International Journal of Emerging Engineering Research and Technology
Volume 4, Issue 6, June 2016, PP 1-10
ISSN 2349-4395 (Print) & ISSN 2349-4409 (Online)

Implementation of Load Balancing Algorithm for Data Intensive Applications using MPI

V Vijayasherly[1], L Shalini[1], Chandresh Sharma[2], Manasvi Pandya[2]

[1]SCOPE, VIT Vellore, [2]B.Tech, CSE

ABSTRACT

Applications which are data intensive and parallel, say data compression and decompression, projectile tracking and other scientific applications need to deploy large data sets and require large computational power. Due to high cost of super computers, they cannot be utilized extensively in such complex scenarios. This led to deploying distributed systems to resolve these problems. High performance distributed computing clusters are designed to establish a system with single image and high throughput. Identifying the busy nodes in a huge cluster makes the cluster more effective through Load Balancing Algorithm(LBA) and gives the heavily loaded nodes minimal time to come up for the better execution in the future runs. MPI (Message Passing Interface) is a standard API (Application Programming Interface) used to create distributed applications.

This paper demonstrates load balancing on a computation cluster by implementing data compression/ decompression so as to achieve higher computational throughput. The load balancing technique implemented can further be used for developing distributed applications for image processing, network routing etc.

Keywords: Load Balancing, Data intensive applications

INTRODUCTION

General

In the present era the popularity of distributed computing is increasing because the computing problems to be solved have become more complicated and larger in size and due to high cost of super computers, they cannot be utilized extensively to solve large scale problems.

The word distributed in terms such as "distributed system", "distributed programming" and "distributed algorithm" originally referred to computer networks [5] where individual computers were physically distributed within some geographical area. The terms are nowadays used in a much wider sense, even referring to autonomous processes that run on the same physical computer and interact with each other by message passing.

A distributed system may have a common goal, such as solving a large computational problem. Alternatively, each computer may have its own user with individual needs and the purpose of the distributed system is to coordinate the use of shared resources or provide communication services [5] to the users.

There are two main reasons for using distributed systems and distributed computing. First, the very nature of the application may require the use of a communication network that connects several computers. For example, data is produced in one physical location and it is needed in another location. Second, there are many cases in which the use of a single computer would be possible in principle, but the use of a distributed system is beneficial for practical reasons. For example, it may be more cost-efficient to obtain the desired level of performance by using a cluster of several low-end computers, in comparison with a single high-end computer. Moreover, a distributed system may be easier to expand and manage than a monolithic uniprocessor system.

One of the challenges of distributed application development is load balancing between the different computing nodes in the distributed computing environment to achieve higher throughput. The paper implements a load balancing algorithm using Message Passing Interface (MPI). This Algorithm would be suitable for applications which are data intensive.

Address for correspondence:

vvijayasherly@vit.ac.in

Motivation

Although intensive work has been done in the area of load balancing [2], the cluster computing environment is different from the traditional parallel systems, which prevents existing load balancing schemes from benefiting large scale parallel applications. A survey of the existing solutions and new efforts in load balancing [2] to address the new challenges in cluster computing is discussed; the survey approaches were classified into three categories: resource aware repartition, divisible load theory and prediction based schemes.

Existing cluster schedulers dedicated for cluster resource management [9] are not suitable for data intensive applications because they operate on a large scale and are optimized for long jobs with moderate size of input and output data. Additionally, network utilization is not optimized by cluster schedulers and availability of a job input data is not optimized to match the availability of the computation resources.

Load balancing algorithm for a distributed cluster implemented in the paper uses load information such as CPU context switches, CPU utilisation, memory utilisation and network utilisation on the node. The performance of the algorithm can be evaluated by comparing it with the algorithm which uses no Load Balancing Scheme.

The effective use of a heterogeneous multicomputer system depends on the scheduling policy used. Often the scheduling algorithms used are those that were intended for homogeneous systems. On the other hand, most scheduling algorithms designed with heterogeneity in mind were developed under unrealistic assumptions. To enhance the performance of a multicomputer system, load sharing can be used during scheduling [6] to distribute the workload evenly among the constituent nodes.

Parallel processing is the process of breaking down the larger problem into small components, tasks or calculations that can be computed in parallel.

The advantages of the distributed systems [4] to that of centralized systems are high performance, high throughput, high availability, economies of scale, expandability and scalability.

Problem Description

The paper "Implementation of Load Balancing Algorithm for Data Intensive Application using MPI" aims at

i Implementing the load balancing algorithm for a compute-cluster using Message Passing Interface (MPI).

ii Implementing the data compression algorithm on the multi computational node environment.

The application chosen is data compression because it is a data intensive application and hence would be most suitable for the implementation of the load balancing algorithm. The load balancing algorithm using CPU context switches, CPU utilisation, memory usage and network traffic on node, performs better than the traditional algorithm which uses only CPU Queue length in the Clusters.

Related Work

A Computer Cluster [6] is a group of linked computers, working together closely so that in many respects they form a single computer. The components of a cluster are commonly, but not always, connected to each other through fast LAN's. Clusters are usually deployed to improve performance and/or availability over that of a single computer, while typically being much more cost-effective than single computers of comparable speed or availability.

Load balancing is a computer networking methodology to distribute workload across multiple computers or a computer cluster, network links, central processing units, disk drives or other resources to achieve optimal resource utilization, maximize throughput, minimize response time and avoid overload. The various schemes [6] currently being used have been discussed below.

OVERVIEW OF THE PROPOSED SYSTEM

Introduction of the Problem and its Related Concepts

One of the major challenges of distributed computing is Load Balancing between the different computing nodes in the compute cluster. Load balancing can be defined as distribution of the

computational work between the different compute nodes in the cluster. Proper load balancing when achieved, results in a more efficient performance of the compute cluster being employed.

The paper "Implementation of Load Balancing Algorithm for Data Intensive Applications using MPI" aims at meeting this challenge of load balancing when an application is executed on a compute cluster. It uses data compression as a data intensive application to be executed on the cluster and performs load balancing for its execution using MPI (Message Passing Interface). This would lead to a higher throughput in the system as compared to the case when load balancing is not employed.

i In the Round Robin Load Balancing Scheme, a slow node can significantly reduce the performance of an overall parallel process and can also result into synchronization problems on the cluster.

ii In the Weight based Load Balancing Scheme, the relative weights to be assigned to each node needs to be carefully determined by the user. Also it does not make use of real time system statistics which may differ at subsequent application executions.

To change the specified weight of a node, system has to be rebooted.

iii In the Random Load Balancing Schemes, there is processing overhead for generating a random number for each request and there is possibility that the load may not be evenly balanced over a small number of requests. It does not make use of real time system statistics which may differ at subsequent application executions.

iv Gzip makes use of single computer's resources to perform task and thus it is not inherently parallel so it cannot be used effectively on clustered system.

v Gzip doesn't provide any graphical user interface.

Proposed Solutions

i The Load Balancing scheme should take into consideration the real time node statistics in order to obtain maximum performance from the cluster.

ii The complete Load balancing scheme should be automated so as to avoid chances of users making wrong assumptions by the user as in the case of Weighted Load Balancing schemes.

iii The Load Balancing scheme should be predictable and aim at early completion of task instead of using random nodes which may decrease the performance as in the case of Random Load Balancing scheme.

iv The Data compression process must be parallelized in order to make use of the clustered system.

v The user interfaces must be provided for ease of usage.

ANALYSIS AND DESIGN

Brief Introduction

Analysis has been performed to identify the system requirements and to decide upon the application functionalities. The description of the design of the system shows the organisation of the different sub systems and the interaction between them so as to achieve desired outcome. The LBA has been developed to perform the data compression on the compute cluster to demonstrate its utility for data intensive applications.

Product Features

The work presented in the paper has the following features

i It implements the load balancing algorithm for a compute-cluster using Message Passing Interface (MPI).

ii It performs the data compression of plain text files on the compute cluster.

iii It performs the decompression of data, which has been compressed on the compute cluster.

iv It stores the results of operations in a shared directory to provide a Single System Image.

v It provides java based user interfaces for operations which results in ease of usage.

vi It provides the complete application runtime log to the user which can be used to trace any faults.

Domain Requirements

Building the Compute Cluster

The compute-cluster is created by the use of MPICH2 binaries. The MPD (Message Passing Daemon) is used to add multiple compute nodes to the cluster by the administrator before first application is run. The process need not be repeated for subsequent executions by the end user.

Parallelization of Task

The application to be executed on the compute-cluster is divided into primitive applications through parallelization. This is performed by using pthreads on the master. The input file is divided into blocks of equal size and subsequently processed by the slave nodes upon request by the master.

Single System Image

In distributed computing, a single system image (SSI) cluster is a cluster of machines that appears to be one single system. Most SSI systems provide a single view of the file system. This may be achieved by a simple NFS server, shared disk devices or even file replication.

The advantage of a single root view is that processes may be run on any available node and access needed files with no special precautions. If the cluster implements process migration, a single root view enables direct access to the files from the node where the process is currently running.

Execution of Load Balancing Algorithm

The Load Balancing Algorithm is executed on the server node for the identification of the best suitable compute node. The node statistics are stored in a linked list after evaluation from LBA.

Statistics Collection

The statistics of various computing nodes are collected using Linux utilities, which are

i MPSTAT- for CPU usage.

ii FREE- for free RAM.

iii SAR – for network statistics.

iv VMSTAT-for number of context switches.

Data Compression and Decompression

The application makes use of GZIP to process data on individual nodes. Further the compressed blocks are archived into a single file of extension .pgz using TAR utility.

The formats supported for compression operation are txt, text, c, pl, sh, java, jsp, and c++.

The compressed output file is stored with .pgz extension.

Design of the Proposed System

System Architecture Design

The architecture of the design needs to be selected while taking under consideration the functional and the non functional requirements of the system. The selection of a system architecture design also affects the important aspects of the system such as performance, maintainability and the durability of the system to be designed.

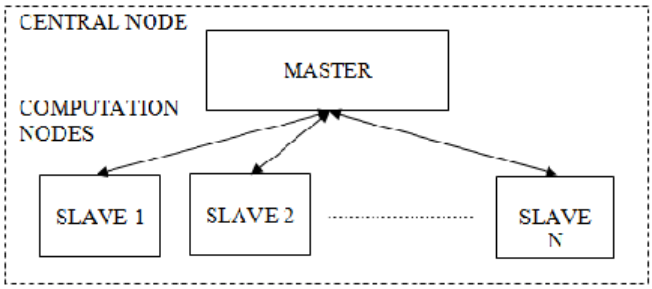

Fig1. *System Architecture*

The system has been organised into master-slave architecture as shown in Fig. 1 where the central node which executes the load balancing algorithm acts as the master while the nodes on which data compression is carried out acts as the slave nodes.

All the slave nodes are connected to master through bi directional communication link.

Application Architecture

The application has been organised into four major modules namely Data Manager, LBA, Statistics Manager, Computing Component as shown in Fig 2. First three modules run on a single central node while the fourth module run on the slave nodes. The different modules communicate using MPI mechanism.

LBA Architecture

LBA Architecture is designed to provide an easily extendable basis for parallel computation. It consists of a single element which enables parallel computation.

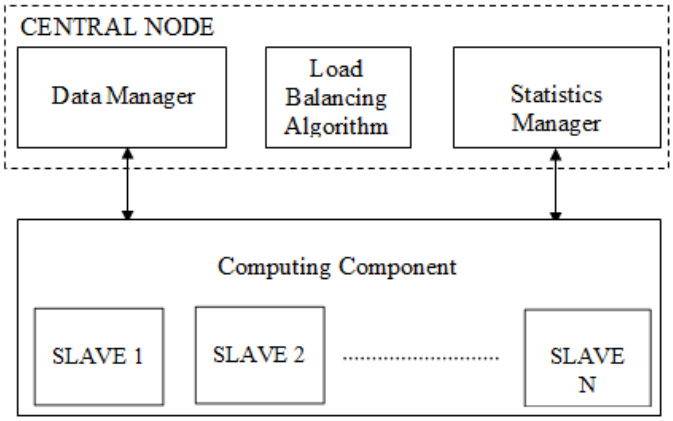

Fig2. *Detailed Application Architecture*

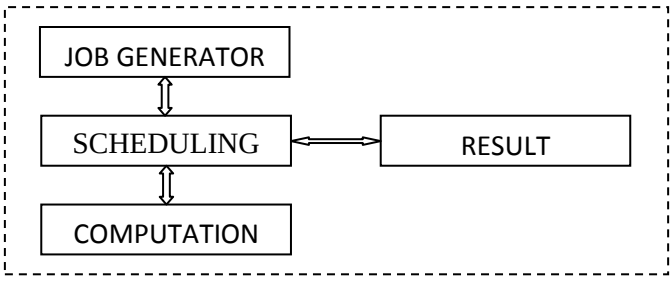

Fig3. *LBA Architecture*

The LBA architecture has been shown in the Fig. 3 for a single cluster, namely LBA element, contains scheduling component and computation components [1]. Scheduling component is used to schedule jobs and dispatch data to computing components. Computing components are designed to perform application logic to receive input data and to return computation results to the scheduling component. Various statistical data are returned together with the results to the scheduling component in order to support scheduler decision making process. Scheduling component and each of the compute nodes are executed at the separate MPI nodes. In order to make LBA architecture generic and easily extendable it has been organised into the following components.

LBA Scheduling Algorithm

Signalling between scheduling component and computing component [1] is done using MPI. This includes information about status of the compute nodes, together with the information on the jobs to be done. The transfer of input and output job data is accomplished using operating system provided resources and is configurable.

Scheduling Component Algorithm

Scheduling component is designed to run while there are data blocks available. After the

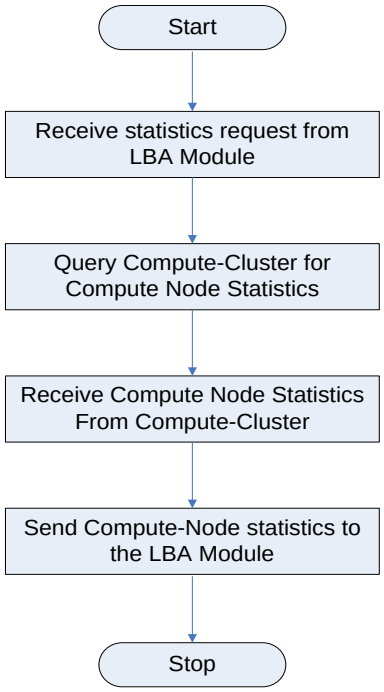

Fig4. *Statistics Manager Module*

request is acquired from the data manager, a target compute node (targetNode) must be determined. Before the decision, the scheduling component updates its information with the statistical data that are queued up from the compute nodes until nodes Exists() is true. After this, the best compute component for the job is picked up (insert Node). "Info" object contains all the information about the status of the compute components.

Finding the best compute node for the job is completed in two phases. In the first phase computing components not suitable for the job are filtered out. The criteria for elimination of compute nodes include inadequate storage size for input data. In the second phase the best node is picked by sorting the nodes based on cpu idle, context switches, free RAM, network usage in decreasing order of priority. In the case of all compute components filtered out in the first phase, the algorithm continues to gather statistics until at least one compute component becomes eligible for the job.

When the job is matched with a compute component, data staging is initiated (initiateDataStagingIN) for that job. Data staging is a process of preloading job input data to the node that is scheduled to execute the job. Initiation denotes that data should be moved, but transfer of data is performed only when number of concurrent transfers in progress is below given threshold and when the data for the earlier starting jobs are already transferred. All of the data staging is performed asynchronously so the scheduling component can continue its execution (implemented using pthreads). Output data for completed jobs are also retrieved in the background when similar conditions are met (initiateDataStagingOUT). These are then available for the developer provided result handler. In the end of its cycle the scheduler component checks if there is any finished data staging (existsFinishedStaging). For every finished data staging the appropriate computing component is notified (Notify) and the job execution can take place. When all the jobs are scheduled, the scheduler component waits for all jobs to complete.

Computing Component Algorithm

Computing component is designed to check for availability of new jobs and to queue jobs for the computing utility. Multiple computing utilities are allowed to be attached to the computing component and to consume jobs from the same queue. The computing utility updates internal statistics, forwards the updated information to the scheduling component, gets jobs from the queue and executes these jobs. Communication calls implemented in the computing component and computing utility are exclusively locked and forced to be performed only one at a time. Locking in the computing component is performed in the waitForJob procedure (implemented using pthread_mutex) that is designed not to block on MPI calls. This is necessary due to threading issues in MPI implementations, but also adds up to the processing requirements for the algorithm.

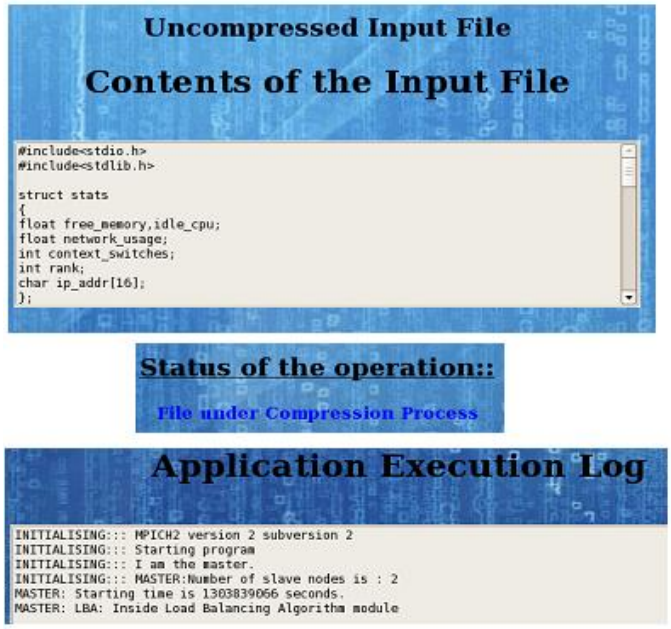

Fig8. *Compression Operation*

The snapshot shown in the Fig 8 shows the page generated by performOperations2.java servlet of the user interface after performing compression.

Fig9. *Decompression Operation*

The snapshot shown in the Fig 9 shows the page generated by performOperations2.java servlet of the user interface after performing decompression.

RESULTS AND DISCUSSION

Results

The application is executed on a compute cluster composed of desktop systems connected with a LAN and running on a LINUX platform. The system runs upon the principle of Single System Image, thus making the cluster environment transparent to the user. The application system is meant for demonstrating load balancing for any data intensive application which can be parallelised and executed on a cluster of compute-nodes. The data intensive application which has been selected is data compression/decompression.

Firstly, the application is capable of accepting an input file to be processed from the user and to notify the user, if the input file submitted is of an unsupported file format.

Secondly, the application checks whether an MPD is set up between the different compute nodes in the compute cluster or not. If the MPD is not already established then the user is notified about the absence of MPD and hence the absence of a proper environment for the execution of the application.

Further when a valid input file is selected, the application system running on the compute cluster divides the input file into multiple sub blocks, so as to achieve parallelisation in the execution of the required operation. The load balancing algorithm executed on running the application checks the statistics of the various compute nodes present in the compute cluster. On the basis of those statistics, a sorted linked list of compute nodes is generated and hence the best suitable compute node for the execution of the task is selected.

Finally the output file with respect to the submitted input file is generated and is stored in an appropriate location in the file system of the compute cluster. Hence the desired result is generated for the user.

Performance Analysis

i The load balancing algorithm is performing in the desired manner hence generating a linked list of various compute nodes and is selecting best suitable compute node.

ii The application system is accepting an input file of supported file format for performing compression/decompression operation.

iii The application checks and successfully notifies the user whether an appropriate environment is present for its execution i.e., if a MPD is running on the compute-cluster or not.

iv The input file when submitted to the application is being divided into sub data blocks to achieve parallelization in execution.

v The application system is generating appropriate output file in shared directory with respect to the input file submitted to the application, providing user with transparent execution.

vi The performance of the cluster has been evaluated for two file sizes for data compression/decompression and following graphs of processing time/no. of slave nodes have been obtained.

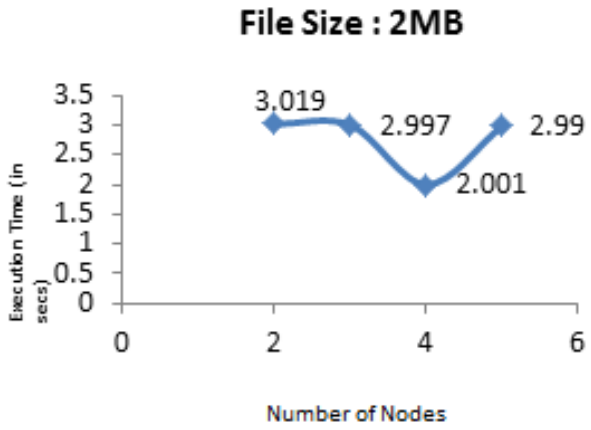

Fig10. *Graph for Execution time vs. No. of slave nodes for decompression*

Figure 10 shows that with the increase in number of nodes the execution time decreases for decompression of 2.0 MB input file.

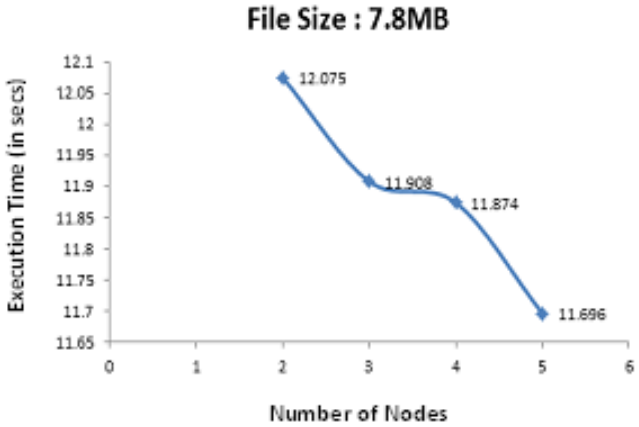

Fig11. *Graph for Execution time vs. No. of slave nodes for decompression*

Figure 11 shows that with the increase in number of nodes the execution time decreases for decompression of 7.8 MB input file.

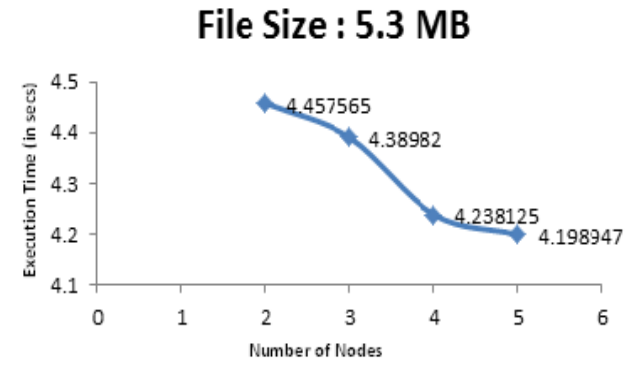

Fig12. *Graph for Execution time vs. No. of slave nodes for compression*

Figure 12 shows that with the increase in number of nodes the execution time decreases for decompression of 5.3 MB input file.

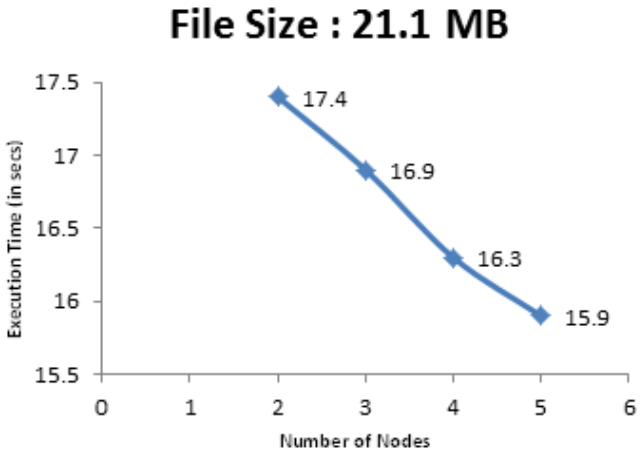

Fig13. *Graph for Execution time vs. No. of slave nodes for compression*

Figure 13 shows that with the increase in number of nodes the execution time decreases for decompression of 21.1 MB input file.

Thus the graphs in figures 10, 11, 12 and 13 show that with increase in number of slave nodes the time to compress/decompress the data decreases. The data obtained for the graphs is an average of values obtained for multiple executions and the results may vary for in case if the cluster network load is high. Also the results are dependent on the hardware resources like CPU, LAN, etc.

CONCLUSIONS AND FUTURE ENHANCEMENTS

Cluster computing is gaining popularity day by day due to various factors of technological advancements in software as well as hardware. In such a scenario, meeting the challenge of load balancing in the cluster computing environment has become a significant need.

Thus this paper demonstrates load balancing algorithm for data intensive application in case of parallel data compression/decompression being executed on a compute cluster.

The requirements stated in section III have been met and the load balancing is being carried out efficiently which has led to an efficient use of the compute cluster.

However, cent percent efficiency is not possible to be obtained from the paper developed because of limiting factors related to communication network such as bandwidth and data transfer rate. In order to extract maximum performance from the system, high speed network switches like Infiniband needs to be configured which would make the network more suitable for running the application system.

REFERENCES

[1] Igor Grudenic, Stjepan Gros, Nikola Bogunovic, "Load Balancing MPI algorithm for high througput Applications", Faculty of Electrical Engineering and Computing, University of Zagreb Unska 3,10000 Zagreb Croatia.

[2] Yawei Li and Zhiling Lan, "A Survey of Load Balancing in Grid Computing", Computational and Information science, Springer-Verlag Berlin Heidelberg, 2004, pp. 280–285.

[3] Jeff Gilchrist, "Parallel Data Compression with BZIP2", Elytra Enterprises Inc. Unit 1, 5370 Canotek Road Ottawa, Canada K1J 9E6

[4] Silberschatz, Galvin,Gagne; "Operating System Concepts";6th Edition

[5] Andrews, Gregory R. (2000), Foundations of Multithreaded, Parallel, and Distributed Programming, Addison–Wesley, ISBN 0-201-35752-6.

[6] Cluster Computing - en.wikipedia.org/wiki/Cluster_Computing

[7] www.mcs.anl.gov/research/papers/mpich2/

I. Raicu, I. Foster, et al. "The Quest for Scalable Support of Data Intensive Workloads in Distributed Systems", ACM HPDC 2009.

[8] D. Zhao, Z. Zhang, X. Zhou, T. Li, K. Wang, D. Kimpe, P. Carns, R. Ross, and I. Raicu. "FusionFS: Towards Supporting Data-Intensive Scientific Applications on Extreme-Scale High-Performance Computing Systems." IEEE BigData, 2014.

[9] Henning Meyerhenke. Dynamic load balancing for parallel numerical simulations based on repartitioning with disturbed difusion. In Proc. International Conference on Parallel and Distributed Systems (ICPADS'09), pages 150-157. IEEE Computer Society, 2009

AUTHORS' BIOGRAPHY

V Vijayasherly, is working for VIT University, Vellore, India in the School of Computing Science and Engineering. She received her B.E. in Computer Science and Engineering from Madras University, Tamil Nadu and M. Tech in Computer Science Engineering from VIT University, India. Prof. Vijayasherly has several years of experience working in the academia and research in Distributed Computing.

L Shalini, is with VIT University, Vellore, India, in the School of Computing Science and Engineering. She received her M.Sc., M.Phil. from Bharathiyar University, MCA from Bharathidasan University, India. Prof. Shalini has several years of experience and her current research interests include Theory of Computation, Algorithms and Data Mining. She has received awards from Computer Society of India in 2015 for her outstanding contribution in the past years.

Mr. Chandresh Sharma, completed B.Tech. Computer Science and Engineering in VIT University, Vellore, India. He is currently working as Senior Developer - Level 3 in CommVault Systems and responsible for design, development of Backup-Recovery module for IBM DB2 database, Amazon RDS.

Mr. Manasvi Pandya, received B.Tech. Computer Science and Engineering from VIT University, Vellore, India and PGDIE from National Institute of Industrial Training, Mumbai. He worked as SAP Business Technology Analyst in Deloitte, India.