the file. For downloading or deletion of the file directory Manager sends INODE key to Block Manager. Block Manager then retrieves INODE Block that contains keys of all data blocks and addresses al all nodes on which these blocks are stored. Block Manager creates connections with these nodes to download or delete these data blocks of a file.

Block Manager uses the concept of computing the contents' hash of data blocks. Owner count is embedded in each data block to maintain the information, if there are more than one owners of a single file. Because of this, a file with same contents cannot be stored more than once. If a user tries to upload a file that already exists then owner count is incremented instead of uploading the file once more and a message is returned to user that your file is successfully uploaded. This concept helps in saving the valuable storage space.

3.4.1 Upload blocks

[Figure.4] represents the functionality of Block Manager in uploading the data blocks of a file.

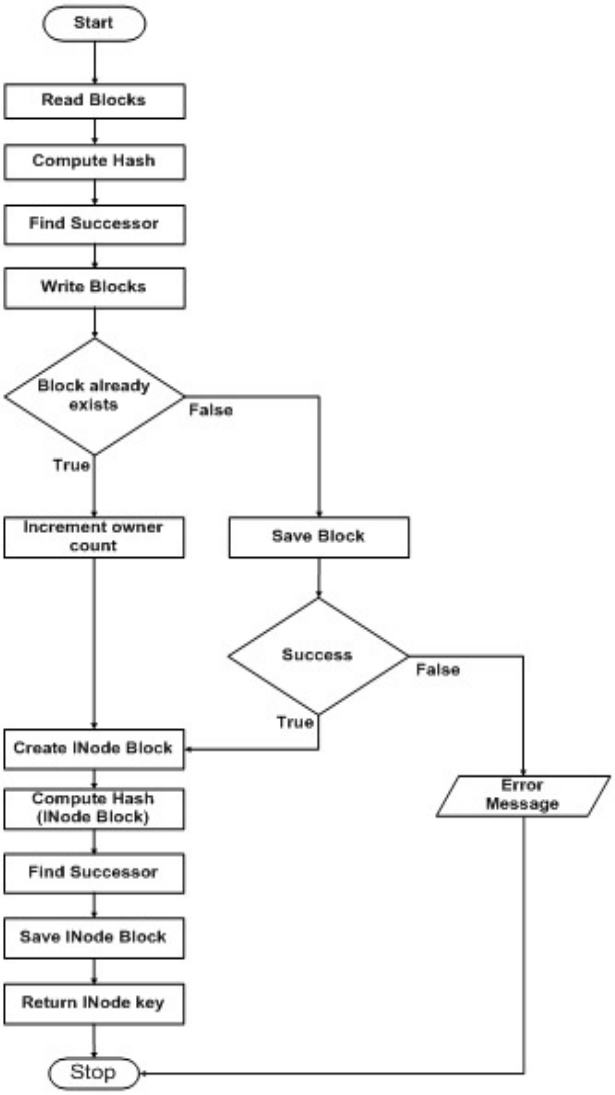

Figure.4 Flow chart of Uploading Data Blocks

3.4.2 Download Blocks

[Figure.5] represents the functionality of Block Manager in downloading the data blocks of a file.

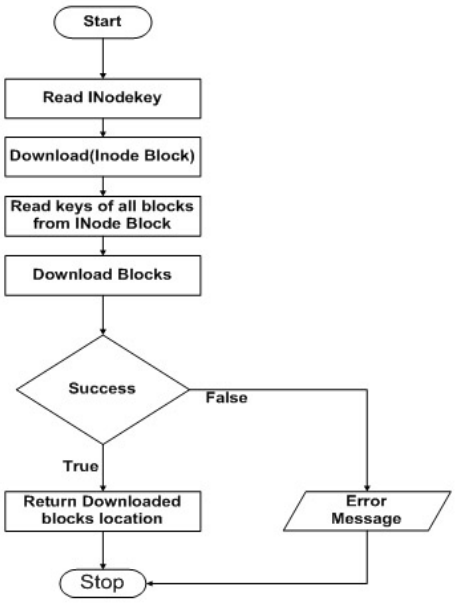

Figure.5 Flow chart of Downloading Data Blocks

3.4.3 Delete Blocks

[Figure.6] represents the functionality of Block Manager in deleting the data blocks of a file.

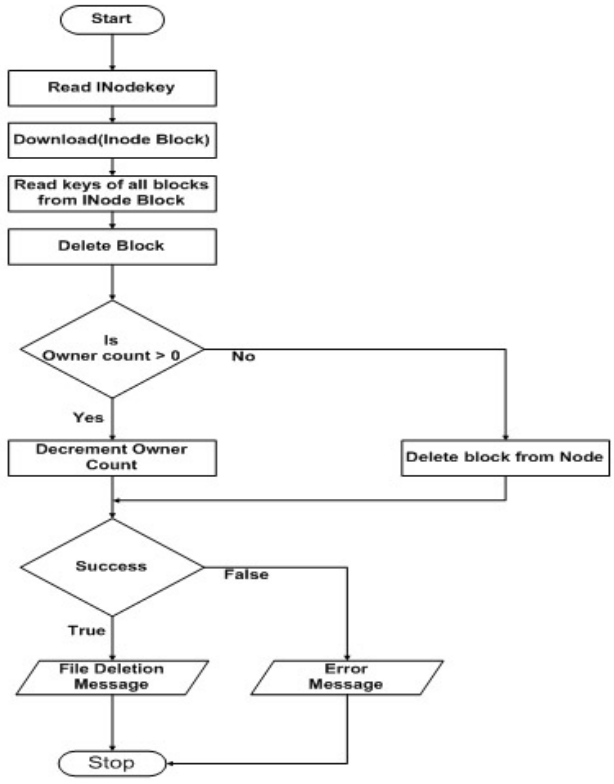

Figure.6 Flow chart of Deleting Data Blocks

3.5 Node Look-Up Service

Node Look-Up Service plays a vital role in our system and performs a routing mechanism. This module interacts with Block Manager module for receiving the hash key of each data block of a file that is to be uploaded. On the basis of this 160 bit hash key Node Look-Up Service depicts a particular node that is responsible for storing the entire data block.

There are number of routing protocols available including Pastry [6], Tapestry [7] and Chord [8]. These are self organizing and completely decentralized systems. The functionality of these systems is to map key of a give data block (file chunk) to a particular node in the network and provides efficient routing scheme as well as endure node failures. Our system uses Chord, described and evaluated in [8], as a routing protocol under Node Look-Up Service Module. Here we present a brief description of Chord.

Chord is a scalable, fault resilient and efficient peer-to-peer lookup protocol. Chord addresses the basic problem of peer to peer applications i.e. the efficient location of the node that stores a preferred data block. The Chord Protocol map a given key onto a node and that node is responsible for storing the value associated with that key. Chord uses consistent hashing [9] to assign keys to chord nodes. Consistent Hashing provides load balancing, since each node receives same number of keys and requires relatively little movement of keys when nodes join and leave the system.

A Chord node requires information about O(logN) other nodes for efficient routing but performance degrades when that information is out of date. Because nodes join and leave arbitrarily, the consistency of O(logN) sate may be hard to maintain. Chord has a simple algorithm to maintain information about other O(logN) nodes. Simplicity, correctness and Efficiency are the most important features of Chord those distinguish it from many other peer-to-peer lookup protocols that's why we choose Chord as routing protocols in our system.

4. IMPLEMENTATION

We design this system to run over Windows Platform therefore we chose .NET framework and VB.NET was selected as a programming language. [Figure.1] shows the detailed system architecture and described in Section 3. The system is based on completely decentralized architecture i.e. peer-to-peer architecture to ensure fault tolerance, load balancing and higher availability. To reduce the complexity of the system it is divided into five different sub modules.

5. COMPARISON

For architectural comparison, we compared our system with various other distributed storage systems available in literature e.g. LANStore [1] is designed only for Local Area Network and to ensure data reliability it used Reed-Solomon codes. Because of overhead of Reed-Solomon Codes they are not much efficient. Distributed Storage Systems must be used to store and retrieve real time data and should not act just as a backup system. Distributed storage system should be much efficient to access real time data but LANStore is designed just as a backup system instead of storage system.

OceanStore [10] is a global scale storage system on a multicast over relay network. It uses Tapestry [7] for locating nodes responsible for storage/retrieval of data blocks. To achieve data redundancy OceanStore use both erasure coding and mirroring. Data nodes in OceanStore serves different responsibility e.g. inner ring nodes are used for handling data redundancy. But this solution is not possible for Laboratories where data nodes are desktop machines and they are unable to handle much processing load.

FAB [11] defines a storage system with a block level interface. This system is designed on the basis of thin client and thick server model. Client uses SCSI command for data handling. But this solution is not suitable for small offices and laboratories.

GFS [3] is much more successful than the systems listed above. But it is not available to anyone to actually use and test it. Therefore it is not a feasible solution for companies that need an in house storage solution.

To overcome all the problems in existing distributed storage systems we present a much better and inexpensive solution. Scalability of our system increases because we use Chord as a routing protocol. Chord is designed for millions of nodes that's why our system can work efficiently for both LAN and WAN. Chord maintains the system stable when data nodes joins and leaves the network arbitrarily. To ensure data reliability we use Luby Transform codes, a technique of erasure codes. LT codes are rate-less and universal so they don't require a specific data block to recover a file from the network. Another important aspect of our system design is that we compute the contents' hash of all data blocks of a file and in each data block we embed an owner count. Because of this, a file with same content cannot be stored twice in our system. This concept helps in saving the valuable storage space and also increases the efficiency of the system.

6. CONCLUSION AND FUTURE WORK

In this paper we presented a robust storage solution i.e. highly scalable, efficient, reliable and inexpensive. As the system is designed for desktop machines, with the passage of time Processing Power, RAM, and Storage Space of these machines will increase significantly. Consequently, with the introduction of newer PCs with higher capacities, the performance and storage of our System will automatically increase to cater for backup and archive requirements of new users and PCs. In future these types of solutions will be widely used.

We deployed this system over 5 LABs in our university with desktop machines connected on LAN. In future we intend to configure it for use over geographically diverse locations connected via internet and measure its performance with respect to its scalability, efficiency and reliability. We would also like to address the security issues related to the safety of users data from malicious attacks.

Acknowledgments

We would like to acknowledge Mr. Naveed, Mr. Khurram, Mr. Imran, Mr. Arsalan, Mr. Tasawer, Mr. Arham and Mr. Basir for their valuable contribution and support.

References

[1] Bilickiv,V.: LanStore: a highly distributed reliable file storage system, The 3rd International Conference on .Net Technologies, University of West Bohemia, Plzen, Czech Republic, May 30 – June 1, 2006.

[2] M. Abd-El-Malek, W.V. Courtright, C.Cranor, Etal.Ursa Minor: Versatile Cluster-Based Storage. Proc. Of the 4th USENIX Conference on FAST.2005.

[3] S. Ghernawat, H. Gobioff, S-T. Leung. The Google File System. Proc. Of the 19th ACM SOSP. 2003.

[4] Michael Luby. LT codes. In The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. http://citeseer.ist.psu.edu/luby02lt.html.

[5] www.cs.uc.edu/~annexste/Courses/cs728-008/.../WBVYS8-MimirPaper.pdf

[6] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems". IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, pages 329-350, November, 2001.

[7] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph," Tapestry: An infrastructure for faulttolerant widearea location and routing," UC Berkeley, Tech. Rep. UCB/CSD-01-1141, April 2001.

[8] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M. F., AND BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup service for internet applications. Tech. Rep. TR-819, MIT, Cambridge, MA, March 2001.

[9] D.R. Karger, E. Lehrnan, F. Leighton, M. Levine, D. Lewin and R. Panigrahy, "Consistent Hashing and random trees: Distributed cashing Protocols for relieving hot spot on the World Wide Web", in Proc. 29th Annual. ACM Symp. Theory of computing, El Paso, TX, May 1997, pp.654663.

[10] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao and J. Kubiatowicz. Pond: The OceanStore Prototype. In Proceedings of the Conference on File and Storage Technology (FAST) 2003.

[11] S. Frolund, A. Merchant, Y. Saito, S. Spence, and A. Veitch. FAB: Enterprise Storage Systems on a Shoestring. In 8th Workshop on Hot Topics in Operating Systems (HOTOSVIII), Kauai, HI, USA, May 2003.

Fawad Riasat Raja received his B.Sc. degree in Software Engineering from University of Engineering & Technology Taxila, Pakistan in 2006. Currently he is enrolled in M.Sc degree of Computer Engineering and also serving as Lecturer in the same University in the department of Software Engineering. His areas of interests are Computer Networks, Software Design & Architecture and Software Testing.

Prof. Dr. Adeel Akram did his Bachelors in Electrical Engineering from UET Lahore and MS in Computer Engineering from National University of Sciences and Technology, Rawalpindi, Pakistan in 1995 and 2000 respectively. He completed his PhD in the area of Ad hoc Wireless Networks from University of Engineering and Technology, Taxila, Pakistan in 2007. Dr. Adeel has over 12 years of industrial experience and has worked for Expert Systems (Pvt) Limited and University of Engineering and Technology Taxila. He has worked in different domains of technology including Computer Networks, Wireless Security, Digital communications, Cryptography, Embedded System Development, VoIP, and quality assurance.