Mapping and Classification of Key Features and Technologies behind Distributed File Systems to Make Recognition for Their Inimitable File System

Chitresh Verma, Rajiv Pandey, Devesh Katiyar

***Abstract:** Distributed file system had become a well known type of file system which suits requirement of users. AFS and HDFS are two popular distributed file systems. Andrew file system is distributed file system which had been used for location independent data storage. Hadoop distributed file system is an available as part of Hadoop package that is used for Big Data analysis. This paper makes the comparison of both file system with respect to feature, architecture and performance in depth. The paper explorers design concepts, authentication, inter process communication and access mechanism.*

***Index Terms:** Andrew file system, Hadoop distributed file system, Hadoop, Distributed file system, file system.*

I. INTRODUCTION

The storage system has been a point of discussion from point they were introduced. Storing large size of data on single disk or single machine was a major problem. Those multiple solution was suggestion in the course of time. The distributed storage of the data over the network became the generally accepted solution. The distributed storage systems need led to the development of several distributed file systems. The Andrew file system and Hadoop distributed file system are part of these popular distributed file systems.

We shall study all aspect of both distributed file systems, in order to increase our understanding of them. Andrew file system is one of the early file systems which were developed in 1980s. Comparatively with Andrew file system the Hadoop file system had been recently development.

This paper had been divided into multiple sections. First section presented an introduction of Andrew file system and Hadoop distributed file system. Second section presented brief explanation of distributed file system. Third section described the HDFS in detail. Similarly, fourth section described the AFS in detail. Fifth section provides appraisal of HDFS and AFS. Finally, sixth section describes conclusion part.

II. DISTRIBUTED FILE SYSTEM

The distributed file system had been an old concept but had regained popularity partly due to increasing cost of hardware and power needs. The distributed file system had described as storage system where data had been stored in different machines and these machines shall be connected to each other. Mostly, recognizable and similar patterns had been followed manner in any file system on actual storage system which could be understood in abstract form. Master nodes, slave nodes and their standalone database as well as accessing clients were some common object in distributed file systems. Basic operations are read, write and update on these distributed file systems. Multiple levels are present in the distributed file system to fulfill the needs of different function in the system and each level assumes the task as per predefined rules. These multiple levels help in handling call by concurrent multiple users at the systems for their respect requirements. Thus, the system becomes efficient and robust for the multiple clients. Major problem related to storage location and address had been solved by transparent in storage path name. The system had tried to make naming structure independent of actual file path. Thus, it had been simplified to address attachment of any file, independent of storage location in the system. Manually managing the machines record like names are difficult and are consider a bad practice as error may occur. Locating the machine names and other data manually is error inviting action in any scenario. Reliability of the system is increased by replication of data at multiple nodes but it also creates nightmares for maintainer of the system. The data availability is vital for the system so the replication of data is necessary but the data storage is mostly automatic. Thus, actual location of the data is difficult to identify each time. File systems and record keeping The systems designs are mostly same for different solution offered in the market for file systems and record keeping. The data stored in predefined location and later retrieved from their location. Encapsulation and abstraction
The data collection is automatic; the user only provides the data that needs to be store. The indexing and access mechanism is hidden from user which helps in simplification of use for the user. Restriction on the data could be applied if any need arise and creates suitable environment for query and indexing of the data. Naming of files and folders System organizes the files in well defined directory to the access mechanism of the files. This access mechanism is easily understood by the end users. But it solves the problem up to limited level as the files count growths, user's faces difficult in remembering the file path and names. Thus, certain search options are also provided by various distributed file system solution developers. Comparison of search time

Revised Manuscript Received on June 05, 2019
Chitresh Verma, Research scholar, AIIT, Amity University, UP, India.
Dr. Rajiv Pandey, AIIT, Amity University, UP, India.
Dr. Devesh Katiyar, Department of C.S., D.S.M.N.R.U., UP, India.

and handling time is used as tool for check that access mechanism is efficient or not. In case of least frequently used algorithm, the higher referencing file makes the system effective. Indexing the data and storing some data index based on count of references is widely used by the most file system.

Relationship between the distributed file systems and databases are comparable in various aspects. But they had basic differs between them. Distributed database exchange computation part to storage location whereas the distributed file system exchanges the data to their respective storage location. So, network bandwidth consumption is far high in case of distributed file systems compared to distributed database.

Large number of clients could access the distributed file systems at same time. Thus, concurrency and its control mechanism are key features of the distributed file systems.

Concurrency control: The concurrency controlling involves sequencing of read and writes operation on the shared resources so that the deadlock condition does not arises. Its more complex with distributed file systems as multiple access is available on each clients system and keeping the atomicity of each transaction. But the problem of sharing file lot to do with multiple write and read access as the data may be altered while it is still under read operation by other users. So, some kind of access control mechanism adopted in the distributed file system as per developer assessment.

Evolution of distributed file system: Distributed file systems had been inspired by existing centralized file systems. The centralized file systems had been adopted to store data in multiple locations at same time building a mechanism for location transparency. The data sharing is mostly reading files operation whereas write operation sharing is very rare operation. So, design of the distributed file systems had been around the read and write operations fetch and concurrency standardization. While many distributed file systems support data overwriting policy like systems logs, other systems did not support such policy to reduce complexity and easier maintain of the system, they simple create new copy of data for new data. Certain distributed file systems had designed with smart system identifier for different actions like identification of type of files. These smart systems could be understand as change in nature of operations like read only for critical data, furthermore write only for system logs when access for client end applications. In terms of data transfer in distributed file systems, temporal local copy of data had been kept as per fetching policy. The size of copy varies from different distributed file systems. Implementation mechanisms: Mounting: It is mechanism of binding the namespaces in such manner they give the impression of being part of single file system. The single hierarchical namespace had been used in the UNIX operating system and it has a special mount point for creating local namespace. Thus, the location transparency is high in such operating systems as user could not feels that the data is not local but it access in distributed environment. Although, system administrator could easily identify the non local namespace using various tools available. The special table tracks the mount points and namespace in most mount mechanism system. These tables were used for accessing remote server and authentication mechanism respectively. For example, network file system of Sun mounts each data set individually and client mounts namespaces in local file system. This creates issues with portability data as location may be multiple for different file entities. Thus, systems like UNIX operating system does not allow mount points to be located in two separate file systems. In some systems mount data like location and client details are managed in the servers. Thus, relocating the data from one location to another requires updates at server level only. The advantage of this system is that client gets namespace transparency. The mapping and location allocation is processed at server and thus the clients request the server for the location and data details. The acquired metadata for data access were used by the client software for all kind of operation on actual data location.

Data caching in the distributed file systems: The data caching plays a vital role in the schema of distributed file system. Cache is special temporary location on client machine which store the data copy for certain time period. Catching also involves the metadata storage for referencing the data. Caching may be in primary memory or secondary memory based. The cache size is important in both case of cache storage as large size requires the hardware on client side and smaller size may require recurrent data calls. So, optimized size of cache had been determined by the file system to make a balance approach.

III. HADOOP DISTRIBUTED FILE SYSTEM

HDFS is used for storage of the data on machines running on commodity hardware. HDFS is short form for Hadoop distributed file system. The file system is made up of multiple components to properly run the two basic operation of read and write operations. It was original built as part of Apache Nutch project but later became as part of Apache Hadoop project.

The Hadoop distributed file system is master-slave type distributed file system. There is a master node called NameNode and many slave nodes called DataNodes. These nodes run on commodity hardware and they are similar to other distributed file systems in various aspects. The unique features of HDFS are many functions like highly fault tolerant systems and running on low cost hardware. Big Data could be handle by HDFS which is one of its key features. Some of the POSIX guidelines had been loosen up for streaming the data in HDFS.

There are certain assumptions and goals in HDFS which had been discussed in this section.

In HDFS, hardware failure is considered as norm rather than the exception which means that failure of hardware is perfectly usual thing. Several servers and instances may be deployed as part of HDFS. It is assumed that some these servers will be not working at any given point of time, so replicated copies had to be kept at various servers. Also, quick and prompt detection and repair of servers form main goal of HDFS.

Accessing data in form of stream is a key feature of HDFS. Unlike traditional file systems, HDFS is batch oriented processing system since it focus on high level data access throughput. It is achieved by relaxing some of POSIX requirements.

The architecture of HDFS based on commodity machines. It consists of multiple nodes and these nodes separated into two types. The two types are DataNodes and NameNode which runs on distributed type of system. The nodes used mostly Linux operating system and Java language for their programming needs.

a) NameNode

NameNode is master node which maintains mapping of the slave nodes that is DataNodes. Changes in namespace of DataNodes are updated by the NameNode. So, all the DataNodes information is recorded by the NameNode and updated as per requirements.

b) Secondary NameNode

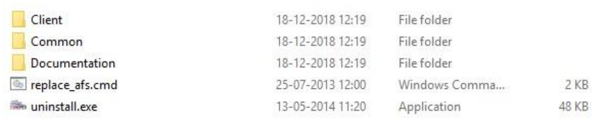
The secondary NameNode is kind of backup master server which becomes active if NameNode fails to work. The secondary NameNode makes HDFS more reliable system.

c) DataNodes

The DataNodes are slave server on which write and read operations are performed by HDFS client. Their addresses and other details are stored in NameNode as block which provide to HDFS client for various operations.

IV. ANDREW FILE SYSTEM

It was introduced by researcher as joint project between Carnegie-Mellon University and IBM in the 1980's. Andrew file system had been referred as AFS in short form. The main goal of AFS was scalability up to minimum to 7000 workstations with goal of one client for each user. It was distributed file system providing users, application programs and system administrators shared storage space. The below sub heading shall describe the features; architecture and performance of Andrew file system.



```
Client               18-12-2018 12:19   File folder
Common               18-12-2018 12:19   File folder
Documentation        18-12-2018 12:19   File folder
replace_afs.cmd      25-07-2013 12:00   Windows Comma...    2 KB
uninstall.exe        13-05-2014 11:20   Application        48 KB
```

Figure 1: Screenshot of files and folder available as part of Andrew client package.

Andrew file system had enhanced security and scalability when compared to traditional distributed file systems. It supports thousands of clients as a file system. The Andrew file system makes use of Kerberos authentication for its control mechanism. It creates shadow copy in cache in native client machine before synchronization with server machine. Additional advantage of this system is that in case of disconnected with the server, the client machine can read the copy in native machine. It supports weak consistency. So, the local copy is used for read and writes operations. After all operation had been completed, the copy is sync with the server copy.

V. COMPARSION OF HDFS AND AFS

Andrew file system and Hadoop distributed file system are design and implemented for different needs but some points of them could compared with each other. In below section, four parameters had been used for comparison of both file systems.

1. File system design

HDFS designed around "write once read many" concept. It creates a file only one time and then read it multiple times. It perfectly suits the MapReduce programming model under Hadoop framework. In future, appending writes may be supported. It follows model that moving computation is better than moving data. The computation performed on the local server where the data is stored provides better performance to entire system. Large data set moving data each time is very difficult, so this type model is an efficient solution. An interface is provided by HDFS which could be used by applications for locating the data location and then performing the data processing at local level. HDFS had been design by its developer to be platform independent. Thus, it could easily implement across various platform.

Andrew file system was designed as location independent file system with main goal of scalability. These goals had been achieved with certain assumptions. These assumptions includes files are small in size so that they could be cached in local file system. Number of read operations is much higher ratio compared to write operation. The access mechanism is mostly sequential access. All files are used by one user at any point of time, so no multiple read and write operation with different users. Mostly, all shared files are rarely altered in this system.

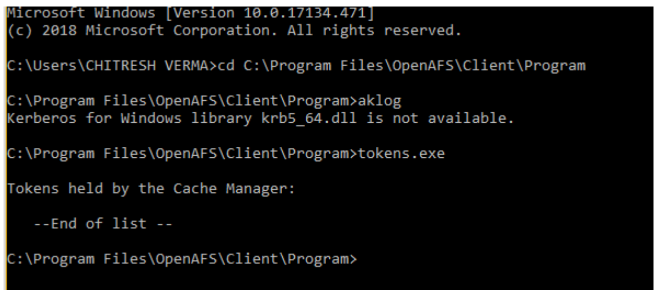
2. Implementation

The Hadoop distributed file system was designed by its developer to run on commodity hardware. The fault tolerant was key feature of the HDFS. Its implementation was based on Linux operating system.

The Andrew file system had been designed to run on high end hardware. Its key feature was scalability with the assumption that storage is unlimited. The server storage had the permanent copy of the data while shadow copies are created when and where they are needed.

3. Authentication

Kereros based authentication mechanism had been support by both file system. Kerberos allocates multiple tokens for communication. The tokens are held by cache manager, the tokens list could be obtain by using "token" command as shown in figure 2.



```
Microsoft Windows [Version 10.0.17134.471]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\CHITRESH VERMA>cd C:\Program Files\OpenAFS\Client\Program

C:\Program Files\OpenAFS\Client\Program>aklog
Kerberos for Windows library krb5_64.dll is not available.

C:\Program Files\OpenAFS\Client\Program>tokens.exe

Tokens held by the Cache Manager:

   --End of list --

C:\Program Files\OpenAFS\Client\Program>
```

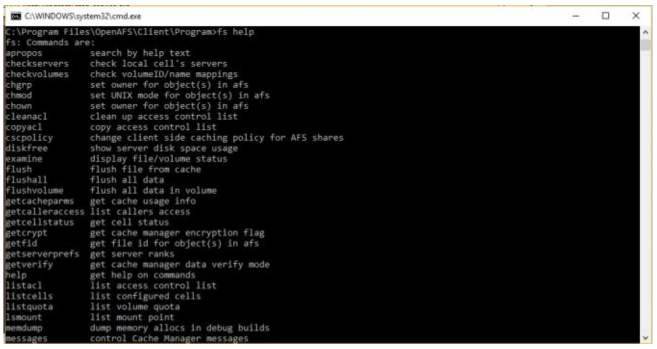
Figure 2: Screenshot of token command to list the token held by the cache manager.

4. Access lists

Andrew file system uses

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

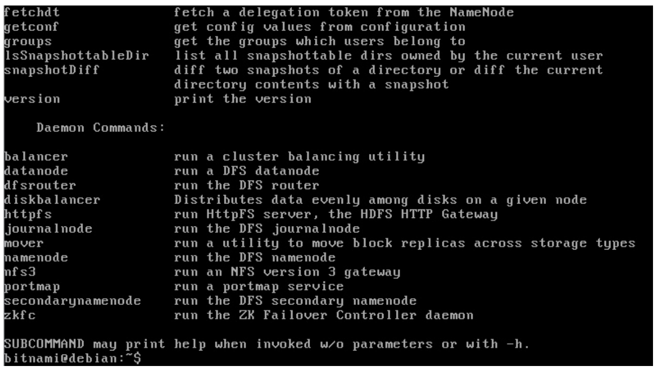
the fs command to list the files and directory access. The figure 3 shows the “fs” command options available to client user.



```
C:\WINDOWS\system32\cmd.exe
C:\Program Files\OpenAFS\Client\Program>fs help
fs: Commands are:
apropos         search by help text
checkservers    check local cell's servers
checkvolumes    check volumeID/name mappings
chgrp           set owner for object(s) in afs
chmod           set UNIX mode for object(s) in afs
chown           set owner for object(s) in afs
cleanacl        clean up access control list
copyacl         copy access control list
cscpolicy       change client side caching policy for AFS shares
diskfree        show server disk space usage
examine         display file/volume status
flush           flush file from cache
flushall        flush all data
flushvolume     flush all data in volume
getcacheparms   get cache usage info
getcalleraccess list callers access
getcellstatus   get cell status
getcrypt        get cache manager encryption flag
getfid          get file id for object(s) in afs
getserverprefs  get server ranks
getverify       get cache manager data verify mode
help            get help on commands
listacl         list access control list
listcells       list configured cells
listquota       list volume quota
lsmount         list mount point
memdump         dump memory allocs in debug builds
messages        control Cache Manager messages
```

Figure 3: Screenshot of fs command of Andrew file system

Hadoop distributed file system uses hdfs command for list the files and directory access. The figure 4 shows the “hdfs” command options available to client user.



```
fetchdt              fetch a delegation token from the NameNode
getconf              get config values from configuration
groups               get the groups which users belong to
lsSnapshottableDir   list all snapshottable dirs owned by the current user
snapshotDiff         diff two snapshots of a directory or diff the current
                     directory contents with a snapshot
version              print the version

    Daemon Commands:

balancer             run a cluster balancing utility
datanode             run a DFS datanode
dfsrouter            run the DFS router
diskbalancer         Distributes data evenly among disks on a given node
httpfs               run HttpFS server, the HDFS HTTP Gateway
journalnode          run the DFS journalnode
mover                run a utility to move block replicas across storage types
namenode             run the DFS namenode
nfs3                 run an NFS version 3 gateway
portmap              run a portmap service
secondarynamenode    run the DFS secondary namenode
zkfc                 run the ZK Failover Controller daemon

SUBCOMMAND may print help when invoked w/o parameters or with -h.
bitnami@debian:~$
```

Figure 4: Screenshot of hdfs command of Hadoop distributed file system.

VI. CONCLUSION

This paper explored various aspect of Andrew file system and Hadoop distributed file system. Andrew file system had been build to be highly scalable whereas Hadoop distributed file system had been build to run on commodity hardware with being scalable system. Access control of Andrew file system is strict compared to HDFS, as one centralized control mechanism available in Andrew. For small/medium level, implementation Andrew file system is well suited whereas Hadoop distributed file system is good for high level implementation involving Big Data. In future, the Andrew file system type centralized control mechanism may be tried on Hadoop distributed file system to check the various characteristics.

REFERENCES

1. Attebury, Garhan, et al. "Hadoop distributed file system for the Grid." Nuclear Science Symposium Conference Record (NSS/MIC), 2009 IEEE. IEEE, 2009.
2. Borthakur, Dhruba. "The hadoop distributed file system: Architecture and design." Hadoop Project Website 11.2007 (2007): 21.
3. Campbell, Richard, and Andrew Campbell. Managing AFS: The Andrew File System. Prentice Hall, 1998.
4. Chandrasekar, S., et al. "A novel indexing scheme for efficient handling of small files in hadoop distributed file system." Computer Communication and Informatics (ICCCI), 2013 International Conference on. IEEE, 2013.
5. Chang, Fay, et al. "Bigtable: A distributed storage system for structured data." ACM Transactions on Computer Systems (TOCS) 26.2 (2008): 4.
6. Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google File System." ACM SIGOPS Operating Systems Review SIGOPS Oper. Syst. Rev. (2003): 29.
7. Griffioen, Jim, and Randy Appleton. "Reducing File System Latency using a Predictive Approach." USENIX summer. 1994.
8. Howard, John H. An overview of the andrew file system. Carnegie Mellon University, Information Technology Center, 1988.
9. Howard, John H., et al. "Scale and performance in a distributed file system." ACM Transactions on Computer Systems (TOCS) 6.1 (1988): 51-81.
10. Huber Jr, James V., et al. "PPFS: A high performance portable parallel file system." Proceedings of the 9th international conference on Supercomputing. ACM, 1995.
11. Kaushik, Rini T., Milind Bhandarkar, and Klara Nahrstedt. "Evaluation and analysis of greenhdfs: A self-adaptive, energy-conserving variant of the hadoop distributed file system." Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on. IEEE, 2010.
12. Kazar, Michael Leon. Synchronization and caching issues in the Andrew file system. Carnegie Mellon University, Information Technology Center, 1988.
13. Leung, Andrew W., et al. "Measurement and Analysis of Large-Scale Network File System Workloads." USENIX Annual Technical Conference. Vol. 1. No. 2. 2008.
14. Liao, Haojun, Jizhong Han, and Jinyun Fang. "Multi-dimensional index on hadoop distributed file system." Networking, architecture and storage (nas), 2010 ieee fifth international conference on. IEEE, 2010.
15. Lin, Hsiao-Ying, et al. "Toward data confidentiality via integrating hybrid encryption schemes and Hadoop distributed file system." Advanced Information Networking and Applications (AINA), 2012 IEEE 26th International Conference on. IEEE, 2012.
16. Maltzahn, Carlos, et al. "Ceph as a scalable alternative to the hadoop distributed file system." login: The USENIX Magazine 35 (2010): 38-49.
17. Nelson, Michael N., Brent B. Welch, and John K. Ousterhout. "Caching in the Sprite network file system." ACM Transactions on Computer Systems (TOCS) 6.1 (1988): 134-154.
18. Patel, Aditya B., Manashvi Birla, and Ushma Nair. "Addressing big data problem using Hadoop and Map Reduce." Engineering (NUiCONE), 2012 Nirma University International Conference on. IEEE, 2012.
19. Satyanarayanan, Mahadev, et al. "Coda: A highly available file system for a distributed workstation environment." IEEE Transactions on computers 39.4 (1990): 447-459.
20. Shafer, Jeffrey, Scott Rixner, and Alan L. Cox. "The hadoop distributed filesystem: Balancing portability and performance." Performance Analysis of Systems & Software (ISPASS), 2010 IEEE International Symposium on. IEEE, 2010.
21. Shahabinejad, Mostafa, Majid Khabbazian, and Masoud Ardakani. "An efficient binary locally repairable code for hadoop distributed file system." IEEE Communications Letters 18.8 (2014): 1287-1290.
22. Shvachko, Konstantin, Hairong Kuang, Sanjay Radia, and Robert Chansler. "The Hadoop Distributed File System." 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST).
23. Sidebotham, Bob. "Vohunes: The Andrew File System Data Structuring Primitive." Proceedings of EUGG Autumn 86 (1986): 473-480.
24. Sur, Sayantan, et al. "Can high-performance interconnects benefit hadoop distributed file system." Workshop on Micro Architectural Support for Virtualization, Data Center Computing, and Clouds (MASVDC). Held in Conjunction with MICRO. 2010.
25. Thekkath, Chandramohan A., Timothy Mann, and Edward K. Lee. Frangipani: A scalable distributed file system. Vol. 31. No. 5. ACM, 1997.
26. Verma, Chitresh, and R. Pandey. "Comparative Analysis of GFS and HDFS: Technology and Architectural Landscape." Proceedings of the2016 IEEE International Conference on Communication Systems and Network Technologies (CSNT ‘16). IEEE. 2016.
27. Verma, Chitresh, and Rajiv Pandey. "An implementation approach of big data computation by mapping Java classes to MapReduce." Computing for Sustainable Global Development (INDIACom),

2016 3rd International Conference on. IEEE, 2016.
28. Verma, Chitresh, and Rajiv Pandey. "Analytics-as-a-Service (AaaS)." Exploring Enterprise Service Bus in the Service-Oriented Architecture Paradigm (2017): 26.
29. Verma, Chitresh, and Rajiv Pandey. "Big Data representation for grade analysis through Hadoop framework." Cloud System and Big Data Engineering (Confluence), 2016 6th International Conference. IEEE, 2016.
30. Verma, Chitresh, and Rajiv Pandey. "Mobile Cloud Computing Integrating Cloud, Mobile Computing, and Networking Services Through Virtualization." Design and Use of Virtualization Technology in Cloud Computing. IGI Global, 2018. 140-160.
31. Verma, Chitresh, Rajiv Pandey, and Devesh Katiyar. "Performance Evaluating System Based on MapReduce in Context of Educational Big Data." International Journal of Organizational and Collective Intelligence (IJOCI) 8.1 (2018): 1-12.
32. Zhang, Jing, et al. "A distributed cache for hadoop distributed file system in real-time cloud services." Grid Computing (GRID), 2012 ACM/IEEE 13th International Conference on. IEEE, 2012.

AUTHORS PROFILE

Chitresh Verma is a PhD scholar at Amity Institute of Information Technology, Amity University, Uttar Pradesh, India. He has good experience of IT industry. His research interests include the contemporary technologies as JAVA programming language, Cloud and Big Data, and Data Analytics.

Dr. Rajiv Pandey Senior Member IEEE is a Faculty at Amity Institute of Information Technology, Amity University, Uttar Pradesh, Lucknow Campus, India. He possesses a diverse back ground experience of around 33 years to include 15 years of Industry and 18 years of academic. His research interests include the contemporary technologies as Semantic Web Provenance, Cloud computing, Big- Data, and Data Analytics. He has been on technical Committees of Various Government and Private Universities.He is intellectually involved in supervising Doctorate Research Scholars and Post graduate Students. He is also an active contributor in professional bodies like IEEE, IET , Machine Intelligence Labs and LMA ..

Dr. Devesh Katiyar is Assistant Professor in Department of Computer Science at Dr. Shakuntala Misra National Rehabilitation University, Lucknow, India.