2012

Design and analysis of peer 2 peer operating system

Anudeep Meka
Louisiana State University and Agricultural and Mechanical College

DESIGN AND ANALYSIS OF PEER 2 PEER OPERATING SYSTEM

A Thesis

Submitted to the Graduate Faculty of the Louisiana State University and Agricultural and Mechanical College

in partial fulfillment of the

requirements for the degree of

Master of Science in Systems Science

In

The Interdepartmental Program in

The Department of Computer Science

By

Anudeep. Meka

B. E., Maharaj Vijayaram Gajapathi Raj College of Engineering, Jawaharlal Nehru Technological University 2010

Vizianagaram, India

May 2012.

ACKNOWLEDGEMENTS

I am very grateful to my advisor Dr. Supratik Mukhopadhyay for his guidance, patience and understanding throughout this work. His suggestions, discussions and constant encouragement have helped me to get a deep insight in my thesis work. I would like to thank Dr. Jianhua Chen, Dr. Konstantin Busch and Dr Jian Zhang for sparing their time to be a part of my thesis advisory committee.

Also I am very thankful to Department of Computer Science. Also I'm very thankful to the scalaris developer's team who helped me a lot in my work. I'm also very thankful to my friend Bharadwaj and Nutan who helped me. I am very much thankful to my friends.

I wish to endow my earnest gratitude to my parents, who believed in me and have been thorough all the rough times. I also want to thank my entire family and friends for their affection, support and compassion.

TABLE OF CONTENTS

ABSTRACT

The peer to peer computing paradigm has become a popular paradigm for deploying distributed applications. Examples: Kadmelia, Chord, Skype, Kazaa, Big Table. Multiagent systems have become a dominant paradigm within AI for deploying reasoning and analytics applications. Such applications are compute-intensive.

In disadvantaged networks the ad-hoc architecture is the most suitable one. Examples: military scenarios, disaster scenarios.

We combine the paradigms of peer-to-peer computing, multiagent systems, cloud computing, and ad-hoc networks to create the new paradigm of ad-hoc peer-to-peer mobile agent cloud (APMA cloud) that can provide the computing power of a cloud in "disadvantaged" regions (e.g., through RF using a router or GPRS)

- To this end we have designed and implemented a peer to peer operating system – PPOS that can leverage the computing power of such a cloud.

1. **INTRODUCTION**

Chapter1 includes the introduction which gives the overview of the entire thesis. Chapter 2 gives a description of the motivating part which is responsible for the development of the thesis. This chapter also explains the prior technologies also. It gives a brief overview of the structured peer to peer networks and also unstructured Peer to peer networks. Through this one can understand why there is a need for the design of new peer to peer OS.

Chapter3 gives the details of the differentiation with related work. Chapter4 gives the architecture of PPOS. Chapter 5 is all about the Distributed hash tables in detail. The distributed hash tables are very important because the operating system is based on the distributed key

Value pair .Chapter 6 deals about the scalaris. Chapter 7 is about the Virtual machine. Chapter 8 is about algorithms. Chapter 9 deals with the Experimental results compared with different operating systems. It follows with vita and references.

2. MOTIVATION FOR DOING THESIS UNDER THIS AREA.

2.1 INTRODUCTION

Cloud computing and multiprocessing are emerging computing paradigms that enable an average user to access unprecedented computing capacity. A cloud data center is a conglomeration of loosely–coupled servers together with management and application software that delivers software, platform, and infrastructure as services to clients.

To effectively leverage the computing power of a cloud, we need an operating system and a virtualization layer that interfaces between the user and the infrastructure. A cloud datacenter is usually centrally managed. Conventional clouds can be private, public, community, or hybrid. A cloud needs to be adaptable to dynamic load through elasticity.

The peer to peer computing paradigm has become a popular paradigm for deploying distributed applications. Examples: Kadmelia, Chord, Skype, Kazaa, Big Table. Multi agent systems have become a dominant paradigm within AI for deploying reasoning and analytics applications. Such applications are compute-intensive.

In disadvantaged networks the ad-hoc architecture is the most suitable one. Examples: military scenarios, disaster scenarios. We combine the paradigms of peer-to-peer computing, multi agent systems, cloud computing, and ad-hoc networks to create the new paradigm of ad-hoc peer-to-peer mobile agent cloud (APMA cloud) that can provide the computing power of a cloud in "disadvantaged" regions (e.g., through RF using a router or GPRS).To This end we have designed PPOS that can leverage the computing power of a cloud.

The figure1 shows that the motivations behind the work. In disastrous environment in order to run high performance video analytics we need to have access to a cloud. So we can create a cloud by means of this setup and can still run the high performance video analytics in disastrous

environments where don’t have to access to clouds like Microsoft azure and Amazon EC2.

Large industrial giants like Microsoft and Amazon provide cloud services to people according to their requirements. The requirements for a project might change whenever the user wants to change. But the cloud requirements are not subject to change due to the Constraints imposed by the industrial giants.

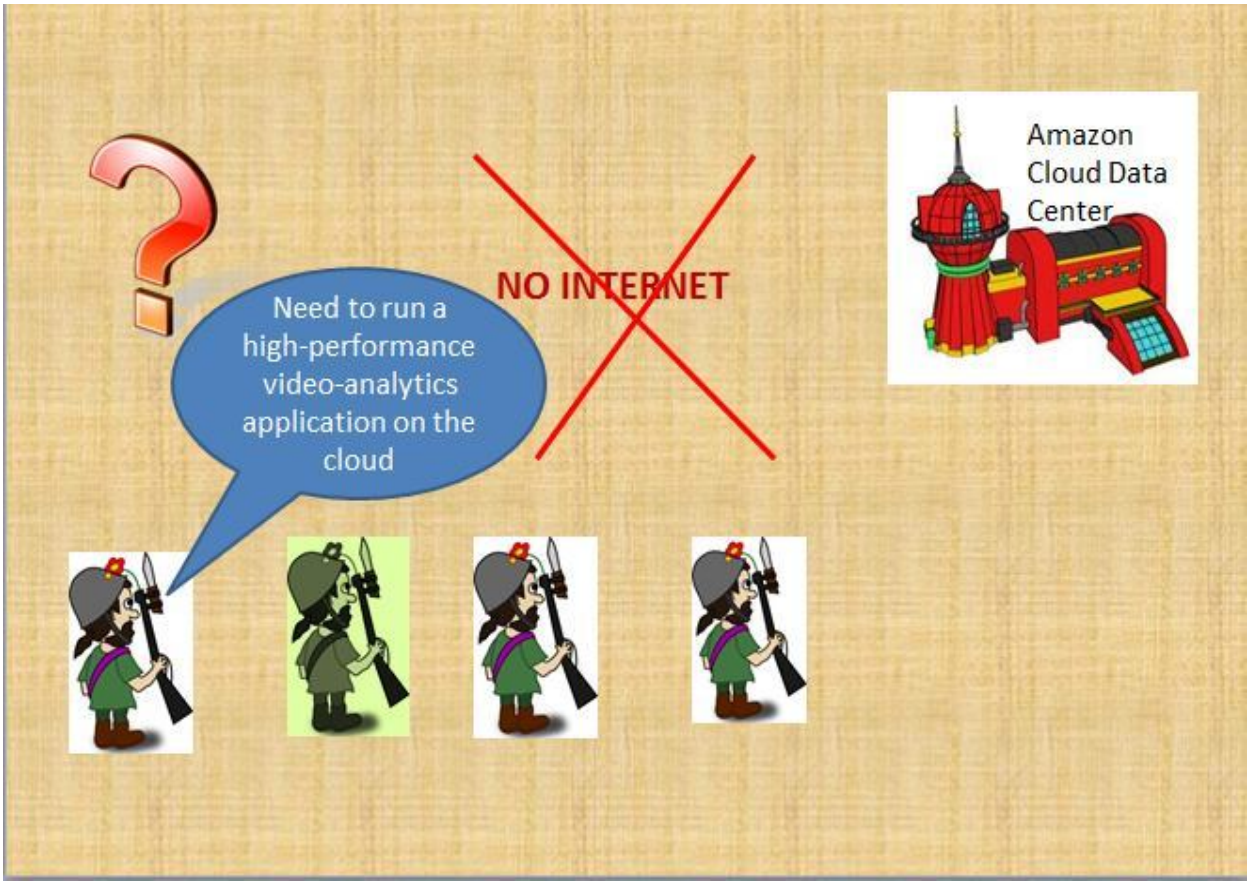


Figure1: Motivations For Doing Thesis Under This Area.

3. DIFFERENTIATION WITH RELATED WORK

Conventional grids [1] are managed centrally; they do not operate on disadvantaged network with nodes running agents communicating among one another; they provide a single point of failure: that of the management infrastructure.

Peer-to-peer grids like jalapeno [2] do not operate on disadvantaged ad-hoc mobile networks. Our architecture is asynchronous loosely coupled one. We provide an eventually consistent view of a single machine with a Unix-like interface and provide the user with a single agent-oriented programming model.

Rather than a shared memory model we use a Linda-like tuple space [3] in the form of a distributed global address space (DGAS).

PPOS concurrency semantics allows the dataflow [4] model of deterministic parallelism. The agent-oriented model of PPOS allows transactions: a computation can be split into transactions with each transaction executing on a location that also contains the required data. PPOS implements locks for mutually exclusive access of data. It has access control for managing use of resources and flow of information.

FOS [5] from MIT is a scalable operating system for multicore machines and clouds .As opposed to synchronous message passing model of FOS, PPOS Message-passing model is asynchronous; asynchrony helps improve Scalability. Fault-tolerance: as long as more than 50% of the nodes in the network are up, PPOS provides an eventually consistent view of a single machine. Partition-tolerance: PPOS is able to operate in disadvantaged networks where there are frequent network partitions. Elasticity: Within PPOS there is a group communication system that manages (eventual) consistency under agents joining and

leaving the network. As nodes leave and join, the system stabilizes to an eventually consistent state. This enables the system to be elastic while providing the user with the view of a single machine. Uniformity of time: Through an implementation of Mattern's global virtual time algorithm PPOS provides a uniform notion of time across the network.

VMware [6] provides a completely virtualized set of hardware to the operating system but it is limited by elasticity constraint and does not provide a single machine view with a uniform programming model to the user as opposed to PPOS.

Xtreem OS [7] is a grid operating system which involves the workloads which are non-interacting whereas PPOS is an agent based operating system where we can find the agents interacting with each other to complete the high performance computing.

4. ARCHITECTURE OF PPOS

4.1 APMA CLOUD ARCHITECTURE

Heterogeneous mobile devices communicate among each other (peer-to-peer) asynchronously through a distributed transactional key-value store Scalaris. Agents run on virtual machines deployed on these devices executing tasks. The PPOS operating system manages execution of agents on the APMA cloud controlling access to resources and providing the user with the view of a single eventually consistent machine and a single programming model. Devices can join and leave the cloud at any time**.**

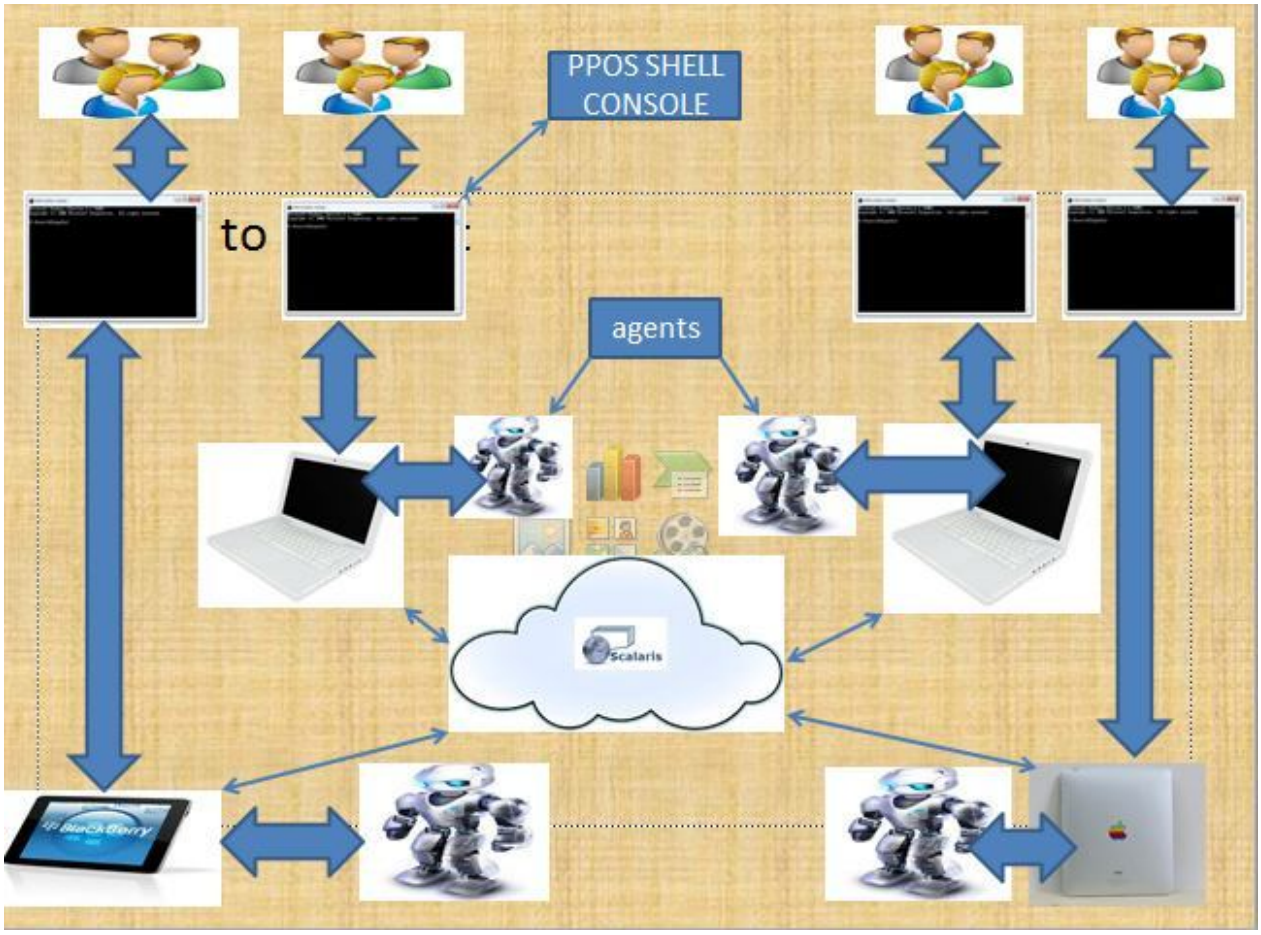


Figure2: The Apma Cloud Architecture

PPOS partitions tasks into agents that run on peer machines. Partitioning can be temporal as well as spatial. Agents can asynchronously communicate among each other . An agent schedules these agents on different machines. Currently, scheduling is based on heuristics and metadata Agents are both publishers and subscribers in contrast with the traditional client server model where servers produce and clients consume.

4.2 PPOS DESIGN PRINCIPLES

Agents implement tasks and run on virtual machines deployed on the hosts of the network. Multiple virtual machines can be tied to a single device. PPOS allows agents to migrate while execution from one host to another. PPOS allows hot swapping of one agent with another at runtime.

Agents communicate peer-to-peer asynchronously through the distributed transactional storage Scalaris. A group communication system within PPOS manages agent communication. Agent deployment and execution is managed by PPOS.

PPOS provide the user with the view of a single eventually consistent machine with a single file system and a single programming model and a Unix-like interface PPOS provides tolerance to network partitions and faults and controls access to resources PPOS allows interaction with the native operating systems of the hosts.

4.3 PPOS ARCHITECTURE DESCRIPTION

Users interact with PPOS through the microkernel scheduling and deploying tasks and accessing resources. The microkernel itself is a set of agents that are replicated on each host. Operating system services like file services, deploying and unloading agents, measurements, etc., are provided by a set of agents.

The interface between the users and the kernel constitutes of Unix-like commands, compilers and interpreters, and system libraries. System libraries consist of agent byte codes that implement certain functionalities.

The microkernel agents and the deployed agents communicate with each other through the Scalaris distributed key-value store. The kernel agents ensure that a snapshot of the state of the system is always stored in Scalaris. The Paxos algorithm maintains eventual consistency even in case of faults and network partitions.

4.4 UNDERLYING TECHNOLOGIES

- A Java-based implementation on the top of the Scalaris distributed key-value store.
- The PPOS virtual machine is built on the top of the Java Virtual Machine
- User can interact with PPOS through the PPOS shell console.

4.5 PAXOS AND EVENTUAL CONSISTENCY

Brewers CAP theorem states that it is impossible for a distributed computing system to provide Consistency, Availability, and Partition-tolerance at the same time we settle for eventual consistency while maintaining availability and partition-tolerance. Paxos is a family of protocols for solving the consensus problem in network of unreliable processors. It can make progress using 2f+1 processers even if ‘f’ processers fail among the group simultaneously. PAXOS has the following the roles for processors to eventually carry out a task. They are client, acceptor, proposer, learner, and leader.

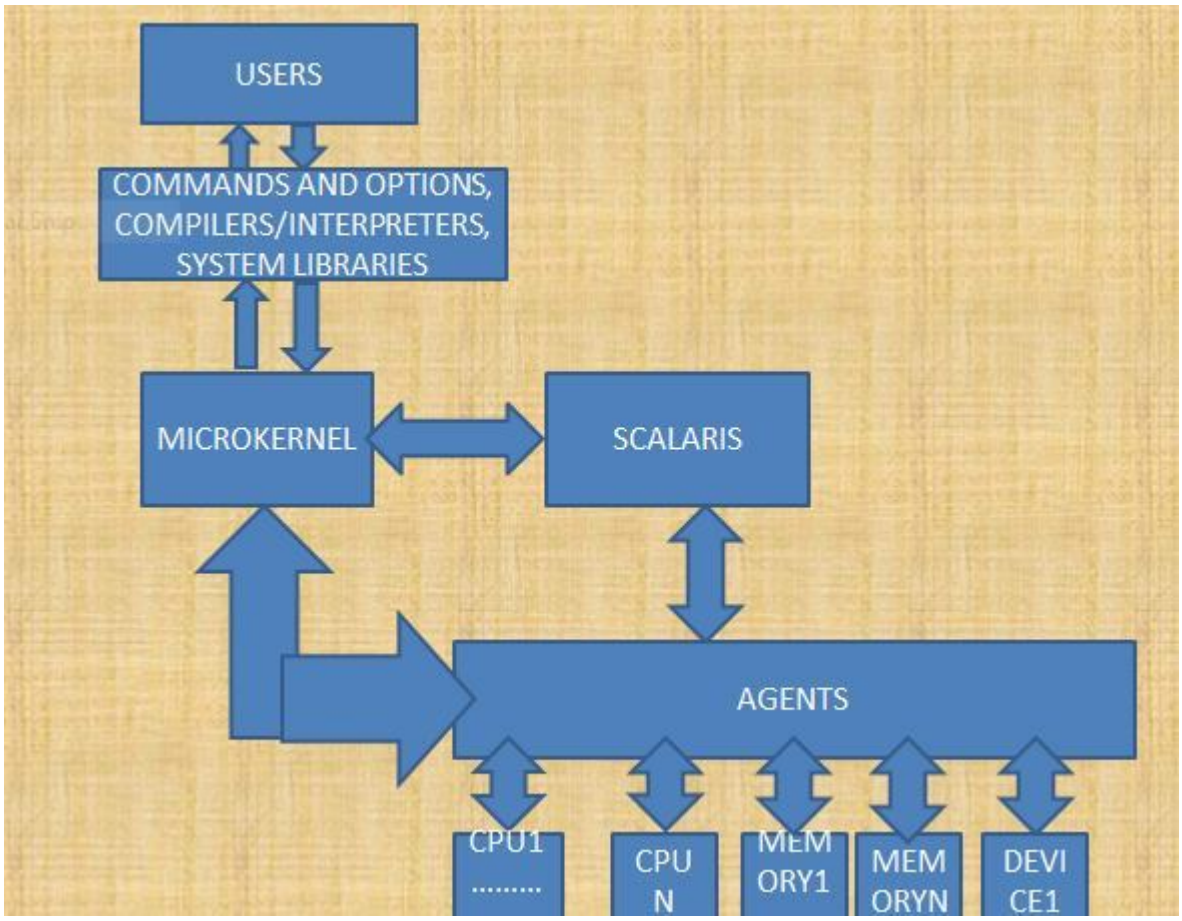


Figure3: Ppos Architecture

5. DISTRIBUTED HASH TABLES [8]

5.1 INTRODUCTION [8]

In our thesis we are mostly concerned about the distributed has tables because they deal with the Structured peer to peer networks .A distributed hash table (DHT)[8] is a reliable, scalable, wide area Data storage system that gives relief to programmers from many of applications of building a Complicated distributed system. DHTs store blocks of data on hundreds or thousands of Machines connected to the Internets, replicate the data for reliability, and quickly locate data. Despite running over high latency, wide area links. The DHT [8] addresses problems of locating Data and replicating it for exactness.

DHT [8] provides a generic interface, which makes it easy for a wide variety of applications to adopt DHT [8]s as a storage substrate: put stores data in the system under a key; get retrieves the data. The key value pairs are stored in the DHT [8] and any participating node can efficiently retrieve the value Associated with a given key. The mapping from keys to values is distributed among the nodes, in such a Way that a change in the set of participants causes a minimal amount of disruption. This allows DHT [8]s to scale to extremely large numbers of nodes and to handle continual node arrivals, departures, and failures.

Distributed hash tables [8] have the following properties:

1. Decentralization: there is no central coordination among the nodes and all the nodes form the system collectively.

2. Fault tolerance: The system must not lose the reliability with the nodes joining the system or leaving the system. .

3. Scalability: the system should function in spite of thousands of nodes and millions of

nodes added to the system.

5.2 STRUCTURE OF DISTRIBUTED HASH TABLES [8]

The structure of distributed hash tables uses key space portioning. The key space portioning gives the ownership of key space to all nodes. The nodes are connected by means of the overlay Network allowing finding the owner of any given key among the key space. Suppose if the key space consists of 160 bit strings and to store a file with given file name and its Data a 160 bit key is generated160-bit key *k*, and a Message *put* (k, *data*) is sent to any node participating in the DHT. The message is moved from one node to another through the overlay network until it reaches the single node responsible for key *k* as specified by the key space partitioning. That node then stores the key and the data. Any other client can then get the contents of the file by again Hashing *filename* to produce *k* and querying any DHT node to find the data associated with *k* with a Message *gets* (*k*).

5.3 OVERLAY NETWORK [8]:

Each node maintains a set of links to all other nodes. Each node forms its neighbors according to the Network topology and all the nodes form the network. For any key *k*, each node either has a node ID that owns *k* or has a link to a node whose node ID is *closer* to *k*, in terms of the key space distance defined above. It is then easy to route a message to the owner of any key *k* using the following the greedy Algorithm, forward the message to the neighbor whose ID is closest to *k*. When there is no such neighbor, then we must have arrived at the closest node, which is the owner of *k* as defined above. This is called the key based routing. The two important constraints on the topology that are to be maintained are maximum number of hops in any network must be low so that requests complete quickly and the maximum number of neighbors on any node is low so that maintenance over head is not excessive.

5.4 ALGORITHMS FOR OVERLAY NETWORK [8]

There are many algorithms that exploit the structure of the overlay network for sending a message to all nodes, or a subset of nodes, in a DHT [8].

6. SCALARIS

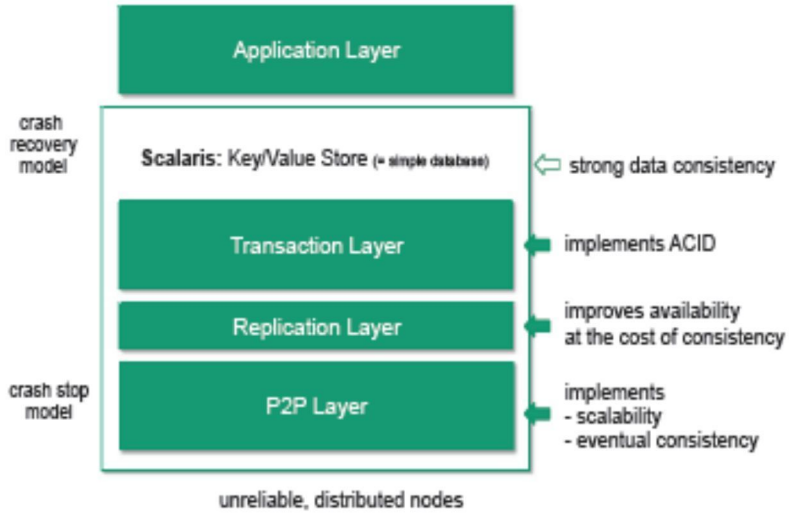
6.1 INTRODUCTION

Scalaris [9] is a distributed key-value store for the large scale Web 2.0 services, implemented in Erlang programming language .It is concurrent and garbage collected programming language. Scalaris implements the four ACID properties which are very much required in a distributed environment. It has the high capability of transactional support for data consistency in case of concurrent data operations and node failures and also the network problems.

Major e commerce firms require highly concurrent access to distributed data and the high end operations must be done in real quick and it should be concurrent. Scalaris is capable of scaling to 'n' number of systems which supports consistent replication and fast transactions even in case of a system failure. No computer is neither a client nor a server, any data is available across multiple systems and transactions can be initiated through key value pair mechanism. Every node acts as a peer to another node in the system and new nodes can be scaled from few servers to thousands of them without any disruption. New nodes can be added or deleted without disturbing the transactions. In Scalaris if certain number of connected each other they are called as the Scalaris ring .Each node in the ring is recognized by the other node using the computer name. We provide each computer names in the properties file namely Scalaris. Properties which uses this file to recognize the nodes to make the transactions.

6.2 ARCHITECUTREOF SCALARIS

The architecture consists of 3 layers [9]. They are –



Source: *portal.acm.org/ft_gateway.cfm?id=1411280&type=pdf*

P2P layer – The key-value pairs in the nodes in the scalaris ring can be retrieved using the structured overlay protocol in the bottom layer. The keys are stored in lexographical order. This gives the advantage of writing different type of queries.

Replication layer - This is the layer which takes care of the replication of the nodes in a ring. This is responsible for adding the nodes from '1' to 'n' in a Scalaris ring without interrupting the service performance and it implements the ACID properties which are very necessary for the concurrent write operations. This layer also uses a consensus protocol called "paxos" which is required for the low communication overhead. This is for implementing fault-tolerant distributed databases.

Transaction layer - This is the top layer which actually hosts the applications like Web 2.0 and also manages the transactions across the multiple nodes in a ring. This can be used for these-commerce applications like online shopping, banking and data sharing.

The main advantage of Scalaris is that it provides

1. Reliable transactions
2. Consistency
3. Implements ACID properties
4. Scalability
5. Data replication
6. Fault tolerance
7. Data distribution

6.3 SCALARIS AND ERLANG

The main advantage of using Erlang is its usage of concurrency and it develops the process to write an Erlang application which is used by powerful primitives. Another advantage of Erlang is the way concurrency supports the error handling. When a process crashes abruptly, it sends a message to the main controlling process which is capable of taking the actions. So because of this error handling there is a fault tolerance complexion of the code.

7. VIRTUAL MACHINE

7.1 INTRODUCTION:

A computer application which is used to create a virtual environment refers to a process called virtualization [10]. It allows the user to see the network infrastructure through aggregation process. If a user wants to operate the software located on any computer then he could use a virtual machine. Through this process of virtualization there is a possibility of running multiple operating systems on single computer platform.

7.2 KINDS [10]

The virtual machines [10] are of different kinds. But the term mostly used to refer to a hardware virtual machines software is called as virtual machine monitor. It is also called a hypervisor. This type of software makes it possible for running multiple identical executions on one computer. Any one of these executions would run as an operating system. Through this one could run multiple applications on different operating systems. The main advantage offered by the virtual machine software is that the users could boot and restart their machines as hardware initialization is not required. Virtual machine [10] is a kind of personal machine with all the required functional hardware different from the original machine.

7.2.1SYSTEM VIRTUAL MACHINE [10]:

Multiple OS environments can co-exist on the same computer, in strong isolation from each other.

The virtual machine can provide instruction set architecture (ISA) that is somewhat different from that of the real machine.

Application provisioning, maintenance, high availability and disaster recovery.

The desire to run multiple operating systems was the original motivation for virtual machines, as it allowed time-sharing a single computer between several single-tasking Operation Systems. In some respects, a system virtual machine [10] can be considered a generalization of the concept virtual memory [10] that historically preceded it. IBM's CP/IMS, the first systems to allow full virtualization [10], implemented time sharing by providing each user with a single-user operating system, the CMS.

Unlike virtual memory [10], a system virtual machine allowed the user to use privileged instructions in their code. This approach had certain advantages, for instance it allowed users to add input/output devices not allowed by the standard system.

The main intention of designing a virtual machine is that to run several operating systems on a single machine. .It allows sharing a single machine between several single tasked operating systems .It saves a lot of time as it allows time sharing. The concept of virtual machine [10] takes its roots from the concept of virtual memory.

Systems allow full virtualization [10] by providing each user a single user operating system. Unlike virtual memory virtual machine allows the user to use their own instructions in their code. This makes it feasible for the users to add any input and output devices which are generally not allowed by the standard system.

The guest operating system whatever installed does not have to be compatible with hardware. It makes it possible to run different operating system on same computer. The use of virtual machine [10] to support different guest operating systems is being popular in embedded Systems. A typical use would be to support a real time operating system at the same time as a high level OS such as Linux or windows. Virtual machines offer other advantages for OS development which includes

debugging access and reboots.

7.2.2. PROCESS VIRTUAL MACHINE [10]

An application virtual machine such as a process virtual machine [10] runs like an ordinary application inside the host OS and supports only a single process. The main feature of this kind of application is that it's created when the process starts and terminates when it exits. The purpose is mainly to provide a programming environment that is free from any platform and extracts the details of the operating system beneath it or the hardware. It allows the program to execute in the same way on any platform.

A process VM [10] provides a high level abstraction and can be implemented using an interpreter. Concerned with performance, it is comparable with that of compiled programming languages and can be achieved by the use of just in time compilation.

This type of VM [10] became popular with Java programming language. This can be implemented by means of Java virtual machine. Other examples include Parrot virtual machine which serves as an abstraction layer for several interpreted languages which run on a VM [10] called common language runtime.

A specific case of a process virtual machine [10] is that it is abstract with the Communication mechanisms of a computer cluster. In this virtual machine there is no single process, but one process per machine in the cluster. They are basically designed to reduce the labor of the Programmer in creating parallel applications by letting him concentrate on algorithms rather than communication schema with the interconnect and the OS. The fact that the communication among these machines in a cluster is not hidden and the entire cluster is represented as a single parallel machine.

Unlike other process VM [10]'s these systems don't have a specific programming language.

They are embedded in existing language and provide binding for several languages. Typically such a system provides links for several programming language. Examples are parallel virtual machine and message passing interface, they can be considered.

8. ALGORITHMS AND ANALYSIS

8.1 LS COMMAND

The purpose of this command is to display the contents of the directory. The ls command writes to the output the contents of the directory. We have written an algorithm to display the contents of the directory.

The LS algorithm has been used to retrieve the list of the files using the command LS. We will retrieve all the files corresponding to the directory. The owner of the file will be one who has uploaded that. The algorithm will traverse the list for all entries which are owned by particular owner. We will obtain the list of all the entries owned by different owners. We kept all these elements in a new list called output and we sorted the output according to the alphabetical order and printed the output.

Ls command requires us to upload the files to the scalaris before we could retrieve them. Different files of different sizes have been uploaded. The file sizes were namely 25mb, 1mb, 2mb, 699mb…

We must be able to retrieve the information available with these files as per our algorithm. In our cloud of two computers we have uploaded the files even from the second computer. The screenshot corresponding to that particular files upload has been kept in the following pages for reference.

The figure 5 shows the list of the files uploaded from one pc and the second figure shows the list of files uploaded from the second pc.

8.1.1 LS ALGORITHM

1. **Initially a key in Scalaris called dir., is created which represents the root folder /. The value associated with this key are the top level files/folders contained in the**

root.

1. **Corresponding to each folder is a key whose value represents the top level files/folders contained in that folder**
2. **This value is associated with this key is formed by concatenating the files/folders along with the computer names using an operator @ for separation and other attributes as they are created/modified including a special binary attribute file/folder.**

2. **Traverse the tree and print the list of nodes using split to separate the nodes at a particular level of the tree.**
3. **Sort the files according to alphabetical order and display the results.**

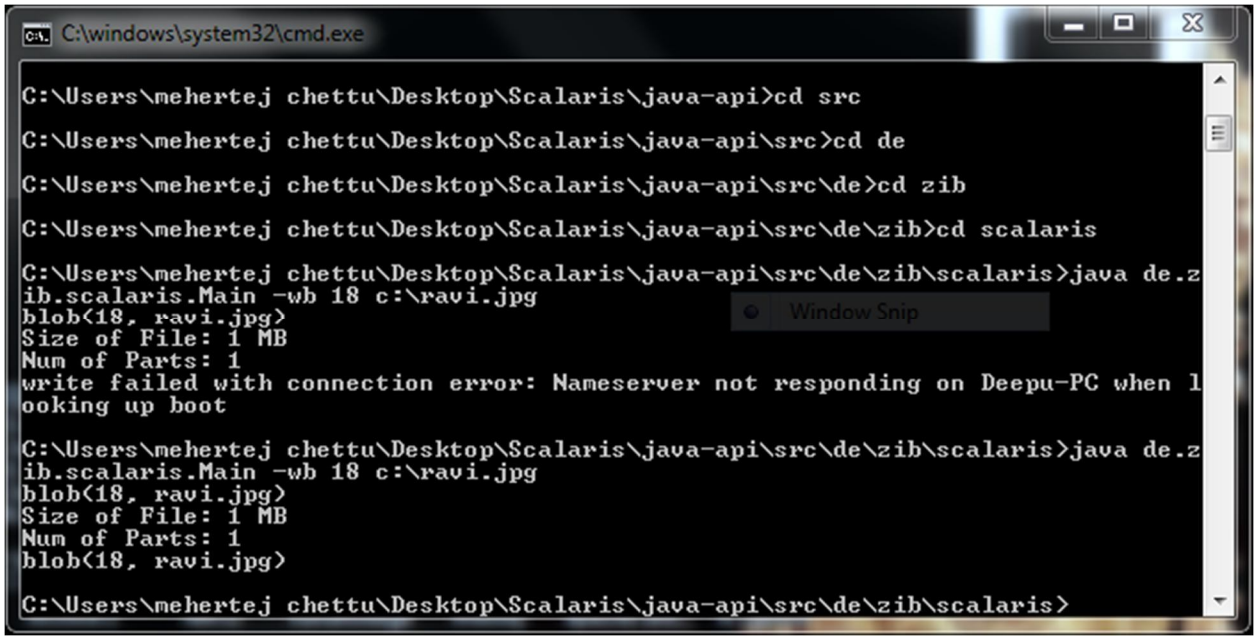


Figure4: Snapshot with Write Blob.

We obtained the above output on both the computers in our cloud. The output was same on both the computers. We made further improvements to our LS command and obtained the list of the files corresponding to a particular letter and particular folder. The figures 5 and 6 show us the output.

In the figure7 we can observe that the list of the files starting with a, r and also the list of all files.

8.2 MKDIR COMMAND

8.2.1 MKDIR ALGORITHM

1) Check for existing folder with name provided.

2) If not exists create a folder with name provided.

Mkdir checks the name for existing folder .If there is no existing folder with the name provided then it creates a new folder with the name.

8.3 CP COMMAND

The **cp** command copies the source file specified by the *Source.*

File parameter to the destination file specified by the *Target.*

File parameter. If the target file exists, **CP** overwrites the contents, but the mode, owner, and group associated with it are not changed.

8.3.1 CP ALGORITHM

1) Check for file exists. Throw error if file not exists.

2) Check for folder exists. Throw error if folder not exists.

3) Copy file from one folder to other folder provided.

The CP command checks for whether the file exists. If the file does not exist, then it would result in and error. Then it would check for whether the folder exists .The folder is nothing but the folder which is created using MKDIR command. The CP command copies the file from one folder to another folder provided.

We implemented two types of copy.

1. Physical copy
2. Logical copy

1. Physical copy: This kind of copy what we have implanted copies the files from one machine to another machine in the cloud. This has taken considerably long time for us to copy a 2 GB file from one machine to another machine on the cloud. The copy of the file has taken place when both the computers are connected to a same network. We were able to access the file on other desktop also.

2. Logical copy: This includes copying the files uploaded from the desktop to scalaris and copying from one virtual directory to another directory. In the above screen shot we were able to move the contents of the directory meka to one more directory chettu. This is called logical copy. Logical copy did not take us much time because everything that was moved was moving logical from one virtual folder to another virtual folder.

8.3.2CP TECHNIQUE:

- **The syntax of cp is**
  - **Cp source folder/file destination folder/file1.**

1. We use source folder name and the destination folder name as the two keys.

2. Using the key corresponding to the source folder name we can get the list of the files and split the files accordingly as described in LS algorithm and search for the file specified in the source. If the file exists, its name is concatenated to the value specified by the destination folder key.

3. Through this we can get the file copied from one folder to another folder. If the file doesn't exist on source folder the console will throw an error.

4. If the file gets copied from the source folder to the destination folder then it displays a message "FILE COPIED".

8.4 RM COMMAND

This command is used to remove the contents from a specified directory. The following algorithm has been used to remove the contents of the directory.

8.4.1 RM ALGORITHM

1) Check for file exists. Throw error if file not exists.

2) Check if permissions available to delete. If not throw error.

3) Delete file from the folder.

The RM command checks for whether the file exists. It will throw and error if the file doesn't exist. The command will also check if the permissions are available for the particular folder to delete. If not it will throw an error. We set the permissions in such a way like that the files which were created and uploaded from the owner PC will only be able to remove the files. If we try to remove the files from any other PC, it will result in permission being denied.

8.4.2 RM TECHNIQUE

The syntax for rm algorithm is –rm folder/file.

1. The rm algorithm works on the fact that the folder (that is the key) has been given and corresponding file associated with file is also given (if a file needs to be deleted).

2. The list of the files associated with that particular key can be obtained and we will search for the file that has to be removed.

3. The files are always associated with its owner that is the host name from which was originally taken from.

4. If the file found with the associated computer name is different from the host machine's name then the file cannot be removed and it displays a message "PERMISSION DENIED".

5. If the file found with the associated computer name is same as that of the host machine name then the file can be removed as the owner himself is removing the file.

6. If the file is not found at all then the console will throw an error.

8.5 LOADING AGENTS

Agent code is stored an agent repository. A microkernel load manager agent downloads the byte code from the server and stores it in Scalaris. It determines on which core the agent will run on based on user-provided metadata or using heuristics. Through Scalaris it communicates with the agent loader of the corresponding machine instructing it to load the agent whose code is available under a particular key in Scalaris. The agent loader of the target machine loads the agent on that machine and updates a list of loaded agents maintained under a key in Scalaris.

8.5.1 UNLOADING AGENTS

Click the icon corresponding to agents in the java applet window and it will correspondingly remove the agents in PS. The agent unloader of the target machine unloads the agent on that machine and updates the list of loaded agents maintained under a key in scalaris.

8.5.2 CAUTION

If you are getting a list of agents redundantly and the class files getting loaded redundantly, then simply remove the list of agents loaded on the group.commnucation properties located in the build folder of grcommn project. This will give the correct output without any errors.

8.6 PS ALGORITHM

The syntax of PS command is –ps.

This command will obtain the list of all the agents loaded on every core through the key for the list of loaded agents maintained in Scalaris

The agent loader will also get the owner name along with the size of the agent in bytes.

We have installed a jetty server and kept the jar files in webapps folder of the scalaris folder in jetty server distribution. The jar files are loaded into the system by providing a URL which hosts those jar files in the computer. The jar files are accessed from the system by that URL and the corresponding class files are extracted and written to scalaris.PS command gives the list of all processes(class files) running on all computers in the group. We could do this by means of reliable group communication and SCALARIS.

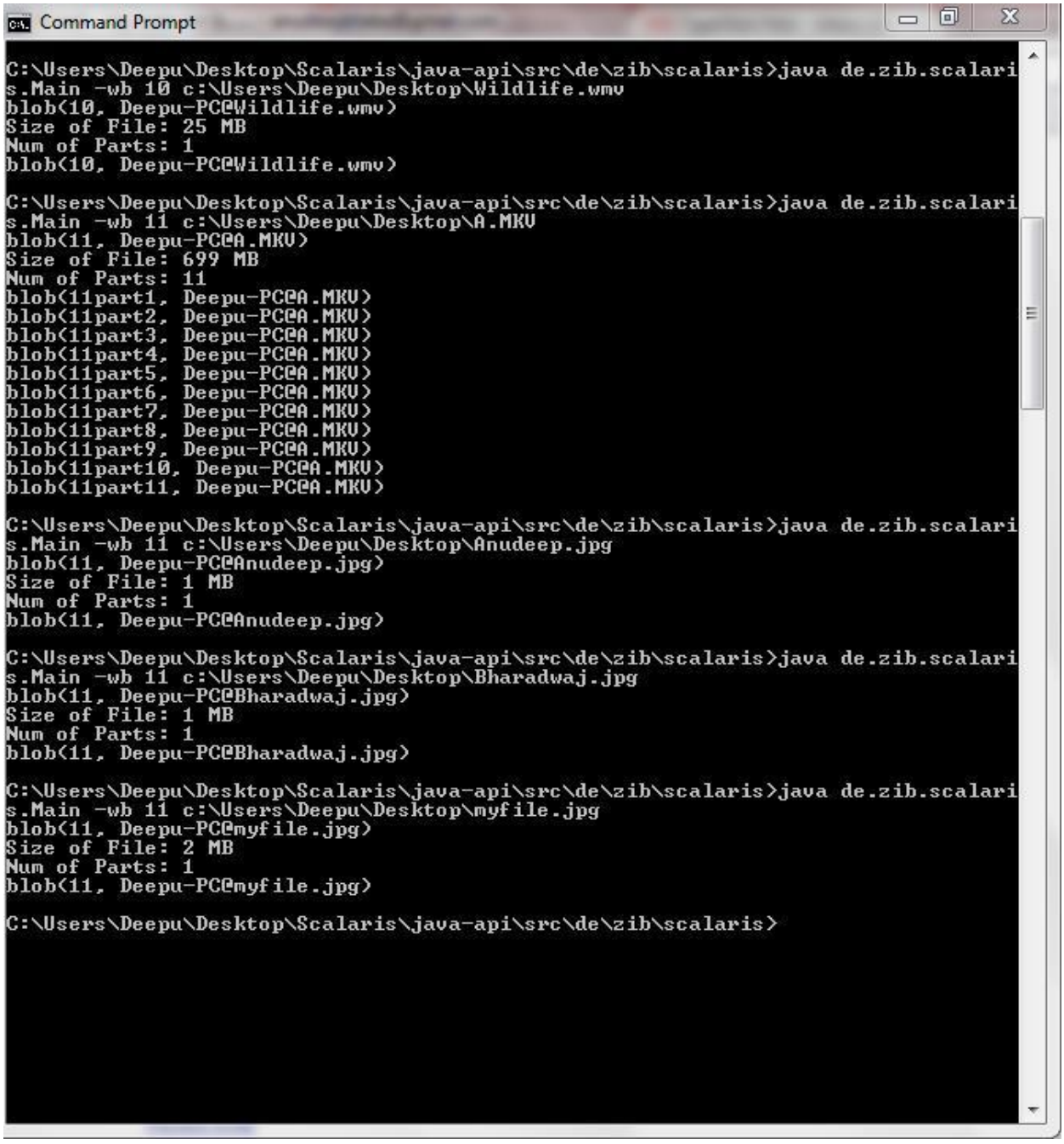


Figure5: Snapshot For Files Uploaded

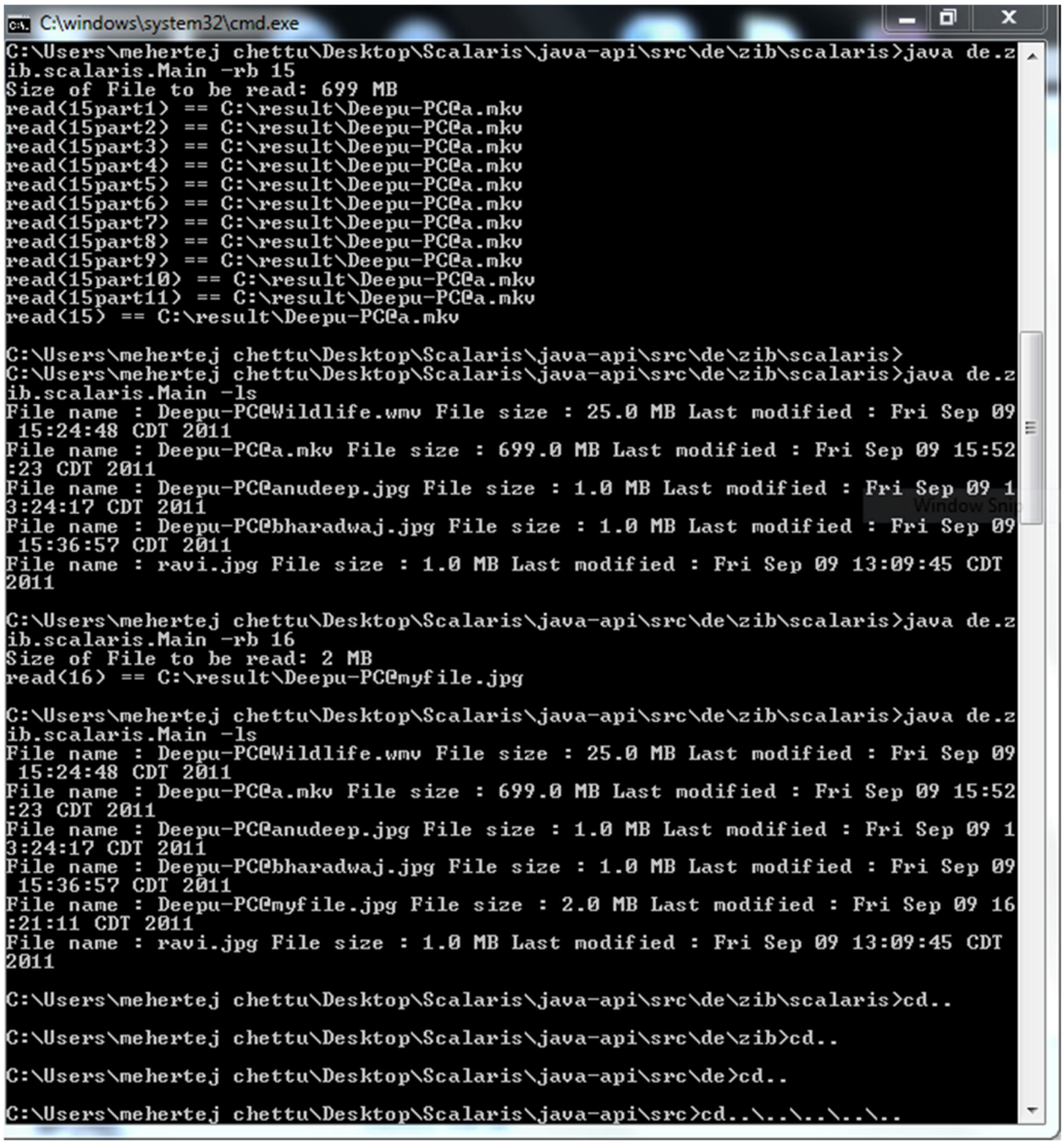


Figure6: Snapshot For Ls.

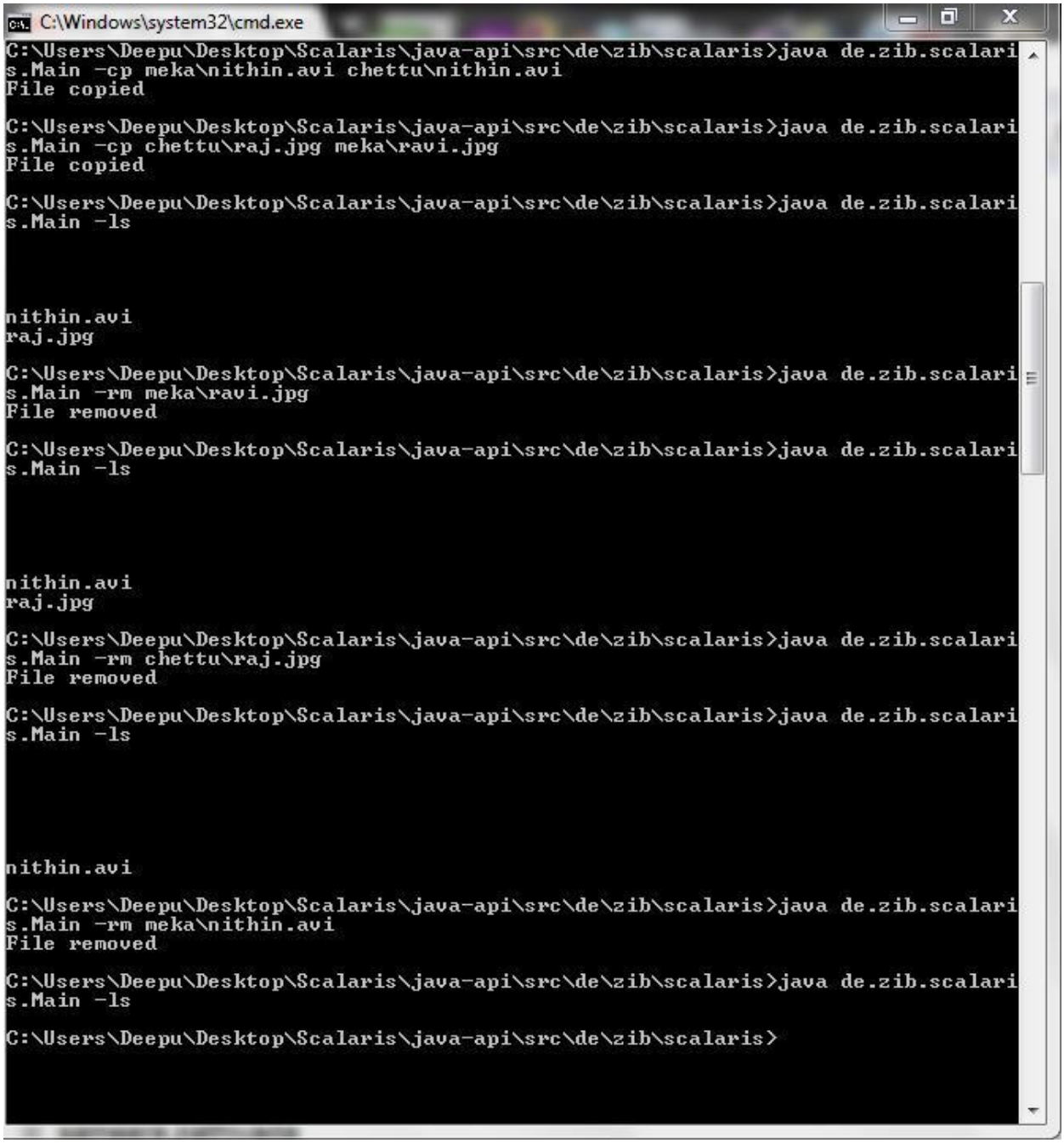


Figure7: Snapshot Of Lsstar.

Figure 8: Snapshot For Mkdir

Figure 9: Snapshot For Cp.

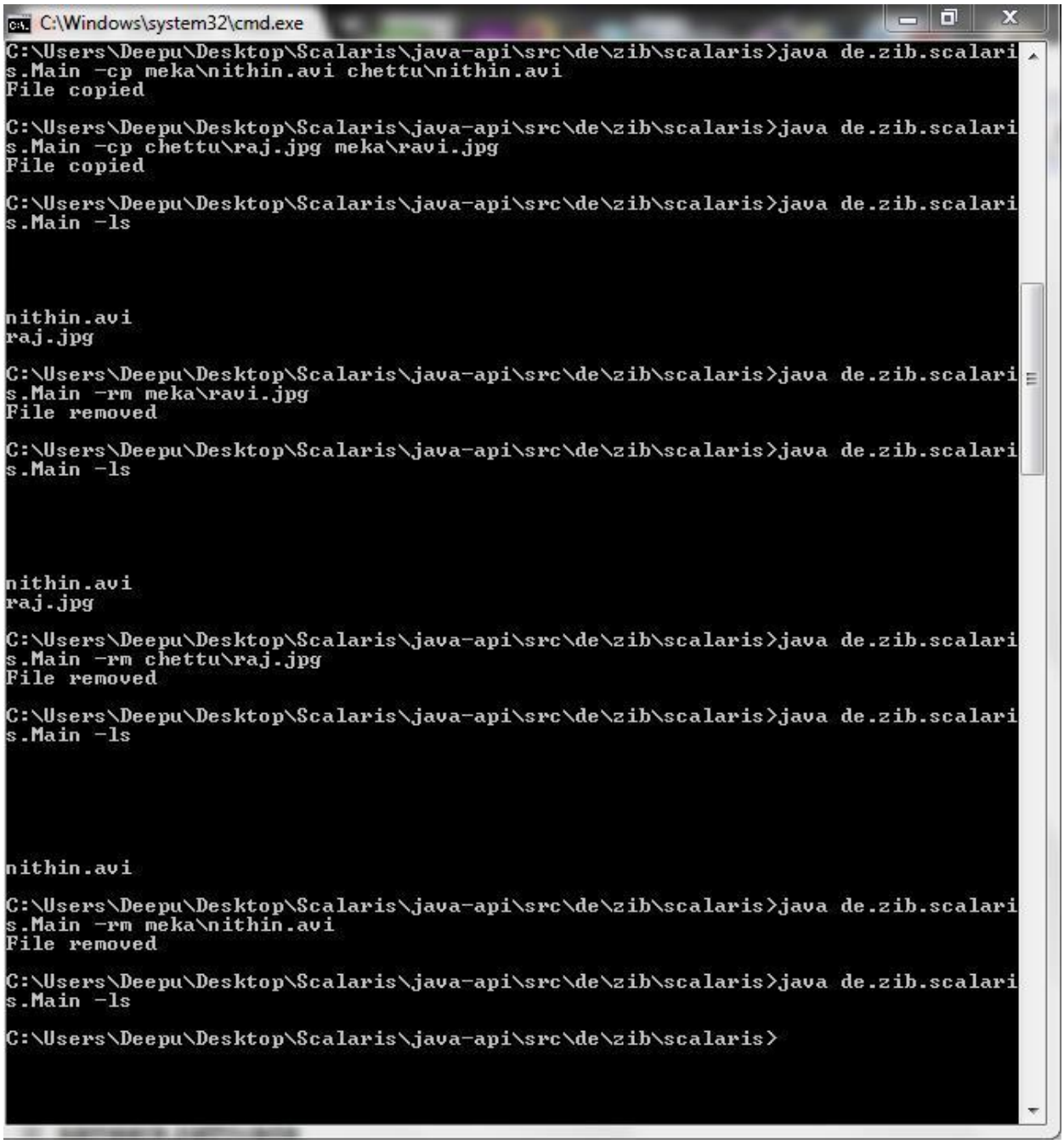


Figure 10: Snapshot For Ls, Cp, Rm.

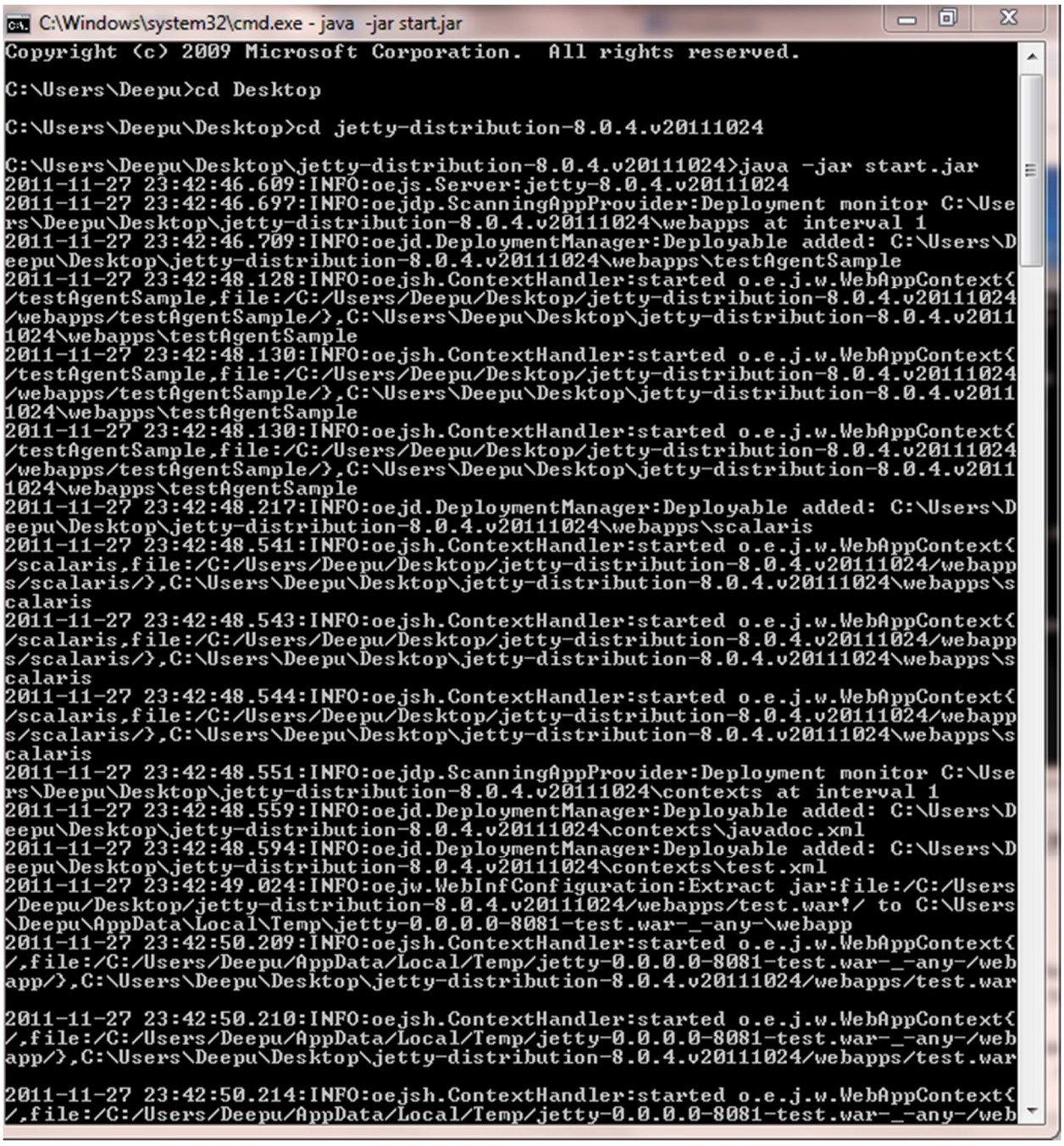


Figure 11: Snapshot For Jetty

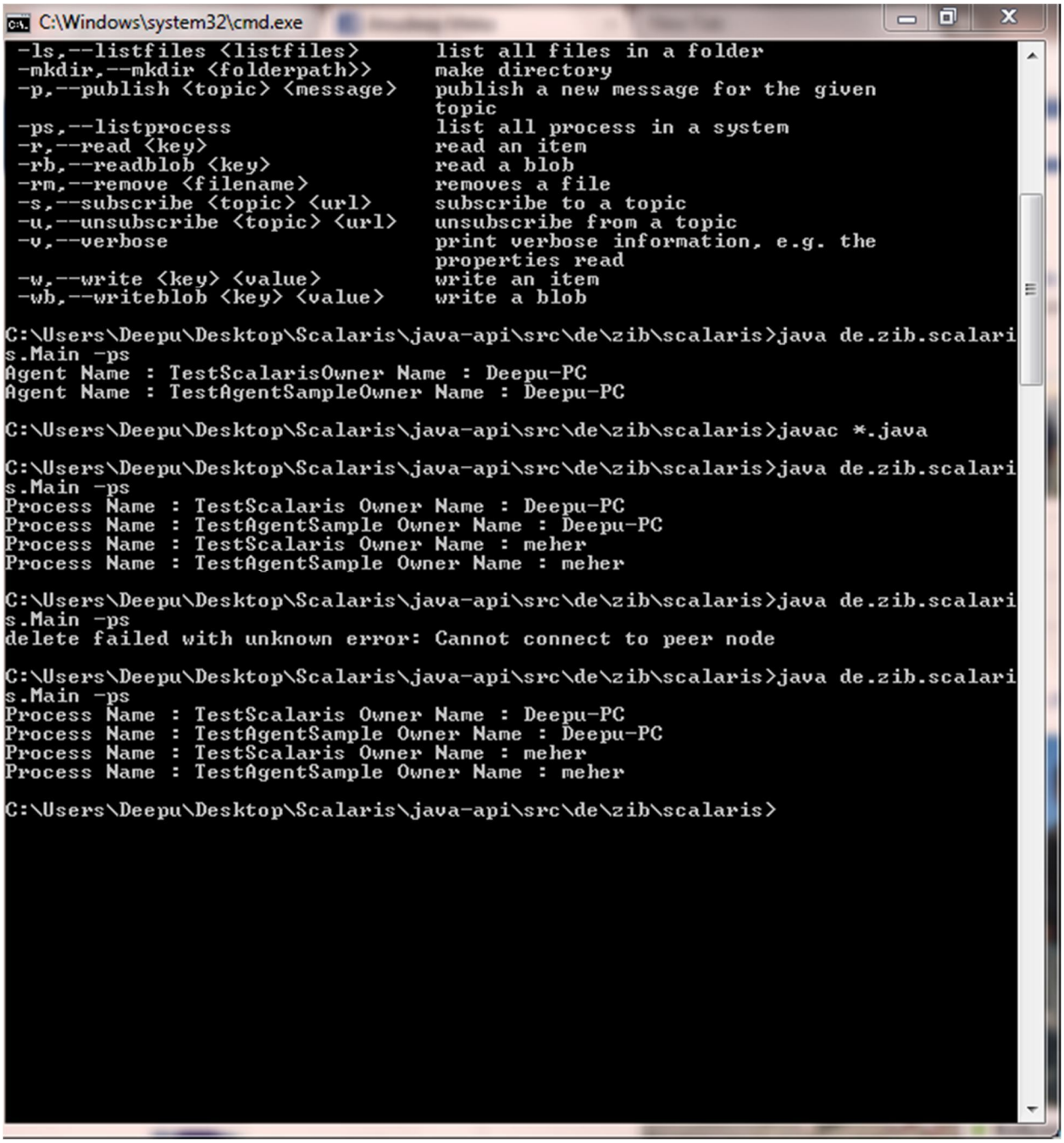


Figure 12: Snapshot For Ps.

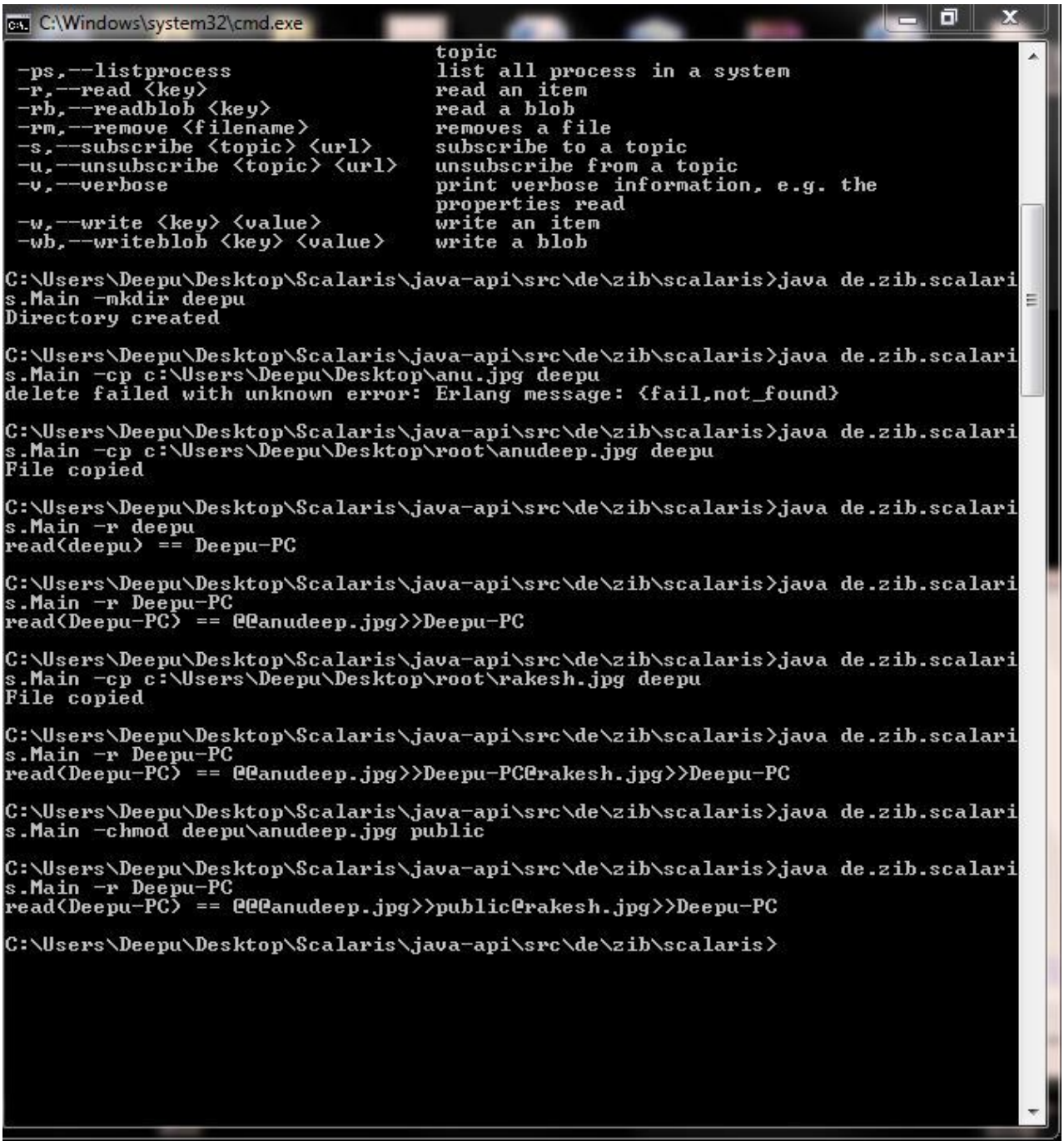


Figure 13: Snapshot for Chmod

9. EXPERIMENTAL RESULTS AND PERFORMANCE ANALYSIS.

The analysis of the above research work has been done with one more Operating system Linux. The following results have been obtained and we have plotted them graphically to show the operating system results. The graph is drawn for PPOS AND LINUX. The time taken to execute the LS command in PPOS AND LINUX has been plotted as a graph.

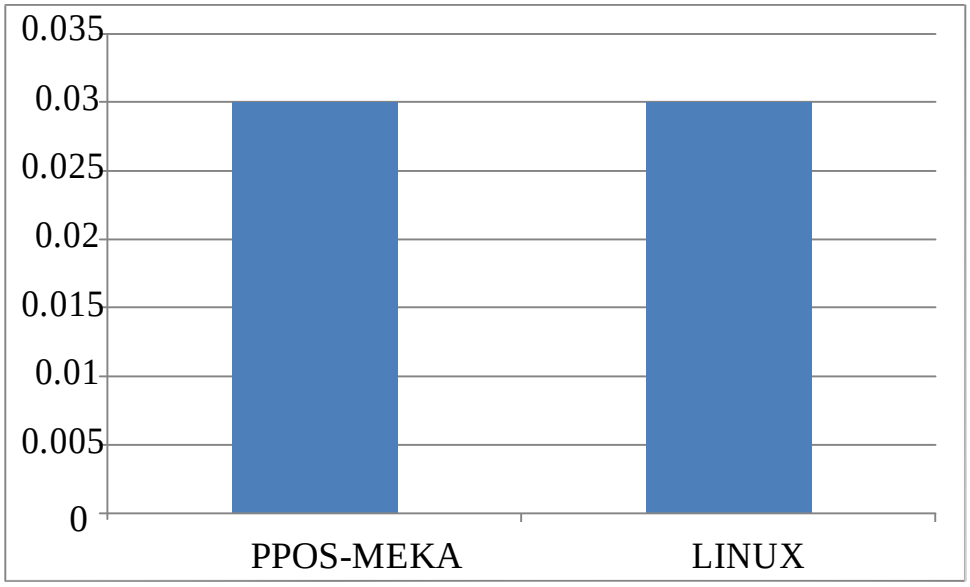


Figure 14: Graph Showing LS.

The y-axis indicates time in millisecond and time on x-axis indicates the operating system. From the above figure we observe that PPOS-MEKA working time is exactly same on a number of computers in cloud with that of LINUX.

The next graph is plotted again between PPOS-MEKA and LINUX for the CP command and the following graph shows that again the time.

.

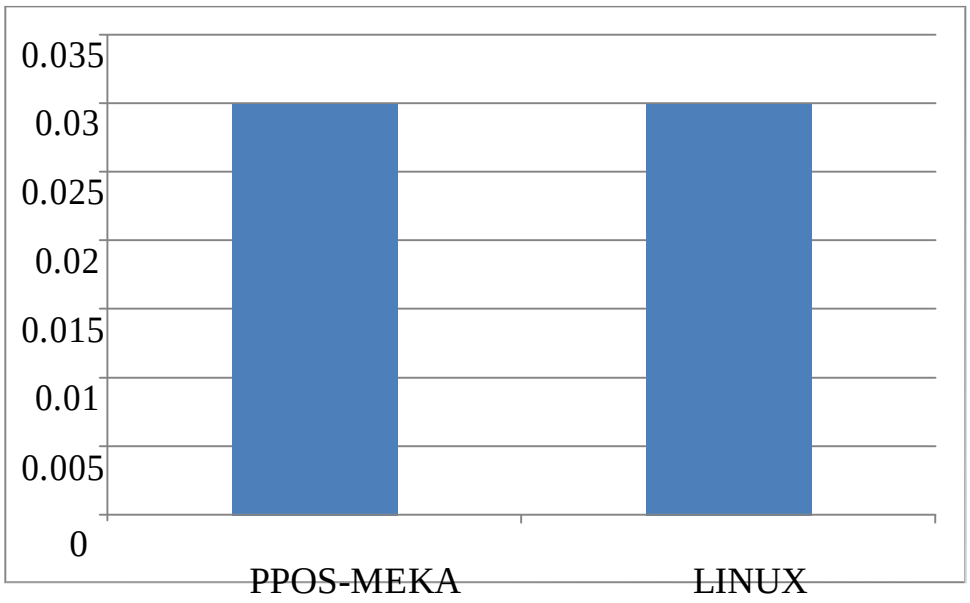


Figure 15: Graph Showing Cp.

The next graph is plotted between two operating systems and the graph indicates that the time required is one and the same for RM and MKDIR.

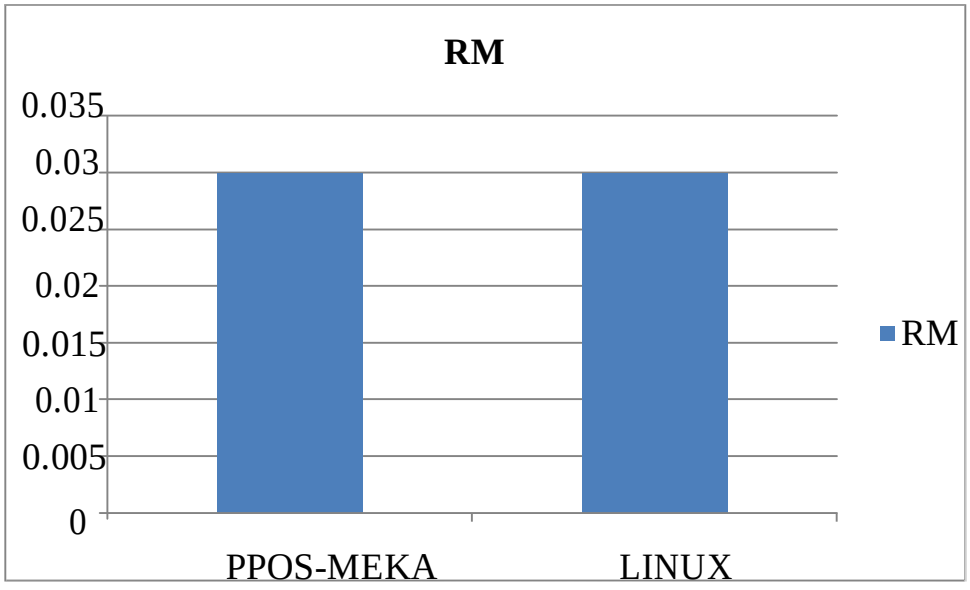


Figure 16: Graph Showing Rm.

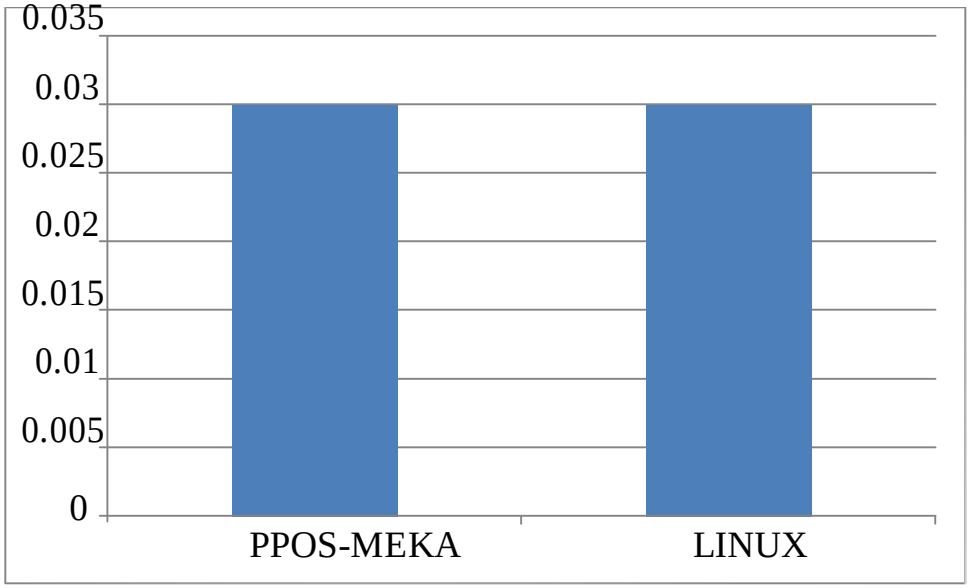


Figure 17: Graph Showing Mkdir

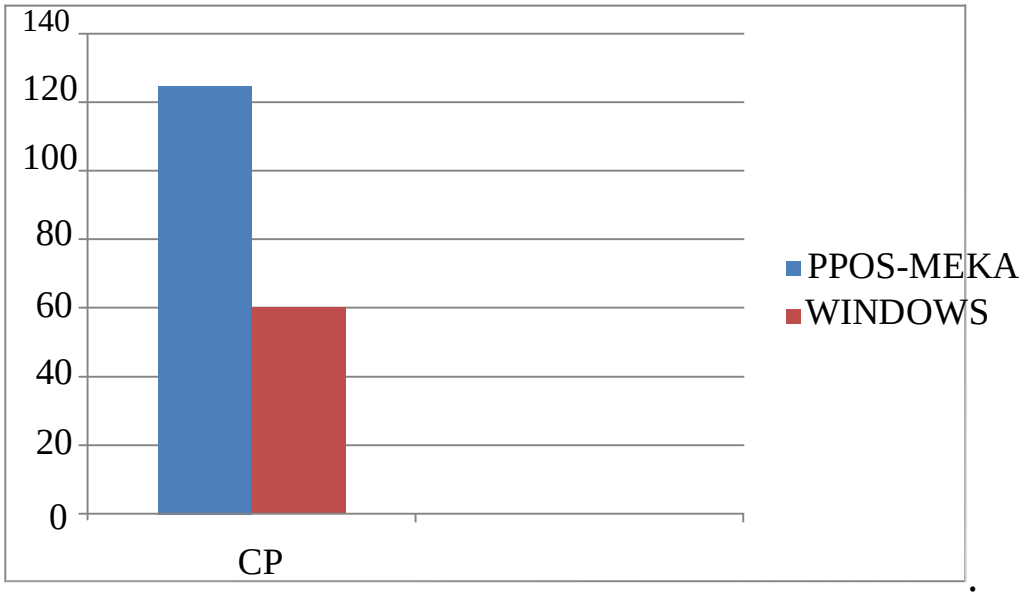


Figure 18: Graph Showing Physical Copy

PHYSICAL COPY: We made a copy of 2gb file between two machines using windows network and took us 60 minutes where as it took 125 min to transfer the file from one machine to the other using wireless router . LOGICAL COPY: The time required for logical copy of the files is

so easy and we have taken a shift to logical copy and copy of the file was so easy and it took less than 9 sec. The performance levels have been considerably proven to be good when we have conducted these experiments and results were successful.

The graph below shows the comparison levels of the operating systems concerned to PS.

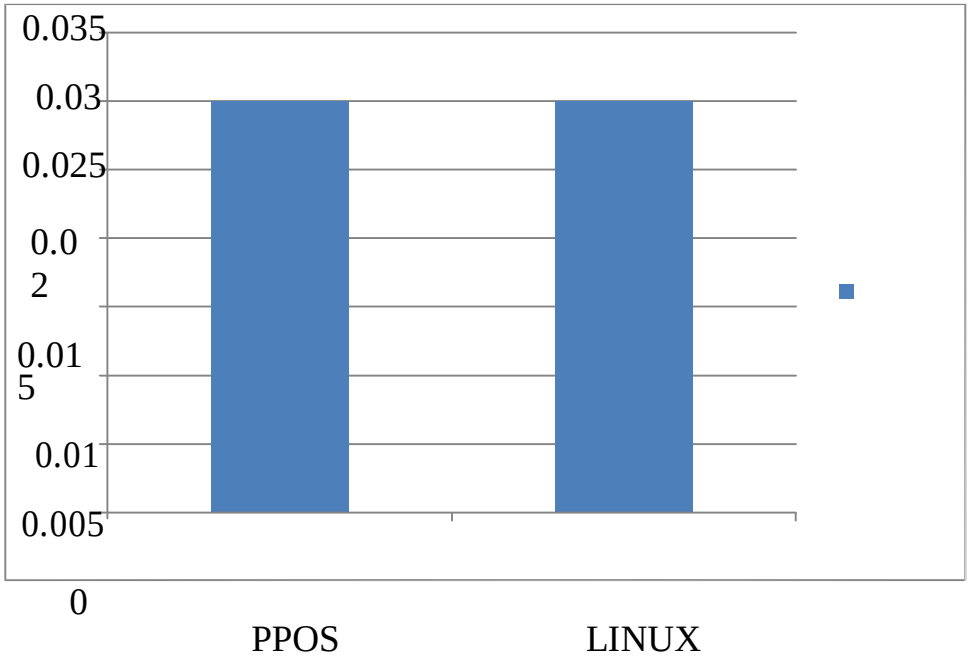


Figure 19: Graph showing Ps.

10. CONCLUSIONS

We present the design and implementation of a prototype peer-to-peer operating system that provides the user with an eventually-consistent view of a single machine with a single file system over such a cloud and a single programming model while allowing elasticity, availability, and scalability.

Future Work: We will study an adaptable version of the Paxos algorithm that does not overload the network.

REFERENCES

1. http://en.wikipedia.org/wiki/Grid_computing
2. http://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&ved=0CDwQFjAB&url=http%3A%2F%2Fjalapeno.therning.org%2Freport.pdf&ei=RSvgTqzBMYrZgAfe-_z0BQ&usg=AFQjCNFZ5F-spv4fccd4n84Z8uDgC8KMAg
3. http://en.wikipedia.org/wiki/Tuple_space
4. http://en.wikipedia.org/wiki/Dataflow_programming
5. A Unified Operating System for Clouds and Manycore: fos
6. http://en.wikipedia.org/wiki/VMware
7. http://www.xtreemos.eu/
8. http://en.wikipedia.org/wiki/Distributed_hash_table
9. http://code.google.com/p/scalaris/
10. http://en.wikipedia.org/wiki/Virtualization

VITA

Anudeep Meka was born in Vizianagaram, India in September 1988. He earned his primary and secondary education from St. Joseph's English medium school in Vizianagaram, Andhra Pradesh. After finishing his high school, he took a very competitive entrance examination for engineering known as EAMCET and stood in top 0.5%. After qualifying in this examination he got admission to Department of Electrical Engineering, M.V.G.R College of engineering, one of the prestigious institutes in Andhra Pradesh, India. He received his Bachelor of Engineering (B.E.) from J.N.T University, Hyderabad, India, in spring 2010.

Then he came to United States of America to pursue a master's degree. He then joined the graduate program at Louisiana State University, Baton Rouge, in fall 2010. He is a candidate for the degree of Master of Science in System Science to be awarded at the commencement of spring

2012.