An Enhancement to CloudSim via Distributed Data Storage

Roopa T P
Dept. Of Computer Science
PESIT South Campus
Bangalore, India

Surbhi Agrawal
Dept. Of Computer Science
PESIT South Campus
Bangalore, India

Snehanshu Saha
Dept. Of Computer Science
PESIT South Campus
Bangalore, India

***Abstract*—The advent of cloud computing technology has made its presence effectively felt in various application areas such as business, industry, scientific, administrative, astronomy, high-energy physics, information, education etc. Its capability of meeting the various continuously changing demands in all fields makes it increasingly popular. In order to use cloud computing technology, the user has to make him/her familiar with the various facets of the technology through the simulators, which serve as training assets to familiarize the users with the real time scenario. Among the various cloud simulators available, the widely used CloudSim 3.0.3 is considered for analysis. This version of the simulator proposed in the paper uses hard drive storage, with new features in data storage being included to achieve distributed storage resulting in load balancing across the nodes. The method helps achieve reduction in data migration and mitigate data loss in case nodes crash.**

Keywords— CloudSim; File stripping; Distributed storage; File replication; Data migration; Quality of Service (QoS);

I. INTRODUCTION

Cloud computing, which is embellished with a potential for flexible distributed data management was first proposed in 2006. It is widely used in all areas where bulk data in sizes of TB or PB are processed. Cloud service providers namely Amazon's S3, Google's GFS and Microsoft's Live Services are used to store and manage the data.

The CloudSim assumed a prominent role in depicting the real time situation which familiarizes the users with the various facets of cloud computing and also enables the user to assess the performance. The users could fathom the real time scenario optimally. Simulators help the researchers and developers test the newly developed methods to evaluate the functioning prior to real time deployment. CloudSim is the most popular simulator tool available for use in a cloud computing environment. It is an event driven simulator, built upon the core engine of a grid simulator, GridSim. Java, the most powerful object oriented programming language is used in CloudSim. Because of Object Oriented feature, CloudSim modules are easily extendable according to the user's requirements. Additionally, CloudSim has features that include modeling and creating a huge data center endowed with an unlimited number of virtual machines. It introduces brokering policies and supports the crucial feature of pay-as-you-go model. One of its unique features is the federated policy, which is rarely available to any other simulator. Because of the elastic nature of CloudSim, it has gained popularity over time. Currently, in HP labs (Palo Alto) and Duke University (U.S.A.) researchers are using CloudSim for evaluation of resource-allocation algorithms and energy-efficient management of data centers [1]. As a completely customizable tool, it allows extension and definition of policies in all the components of the software stack, making it an extremely suitable research tool. The tool proficiently handles the complexities arising from simulated environments. Performance related to provisioning and service delivery policies in a repeatable and controlled environment can be tested free of cost using CloudSim. CloudSim also helps to overcome the bottlenecks before deployment in real time commercial clouds [2].

The existing CloudSim simulator has typical hard drive storage. Entire data are stored in a particular node. In case the node crashes, the consequences are severe. The storage and retrieval time in this case is significantly high. These disadvantages are addressed via the solution proposed in our work.

To make CloudSim more efficient in terms of data storage, prevention of data loss due to crash of the nodes, a distributed storage paradigm has been implemented in the analysis. As a result, the improved CloudSim has the following new features:

- It splits the input file into a number of chunks and stores different chunks at different nodes, to enable retrieval of stored data even under crash conditions.
- Replicas of each of the chunks are created to facilitate retrieval even in the absence of original chunks.
- The master chunk and the replicated chunk are stored on different nodes which ensure almost complete recovery of the stored files even when certain nodes crash.

It is to be noted that the key change suggested relates to adapting a distributed storage concept.

The remainder of the paper is organized as follows. Section II discusses about the related papers. Section III consists of algorithms for the proposed work and the figurative description of the implementation. Section IV describes the results of our work followed by the conclusion and comments about future enhancements.

II. RELATED WORK

Significant efforts are on to enhance the CloudSim. We mention a few cited contributions which serve as a baseline for our work.

A. *A modeling and simulation toolkit for cloud data storage: An extension to CloudSim*

Cloud computing has been gaining a lot of popularity over the last few years with cloud storage technology being one of the most important hallmarks. Massive data are generated everyday making the job of managing increasingly complex. File striping and the data replica management functions are introduced in [3].

B. *A Load Balancing Algorithm For Private Cloud Storage*

Cloud computing demands a lot of a shared pool of configurable resources e.g. storage, servers and applications. These resources could be from a Private or a Public cloud. Private clouds are restricted to a particular organization. Utilization of such private cloud storage can often lead to an increase in storage demand. As this process takes place, the data is placed or moved around within the nodes. To accomplish this, one has to maintain the load across several nodes and thus the concept of load balancing and data migration came into existence [4].

C. *A Partial Replication Load Balancing Algorithm for Distributed Data as a Service*

An easy and direct algorithm to solve the issue of load balancing in the process of implementing Data as a Service (DaaS) in the cloud is discussed in this paper. The proposed algorithm is based on an earlier approach to efficient bi-directional data downloading. The main contribution was to put forward a solution to the problem of high storage demand while storing replicated data in multiple cloud nodes. This approach introduces a model to store partial replicas of the data on multiple distributed cloud servers. A direct method to download this data from multiple cloud servers by coordinating the download process among the different nodes and different parts of the replicas on the cloud is detailed [5].

D. *Enhanced Distributed Storage on the cloud*

A protocol was designed splitting a file into multiple sub files and store them in different virtual machines belonging to either different clouds or different clusters belonging to the same cloud or the same cluster of the same cloud. Such distribution is expected to enhance security and space utilization. The model helps build reliability as the data is not on a single cloud or on the same cluster [6].

E. *Optimize Block-Level Cloud Storage with Load Balancing Strategy*

This paper presents a dynamic load balancing strategy between multiple volumes of servers in the "Orthrus" system. The optimal iCSI connections deployment method with a genetic algorithm based on machine performances is proposed in this paper. Dynamic load balancing strategy helps "ORTHRUS" achieve higher I/O performance [7].

F. *MR-CloudSim: Designing and implementing MapReduce Computing Model on CloudSim*

In this paper MapReduce model has been implemented on CloudSim. Since CloudSim does not provide much perspective in the context of file size and the contents, implementing MapReduce requires modifying and/or improving existing functions of CloudSim. The MapReduce technique implemented has not yet been tested in real time [8].

III. PROPOSED WORK

The current library of CloudSim 3.0.3 consists of two classes supporting the storage functions. These classes inherit the Storage Class, which is the super class. The first class is the Harddrive-Stroage Class, which simulates a typical hard drive storage. The second class is the SanStorage class representing a storage area network composed of a set of hard disks connected in a LAN, yet to be completely functional. The algorithms for the private cloud storage described by [4] are extended to the simulation platform by eliminating the concept of data migration. Distributed storage for the cloud simulation platform CloudSim is being proposed here. It consists of two algorithms-

File Splitting Algorithm –This splits a file into a number of chunks based on the size of the input file and the number of nodes available for storage. It also creates the replicas of the chunks.

Split File Placement Algorithm- This algorithm ensures the placement of the master chunk and its replica in separate nodes.

A. ***File Splitting Algorithm***

The proposed algorithm treats a file to be stored as an input from the user. Then it performs the splitting of the file based on the file size and the number of nodes on which the file chunks have to be placed. The replicas of each of the chunks are created. In the simulation environment, the user has to provide the minimum and maximum capacity for each node. This prompts the simulator to define the capacities of the available nodes between the minimum and maximum limits specified by the user.

B. Split File Placement Algorithm

The algorithm places the master chunks and the replica chunks across various nodes. First, the process of placing the master chunks sequentially is carried out. Load capacities of each node are carefully considered before the placement of file chunks. Depending on the capacities of the nodes, the master chunks are placed. Next, it performs the placement of replica chunks in the same way while ensuring that any replica chunk is not placed on the same node, where its corresponding master chunk is placed already. This guarantees a distributed storage. Hence, a node which is heavily loaded will not be considered for placement of the chunk further and, thereby will be directly placed onto the lightly loaded node. Therefore, data migration has been reduced to a great extent, compared to the work done earlier [4].

The two algorithms are enunciated below.

Algorithm1. File Splitting

```
Input: Double MinstorageCapacity_i, MaxstorageCapacity_i
//Minimum & maximum storage capacities for i-th    node in the
cluster.
Output: Chunks of the input file and its corresponding replicas
Capacity_i=random (MinstorageCapacity_i,MaxstorageCapacity_i)
//Capacity_i is randomly generated within the given capacity
range for i-th node in the cluster.
SplitSize=Filesize/no_of_nodes. //determine the split size for
splitting the input file into number of chunks and hence based on
this size, file chunks are created. Create a copy for each chunk.

Call Split File Placement Algorithm.
```

Algorithm2. Split File Placement

```
 Input: int n //no. of nodes in the cluster
 Int  a, a1//location of master file & replica file  respectively.
 totalCapacity <- 0 //total capacity of cluster
 float delta //system parameter
 float cluster_load //workload for cluster
 Int TotalNumberOfChunks//both master and replica chunks
Output: Master chunk and its corresponding replica chunk are
placed on different storage nodes

// compute total capacity of cluster
for i <- 0 to n do
totalCapacity+= Capacity_i
end for
//provide different locations to a and a1
while (TotalNumberOfChunks>0)

//place master chunk of the file
for i <-a to n do
ideal_load_i  = ((cluster_load/totalCapacity)*Capacity_i)
heavyloadnode_i<- (1+delta)* ideal_load_i
ResidualCapacity_i<- Capacity_i - UtilisedCapacity_i
if(ResidualCapacity_i> SplitSize) then
if(UtilisedCapacity_i < heavyloadnode_i) then
          place master chunk of the file at nodei
end if
end if
end for
//place replica of master chunk
for i <-a1 to n do
ideal_load_i  = ((cluster_load/totalCapacity)*Capacity_i)
heavyloadnode_i<- (1+delta)* ideal_load_i
ResidualCapacity_i<- Capacity_i - UtilisedCapacity_i
if(ResidualCapacity_i> SplitSize) then
if(UtilisedCapacity_i< heavyloadnode_i) then
          place replica chunk of the file at nodei
end if
end if
end for

a++ // Increment to place the master copy in the next node
a1++ // Increment to place the replicated copy in the next node

if(a==n)
        a=random(0,n)
end if

if(a1==n)
        a1= random(0,n)
end if

if(a==a1)
         a1=random(0,n)
end if
//ensure a and a1 do not point to the same node location
end while
```

By following the above algorithm, it can be shown that the master copy and the replica copy of the same chunk are never stored in the same node. The algorithm reduces data migration. It ensures that all the nodes are equally utilized and are normally loaded.

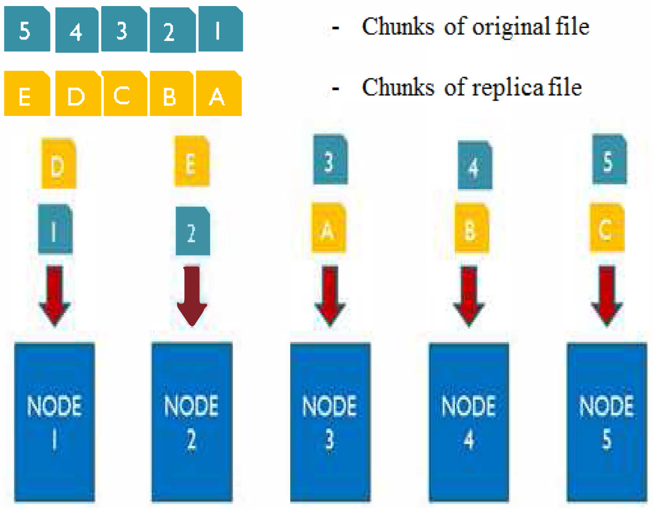


Fig. 1. Distributed Data Storage with equal node capacities

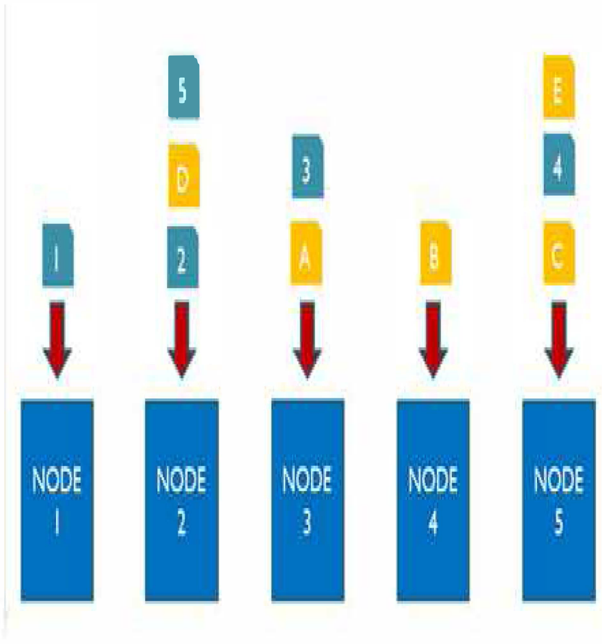


Fig.2. Distributed Data Storage with uneven node capacities

It can be observed from the Figure1 that the file is split into five chunks numbered 1,2,3,4 and 5. The corresponding replicas A, B, C, D and E are formed. The nodes have equal storage capacity and hence equal numbers of chunks/replicas are stored in the nodes. This helps in understanding the concept of file splitting and replica formation. Figure 2 depicts the real time scenario in the implemented algorithm where the nodes have different capacities and the storage pattern of chunks/replicas is varied.

IV. SIMULATION & RESULTS

The concept outlined above was deployed over the existing CloudSim. Additional methods were created for storing, deleting and retrieving the files ensuring distributed storage. The simulation time was compared against typical hard drive storage in the current version of CloudSim. The following graphs demonstrate the results. The load in the following graphs is assumed to be stored across 15 nodes in a cluster simultaneously, against hard drive storage in the same node. It can be seen from the graph, shown in Fig3 that the conventional hard drive storage has proportionate increase in the time required for storage whereas the storage time in the distributed data storage system is remarkably less.

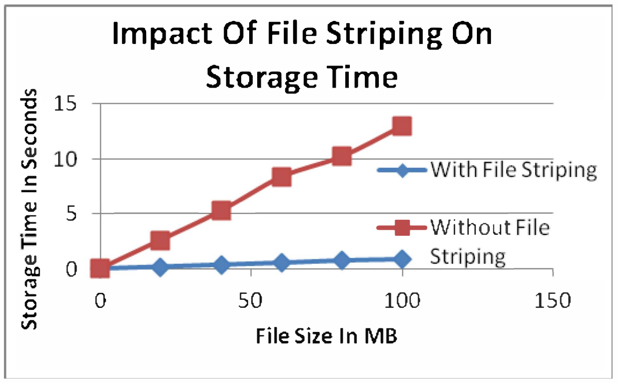


Fig. 3. Impact of File Stripping on Storage Time

Similar observations can be made from graph of Fig4, which displays the retrieval time in the context of conventional and proposed data storage. The retrieval time in the proposed data storage system is far lesser when compared to hard drive storage. This is achieved by input file striping and creation of replicas.

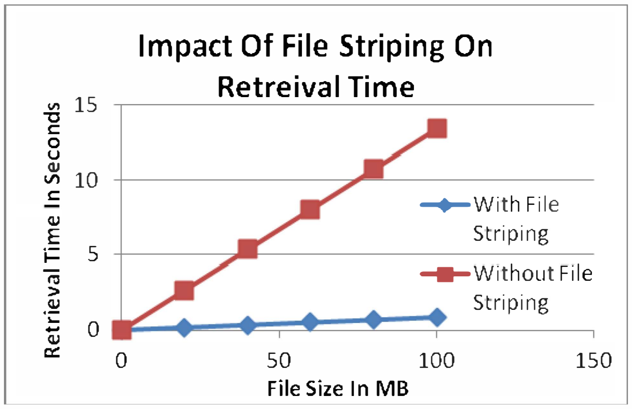


Fig.4. Impact of File stripping on retrieval time

Figure 5 exhibits the decrease in time complexity, made possible by elimination of data migration in the proposed concept of distributed data storage. The time complexity was calculated using the Step Count method. A variable 'count' was introduced in the programs which included data migration and the one which eliminated the concept of data migration. A program step is a syntactically or semantically meaningful program segment whose execution time is independent of the instance characteristics [9]. The number of steps contributed by each statement was calculated using the following formula

$$\text{Step per execution} * \text{frequency} \qquad (1)$$

The contributions of all the statements were added and the time complexities were calculated. We discover that our algorithm has brought down the time complexity from O $(4n^2)$ to O (n^2).

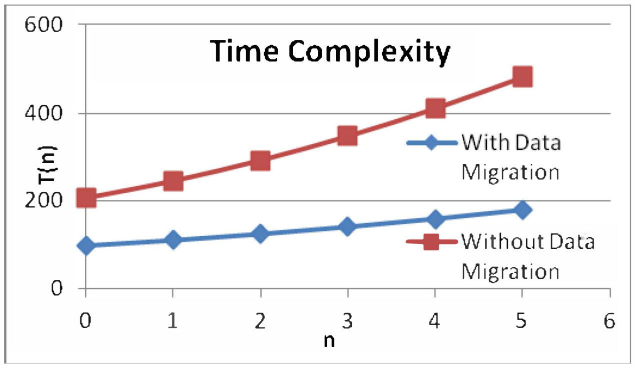


Fig. 5. Time Complexity based on Data migration

Throughput refers to the performance of tasks by a computing service or a device over a specific period. It measures the amount of completed work against time consumed and may be used to measure the performance. Figure 6 shows the total system time consumed which includes storage and processing across 15 nodes. It also measures the entire system time consumed in identifying the location of master chunks and the replica chunks and retrieving them simultaneously from the nodes.

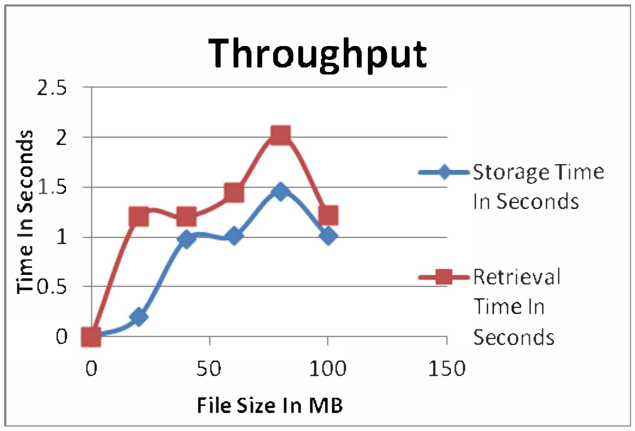
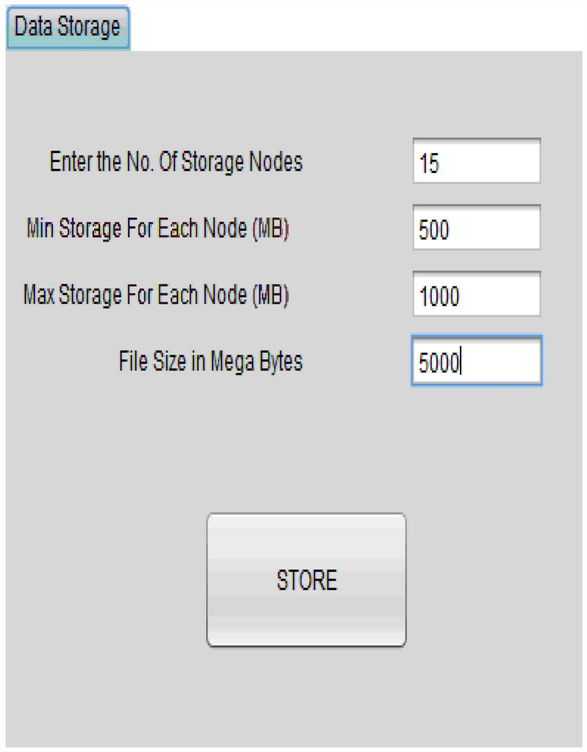


Fig. 6. Throughput

Current version of CloudSim does not have a user interface. Our work incorporates a user interface. This enables the user to input only the required data without the necessity of writing any program as required in the present CloudSim. Figure 7 shows one of the user interfaces created for storing the data. The interface was created using Java Swings. Swing is a set of classes that provides powerful and flexible GUI components compared to the Abstract Window Type (AWT). The Swing does not replace it, but swing is built on the foundation of AWT. The key features of swings include: a) Swing Components are lightweight, which means that they are written completely in Java and do not map directly to platform specific peers b) Swing supports a pluggable look and feel, implying that it is possible to separate the look and feel of a component from the logic of the component [10]. The number of nodes across which the load needs to be stored is given as an input by the user. A random storage capacity for each node is generated in the range of the minimum and maximum storage capacity in MB. The size of the input file is entered by the user. Once the values are entered, the STORE button is clicked and the load distributed across various nodes is obtained in the form of a graph.



Enter the No. Of Storage Nodes	15
Min Storage For Each Node (MB)	500
Max Storage For Each Node (MB)	1000
File Size in Mega Bytes	5000

Fig.7. Data Storage Interface

Figure 8 shows the distribution of load of an input file size 5000MB in 15 different nodes which remains approximately the same over most of the nodes. Each of the nodes is assigned a random storage capacity within the minimum and maximum storage capacity given by the user in the interface. The file chunks are placed from the first node onwards. Each time the chunk is placed in a particular node, the residual capacity of the node decreases. This type of assignment continues and once the last node is reached, the control comes back to the first node and the cycle of loading the nodes continues. The example considers nodes 0 and 1 to possess less residual capacity rendering them incapable of accommodating subsequent file chunks. They are assigned to the nodes 2 and 3 which have higher storage capacity in them. This process has been shown as shooting up the load in the graph.

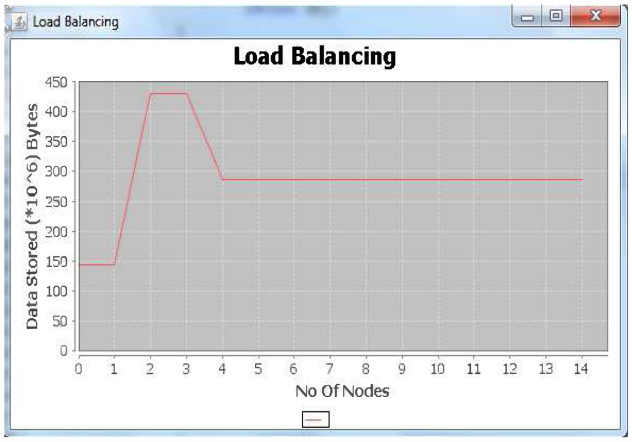


Fig. 8. Load Balanced across 15 nodes for a 5000MB File during the first run

To depict the real time conditions of the nodes, different random storage capacities are assigned to the nodes. When the same values shown in the interface are assigned for data storage the second time, the pattern of the load balancing gets modified as shown in Fig9 due to higher capacities assigned to nodes 5 and 6.

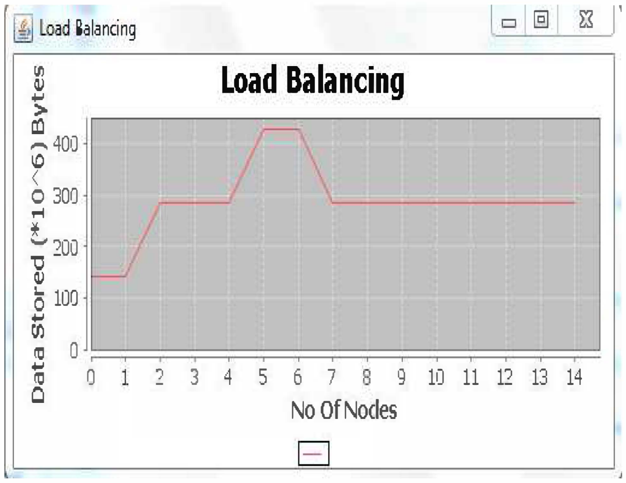


Fig. 9. Load Balanced across 15 nodes for a 5000MB File during second run

The advantage of this load balancing strategy over the different nodes can easily be grasped by observing the load graph in a hard drive storage system shown in Fig: 10, where the entire data is stored in one node which will be lost in case of the failure of that node.

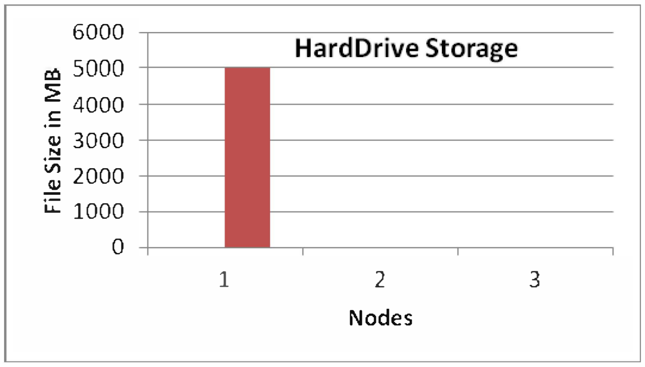


Fig. 10. Load Pattern in a Hard drive Storage

V. CONCLUSION

The paper proposes a concept of improving the performance of the CloudSim in terms of faster data storage and reliable retrieval by resorting to distributed data storage. There is demonstrable evidence that the method is able to impart reduced storage and retrieval time and also eliminates the possibility of data loss in the event of "node-crash".

VI. FUTURE WORK

Improvements can be accomplished by extending this concept to the storage of multiple files. A cache can be provided to store the most recently accessed files, which will help in reducing retrieval time further. Files can also be encrypted which may augment data security.

REFERENCES

[1] Arif Ahmed and Abadhan Saumya Sabyasachi. "Cloud Computing Simulators: A Detailed Survey and Future Direction". IEEE International Advance Computing Conference, 2014.pp: 866-872

[2] Rajkumar Buyya, Rajiv Ranjan and Rodrigo N. Calheiros. "Modeling and Simulation of Scalable Cloud Computing Environments and the CloudSim Toolkit: Challenges and Opportunities", High Performance Computing & Simulation, International Conference on HPCS'09, pp. 1-11

[3] Saiqin Long, Yuelong Zhao. "A toolkit for modeling and simulating cloud data storage: an extension to CloudSim". International Conference on Control Engineering and Communication Technology, 2012, pp. 597-600.

[4] Prabavathy.B, Priya.K, Chitra Babu. "A Load Balancing Algorithm For Private Cloud Storage". 4th ICCNT, 2013.

[5] Klaithem Al Nuaimi, Nader Mohamed, Mariam Al Nuaimi, Jameela Al-Jaroodi "A Partial Replication Load Balancing Algoirthm for Distributed Data as a Service". IEEE, 2013, pp. 35-40

[6] Shushant Srivastava, Vikas Gupta, Rajesh Yadav, Krishna Kant "Enhanced Distributed Storage on the cloud". Third International Conference on Computer and Communication Technology, 2012, pp.321-325

[7] Li-Zhou, Yi-Cheng Wang, Ji-Lin Zhang Wan, Yong-Jian Ren “Optimize Block-Level Cloud Storage With Load Balance Strategy”. IEEE 26th International Parallel and Distributed Processing Symposium Workshops &PhD Forum, 2012, pp: 2162-2167

[8] Jongtack Jung and Hwangnam Kim “MR-CloudSim: Designing and implementing MapReduce Computing Model on CloudSim”, IEEE, 2012, pp: 504-509.

[9] A.M.Padma Reddy “Basic Concepts” in “Data Structure Using C”, 12th edition, Bangalore, India: Sri Nandi Publications, 2011, ch.1, sec.5, pp.53

[10] Herbert Schildt “Introducing Swings” in“The Complete Reference Java”, 7th edition, TATA McGRAW-HILL EDITION, 2012, pp.859-860.

[11] CloudSim, http://www.cloudbus.org/cloudsim/

[12] NetBeans IDE, https://netbeans.org/