**2. LITERATURE SURVEY**

* 1. **INTRODUCTION**

Literature survey deals with the process of defining the functions of existing system. To create or develop a new system and study the prior system, Analysis difficult problems faced by that system. The disadvantages of existing system are discussed to prove the way of proposed system. Then the proposed system is defined for the problem and the advantages of the proposed system are also defined.

* 1. **RELATED WORK**
     1. **The Data Recovery File System for Hadoop Cluster**

Data Recovery system is most challenging aspects in the internet or World Wide Web applications. Now a day evens a Tera Bytes (TB) and Peta Bytes (PB) of data is not enough for storing large chunks of database (DB). Hence IT industries use concept is known as Hadoop in their applications. This approach has been adopted in Cloud computing environment for unstructured data. Hadoop is an open source distributed computing framework based on java and supports large set of distributed data processing. HDFS (Hadoop Distributed File System) is popular for huge data sets and streams of operation on it. Available Hadoop in cloud is one of the important factors. But in Hadoop Distributed File System, Master Namenode Failure affects the performance of the Hadoop Cluster. Examine the behavior of Name node and what are the issues of Name node failure. Presents a Scenario to overcome this failure scheme replicates the Name node on the other Data node so that the availability of the metadata is increases and also Decreases the loss and delay of data. Hadoop cluster is suddenly unavailable when Name node is terminated. Given that HDFS Architecture Proper solution to increases the performance and decreases the time of delay. Proposed HDFS Architecture is solved automated failover problem as well as increases reliability of Hadoop and focused on selection of Backup Name node after failure of primary Name node.

* + 1. **Bitable: A Distributed Storage System for Structured Data**

Big table is a distributed storage system for managing structured data that is designed to scale to a very large size: Peta Bytes of data across thousands of commodity servers. Many projects at Google store data in Big table, including web indexing, Google Earth, and Google Finance. These applications place very different demands on Big table, both in terms of data size (from URLs to web pages to satellite imagery) and latency requirements (from backend bulk processing to real-time data serving). Despite these varied demands, Big table has successfully provided a flexible, high-performance solution for all of these Google products. Describe the simple data model provided by Big table, which gives clients dynamic control over data layout and format, and describe the design and implementation of Big table.

Bigtable is a distributed system for storing structured data at Google. Bigtable clusters have been in production use since April 2005, and spent roughly seven person-years on design and implementation before that date. As of August 2006, more than sixty projects are using Bigtable. Users like the performance and high availability provided by the Bigtable implementation, and that they can scale the capacity of their clusters by simply adding more machines to the system as their resource demands change over time. Given the unusual interface to Bigtable, an interesting question is how difficult it has been for users to adapt to using it. New users are sometimes uncertain of how to best use the Bigtable interface, particularly if they are accustomed to using relational databases that support general-purpose transactions. Nevertheless, the fact that many Google products successfully use Bigtable demonstrates that design works well in practice. In the process of implementing several additional Bigtable features, such as support for secondary indices and infrastructure for building cross-data-center replicated Bigtables with multiple master replicas. Deploying Bigtable as a service to product groups, so that individual groups do not need to maintain their own clusters. As service clusters scale, Need to deal with more resource-sharing issues within Bigtable itself [3, 5]. Finally, found that there are significant advantages to building own storage solution at Google. Gotten a substantial amount of flexibility from designing own data model for Bigtable. In addition, to control over Bigtable’s implementation, and the other Google infrastructure upon which Bigtable depends, means that can remove bottlenecks and inefficiencies as they arise.

* + 1. **Benchmarking Personal Cloud Storage**

Personal cloud storage services are data-intensive applications already producing a significant share of Internet traffic. Several solutions offered by different companies attract more and more people. However, little is known about each service capabilities, architecture and most of all performance implications of design choices. Presents a methodology to study cloud storage services. Apply methodology to compare 5 popular offers, revealing different system architectures and capabilities. The implications on performance of different designs are assessed executing a series of benchmarks. Results show no clear winner, with all services suffering from some limitations or having potential for improvement. In some scenarios, the upload of the same file set can take seven times more, wasting twice as much capacity. The methodology and results are useful thus as both benchmark and guideline for system design.

A methodology to check both capabilities and system design of personal cloud storage services. Then evaluated the implications of design choices on performance by analyzing 5 services. It shows the relevance of client capabilities and protocol design to personal cloud storage services. Dropbox implements most of the checked capabilities, and its sophisticated client clearly boosts performance, although some protocol tweaks seem possible to reduce network overhead. On the other extreme, Cloud Drive bandwidth wastage is an order of magnitude higher than other offerings, and its lack of client capabilities results in performance bottlenecks. SkyDrive shows some performance limitations, while Wuala generally performs well. More importantly, Wuala deploys client side encryption, and this feature does not seem to affect Wuala synchronization performance. These 4 examples confirm the role played by data center placement in a centralized approach: taking the perspective of European users only, network latency is still an important limitation for U.S. centric services, such as Drop box and SkyDrive. Services deploying data centers nearby test location, such as Wuala, have therefore an advantage. Google Drive follows a different approach resulting in a mixed picture: it enjoys the benefits of using Google’s capillary infrastructure and private backbone, which reduce network latency and speed up the system. However, protocols and client features limit performance, especially when multiple files are considered.

* + 1. **Inside Dropbox: Understanding Personal Cloud Storage Services**

Personal cloud storage services are gaining popularity. With a rush of providers to enter the market and an increasing offer of cheap storage space, it is to be expected that cloud storage will soon generate a high amount of Internet traffic. Very little is known about the architecture and the performance of such systems, and the workload they have to face. This understanding is essential for designing efficient cloud storage systems and predicting their impact on the network. Presents a characterization of Dropbox, the leading solution in personal cloud storage in the datasets. By means of passive measurements, analyze data from four vantage points in Europe, collected during 42 consecutive days. The contributions are threefold: Firstly, Dropbox, show to be the most widely-used cloud storage system, already accounting for a volume equivalent to around one third of the YouTube traffic at campus networks on some days. Secondly, characterize the workload users in different environments generate to the system, highlighting how this reflects on network traf- fic. Lastly, results show possible performance bottlenecks caused by both the current system architecture and the storage protocol. This is exacerbated for users connected far from storage data-centers.

To the best of knowledge, First to analyze the usage of Dropbox on the Internet. This analysis assessed the increasing interest on cloud-based storage systems. Major players like Google, Apple and Microsoft are offering the service. In this landscape, Dropbox is currently the most popular provider of such a system. By analyzing flows captured at 4 vantage points in Europe over a period of 42 consecutive days, The Dropbox is by now responsible for a considerable traffic volume. In one of datasets, for instance, Dropbox is already equivalent to one third of the YouTube traffic. Presented an extensive characterization of Dropbox, both in terms of the system workload as well as the typical usage. Main findings show that the Dropbox service performance is highly impacted by the distance between the clients and the data-centers, which are currently located in the U.S. The usage of per-chunk acknowledgments in the client protocol combined with the typically small chunk sizes deeply limits the effective throughput of the service. Identified two possible improvements to the protocol: (i) the usage of a chunk bundling scheme; (ii) the introduction of delayed acknowledgments. The recent deployment of a bundling mechanism improved the system performance dramatically. The overall performance will be improved by the deployment of other data-centers in different locations.

* + 1. **Implementation of BIST Capability using LFSR Techniques in UART**

The increasing growth of sub-micron technology has resulted in the difficulty of VLSI testing. Test and design for testability are recognized today as critical to a successful design. Built-in-Self Test (BIST) is becoming an alternative solution to the rising costs of external electrical testing and increasing complexity of devices Small increase in the cost of system reduces large testing cost. BIST is a design technique that allows a circuit to test itself Test pattern generator (TPG) using Linear Feedback Shift Resister (LFSR) is proposed which is more suitable for BIST architecture. Here implemented Universal asynchronous receiver transmitter (UART) with BIST capability using different LFSR techniques and compared these techniques for the logic utilization in SPARTAN3 XC3S200-4FT256 FPGA device.

It is concluded that BIST implementation for UART has increases area overhead, delay and design time as shown in figure 8 for each case but as LFSR replaces external tester feature such as TPG to give 100% fault coverage. In applications where area overhead is a big concern, LFSRs prove to be a better choice. Cellular Automaton provides a good alternative for LFSRs when high fault coverage is needed. BIST can be used to perform these special tests with additional on-chip test circuits, eliminating the need to acquire such high-end testers.

* + 1. **The Google File System**

Designed and implemented the Google File System, a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients. While sharing many of the same goals as previous distributed file systems, design has been driven by observations. Application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system assumptions. This has led us to reexamine traditional choices and explore radically different design points. The file system has successfully met storage needs. It is widely deployed within Google as the storage platform for the generation and processing of data used by the service as well as research and development efforts that require large data sets. The largest cluster to date provides hundreds of terabytes of storage across thousands of disks on over a thousand machines, and it is concurrently accessed by hundreds of clients. The file system interface extensions designed to support distributed applications, discuss many aspects of design, and report measurements from both micro-benchmarks and real world use.

The Google File System demonstrates the qualities essential for supporting large-scale data processing workloads on commodity hardware. While some design decisions are specific to unique setting, many may apply to data processing tasks of a similar magnitude and cost consciousness. Reexamining the traditional file system assumptions in light of current and anticipated application workloads and technological environment. Observations have led to radically different points in the design space. The component failures as the norm rather than the exception, optimize for huge files that are mostly appended to (perhaps concurrently) and then read (usually sequentially), and both extend and relax the standard file system interface to improve the overall system. The system provides fault tolerance by constant monitoring, replicating crucial data, and fast and automatic recovery. Chunk replication allows us to tolerate chunk server failures. The frequency of these failures motivated a novel online repair mechanism that regularly and transparently repairs the damage and compensates for lost replicas as soon as possible. Additionally, check summing to detect data corruption at the disk or IDE subsystem level, which becomes all too common given the number of disks in the system.

* + 1. **Securing Data Transfer In The Cloud Through Introducing Identification Packet And UDT -Authentication Option Field: A Characterization**

The emergence of various technologies has since pushed researchers to develop new protocols that support high density data transmissions in Wide Area Networks. Many of these protocols are TCP protocol variants, which have demonstrated better performance in simulation and several limited network experiments but have limited practical applications because of implementation and installation difficulties. On the other hand, users who need to transfer bulk data (e.g., in grid/cloud computing) usually turn to application level solutions where these variants do not fair well. Among protocols considered in the application level solutions are UDP-based protocols, such as UDT (UDP-based Data Transport Protocol) for cloud /grid computing. Despite the promising development of protocols like UDT, what remains to be a major challenge that current and future network designers face is to achieve survivability and security of data and networks. Previous research surveyed various security methodologies which led to the development of a framework for UDT. The lower level security by introducing an Identity Packet (IP) and Authentication Option (AO) for UDT.

New applications vary in traffic and connection characteristics in various communication links. Most of them still use TCP for data transfer because of its reliability and stability. Despite being widely used, there are a number of serious security flaws inherent in TCP and UDP [2,3,4]. There have also been performance issues in the implementation of large networks that require high bandwidths. On the other hand, the current and future Internet has no mechanism to constrain hosts that offer and use excess traffic more than the required bandwidths. Particularly, it contains no defense against DDoS (distributed denial of service) attacks. It does not include an architectural approach to protect its own elements from any attack [15]. Its existing link encryption may be sufficient and efficient for securing connectivity but not securing the Internet per se. The link encryption can also create network and application incompatibilities. The ‘first packet identity’ which needs to be instituted but devised in such as way that it is robust enough that a receiver cannot be flooded by requests that require the receiver to take excessive action before they have verified the identity and trust at the application level and also introduced and proposed a state-of-the-art security mechanism for UDT and for its future implementations to various network topologies by adding a field for AO, authentication options. Demonstrated the use of MD5 [25], the use of other hash functions, such as SHA-1 or SHA-256. The preceding discussions presented and focused on the conceptual low-level protection of the end node since UDT relies on TCP and UDP for data delivery. Proposed the inclusion of identity on its packet header (IP) and Authentication Option (AO) before the transmission is validated at the application level.

* + 1. **ZooKeeper: Wait-free Coordination for Internet-scale Systems**

ZooKeeper is a service for coordinating processes of distributed applications. Since ZooKeeper is part of critical infrastructure, ZooKeeper aims to provide a simple and high performance kernel for building more complex coordination primitives at the client. It incorporates elements from group messaging, shared registers, and distributed lock services in a replicated, centralized service. The interface exposed by Zoo-Keeper has the wait-free aspects of shared registers with an event-driven mechanism similar to cache invalidations of distributed file systems to provide a simple, yet powerful coordination service. The ZooKeeper interface enables a high-performance service implementation. In addition to the wait-free property, ZooKeeper provides a per client guarantee of FIFO execution of requests and linearizability for all requests that change the ZooKeeper state. These design decisions enable the implementation of a high performance processing pipeline with read requests being satisfied by local servers. The target workloads, 2:1 to 100:1 read to write ratio, that ZooKeeper can handle tens to hundreds of thousands of transactions per second. This performance allows ZooKeeper to be used extensively by client applications.

* + 1. **Modularizing B+-Trees: Three-Level B+-Trees Work Fine**

The objective of this research is to improve the single-thread performance of a B+-tree in memory. Existing works have been utilized changes in hardware for the performance improvement. While utilizing these works and modularize B+- trees in memory especially for write-intensive workloads. The modularization is mainly aimed at utilizing the difference in read/write ratio between levels, which is large for write-intensive workloads. It show how to modularize B+-trees and effective selections of algorithms and the node size at each level. The cost to switch algorithms is minimized because algorithms at a level are statically defined at compile time. The best selection formed three-level B+-trees and achieved two- or threefold performance improvement over a typical implementation of a B+-tree. In the three-level B+-trees, perform linear search on small unsorted leaf nodes, and interpolation search on the large sorted root node, and linear search on small sorted internal nodes.

Modularized B+-trees and show effective selections of algorithms and the size of nodes at each level. As a result, two- or threefold performance improvement was achieved over a typical implementation of a B+-tree. The best selection and formed three-level B+-trees, in which linear search was performed on small unsorted leaf nodes and on small sorted internal nodes except the root node, and interpolation search was performed on the large sorted root node. In addition, the three-level modular B+-trees enable us to utilize large pages and software prefetch only at levels specified. Evaluation results suggest that allocation of large pages to nodes at a high level takes precedence over that at a low level. Prefetching identifiers only on internal nodes increased throughput while prefetching those on all nodes decreased throughput. While utilizing techniques proposed in existing works in each module, propose the method to make use of the techniques in a single tree and effective selections of modules. The modularization increased throughput of get operations from 1.819 to 4.463 million pairs/second (2.453 times faster), and that of delete operations from 1.564 to 4.010 million pairs/second (2.564 times faster). The increase in throughput of put operations was larger than that of get and delete operations.

* + 1. **Web Caching With Consistent Hashing**

A key performance measure for the World Wide Web is the speed with which content is served to users. As traffic on the Web increases, users are faced with increasing delays and failures in data delivery. Web caching is one of the key strategies that has been explored to improve performance. An important issue in many caching systems is how to decide what is cached where at any given time. Solutions have included multicast queries and directory schemes. Offer a new Web caching strategy based on consistent hashing. Consistent hashing provides an alternative to multicast and directory schemes, and has several other advantages in load balancing and fault tolerance. Compared the system to other cache systems. Demonstrated that when the user is responsible for knowing which cache has the appropriate data, significant penalties in the critical loop of a user request can be avoided. This knowledge can be provided to the user with a hash function. Since it is likely that users of a large network, such as the Internet, may have inconsistent views of live caches, suggest the use of a Consistent Hash function which balances data quite well despite conflicting user views. Additionally, described implementation of the system that uses Consistent Hash function in order to show that it is quite practical to integrate such a system into the World Wide Web. The system handles locality issues, balances load among caches, and possesses a high level of fault tolerance that is absent from other Web caching systems.