ISSN: 2393-8528

IJICSE
Open Access

Contents lists available at www.ijicse.in

International Journal of Innovative Computer Science & Engineering

Volume 2 Issue 3; July-August-2015; Page No. 44-51

Implementation of Randomized Hydrodynamic Load Balancing Algorithm using Map Reduce framework on Open Source Platform of Hadoop

Rashi Saxena[1], Sarvesh Singh[2]

[1](Research Scholar, Jayoti Vidyapeeth Women's University, India)

[2](HOD, Computer Science & Engineering Department, Jayoti Vidyapeeth Women's University, India)

ARTICLE INFO

Received: 23 June 2015
Accepted 18 July 2015

Corresponding Author:

Rashi Saxena

Department of Computer Science and Engineering, Jayoti Vidyapeeth Women's University, Jaipur

Key words: *Dynamic Load Balancing, Randomized Hydrodynamic Load Balancing.*

ABSTRACT

Load balancing is performed to achieve the optimal use of the existing computational resources as much as possible whereby none of the resources remains idle while some other resources are being utilized. Balanced load distribution can be achieved by the immigration of the load from the source nodes which have surplus workload to the comparatively lightly loaded destination nodes. Applying load balancing during run time is called dynamic load balancing (DLB). This paper presents the randomized hydrodynamic load balancing (RHLB) method which is a hybrid method that takes advantage of both direct and iterative methods. Using random load migration as a direct method, RHLB approach intends to solve the problems derived from the exceptional instantaneous load rises, and diffuse the surplus workload to relatively free resources. Besides, using hydrodynamic approach as an iterative method, RHLB aims to consume minimum possible system resources to balance the common workload distributions. The results of the experiments designate that, RHLB outruns other iterative based methods in terms of both balance quality and the total time of the load balancing process.

INTRODUCTION

High performance computing (HPC) systems are very effective at solving problems that can be partitioned into small tasks. Distributing tasks among computing nodes (CNs) is the key issue to increase the throughput and the utilization of an HPC system. Any strategy, used for even load distribution among CNs, is called load balancing (LB). The main purpose of load balancing is to keep all computational resources in use as much as possible, and not to leave any resource in idle state while some other resources are being utilized. Conceptually, a load balancing algorithm implements a mapping function between the tasks and CNs. Load balancing is especially more crucial in case of the no uniform computational patterns and/or in heterogeneous environments. In dynamic load balancing (DLB), the load is balanced during the computation process. The primary mechanism of the DLB is the notion of migration of tasks among CNs. Given the instantaneous load distribution of the whole system, queued tasks, waiting to be executed, are migrated from a heavy loaded CN to a relatively free CN. Thus, even in case of non-uniform computational patterns or instantaneous computational power needs, balanced workload distribution among the whole system can be achieved and the utilization of the CNs can be increased. However, any DLB method has to deal with some issues like forecasting the right time to invoke the balancing process, selection of the node that makes the load balancing decisions and how to migrate the load among the CNs. Dynamic load balancing techniques can be classified according to migration operation as direct and iterative methods. Direct methods aim to find out the last execution node of the surplus load in one load balancing step. Hence, direct methods need actual information about the load distribution of the whole system. These kinds of methods are most suitable for systems equipped with a broadcast mechanism [2]. Load balancing process is performed by migration of the partition objects that contain the surplus workload. Load transfer is controlled by two dynamically selected threshold values. The process is invoked by the sender (highly loaded) node and the invocation mechanism is

periodically triggered. They also extended their system with a prediction model [12]. In the prediction model, the system gathers the task execution times of the nodes and reveals a statistical solution for the migration problem. By broadcasting this table to each node periodically, idle nodes can be detected by the utilized nodes so that load transfer operation can be disposed. Although direct dynamic load balancing methods enable the system to reach the balanced state in a short amount of time, the fact that they consume large amounts of resource to compute the destination CN for migrating node composes a weakness. Iterative methods alternatively find out the last execution point by migrating load iteratively. Step by step, by transferring load to lightly loaded neighbors, surplus load is migrated to its final destination. Thus, unlike direct methods, balanced state of the whole system is granted after several iterations. A CN using an iterative method has to manage only its direct neighbors, does not need the knowledge about the whole system. So, its computational need is smaller compared to the cases where the whole system should be taken into consideration as in direct methods. However, an instantaneous change in the state of another CN that is not its direct neighbor does not immediately affect the local load distribution of the iterative method. In addition, aging which refers to the length of the delay from the time of the determination of the workload information to the time it is used in balancing decisions, is less crucial in iterative methods than direct methods [8]. By using iterative dynamic load balancing methods, both the aging of the load information of the CNs and the computational power can be reduced. RHLB approach takes advantage of both direct and iterative methods and aims to avoid the weaknesses of both methods. Using random load migration as in direct methods, RHLB approach intends to solve the problems derived from the exceptional instantaneous load rises, and diffuse the surplus workload to proportionally free resources. Also using hydrodynamic approach as an iterative method, RHLB aims to consume minimum possible system resources to balance the unexceptionally regular workload distributions. The results of the experiments designate that, RHLB outruns other iterative methods by decreasing the total executing time of the load balancing process. Besides, in RHLB idle time of resources is almost half of its closest competitor in dynamic load balancing process.

[1] Randomized Hydrodynamic Approach

RHLB approach aims to distribute the workload that both exists before the execution and generated during the execution, in an efficient manner. Using hydrodynamic approach as an iterative method, RHLB intends to consume minimum available system resources to balance the regular workload distributions. Besides, by using random load distribution like a direct method, RHLB intends to solve the problems that arise from the exceptional instantaneous load rises, and diffuse the surplus workload to relatively free resources within the minimum amount of time. Consequently, using a hybrid method enables RHLB to avoid the weaknesses of both methods. RHLB executes an analogous algorithm like hydrodynamic approach with some crucial differences. Besides, it consists of two different phases called the randomized section and the hydrodynamic section. To begin with, it is not necessary to stop all the running CNs to execute a load balancing process in RHLB. Each neighborhood is free to balance its local workload at different times. Having these properties, RHLB exhibits a non-blocking structure. To construct this structure, RHLB continuously checks if any of the load balancing messages of other CNs have been arrived to the receiving node's incoming buffer area, which is managed by message passing interface (MPI) library. Hereby, all incoming messages and requests can be queued so that they are analyzed by the main load balancing thread whenever they are permitted to execute the load balancing process. Being a nonblocking method, RHLB arrives to the balanced state more slowly compared to the HA; however each CN spends less time on load balancing process compared to the whole process time. As for the other difference, RHLB method uses randomized migrations which depicted in Figure 2. A trigger mechanism is defined to determine whether to use the direct load balancing technique on the current iteration step or not. The trigger mechanism can be described as; if the load amount of the balancing CN is greater than a certain value, then the CN is supposed to constitute an undesirable peak on the load gradient of the whole system. Therefore direct load balancing process takes place on that CN. In the randomization section, RHLB aims to determine destination node(s) that can be used to transfer instant load. In this determination phase, all the CNs in the system except the inquiring node and its direct neighbors constitute the candidates. Among these candidates, one or more CNs (the number of CNs is proportional to the total

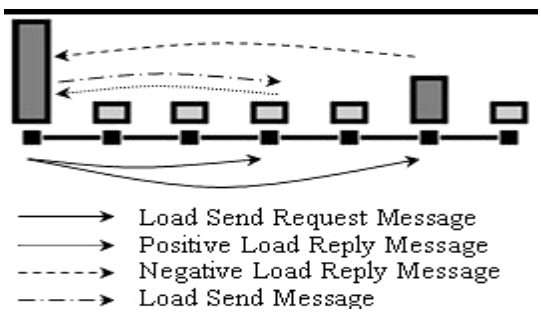

Fig. 1: Randomized Process

Systems size), are selected randomly as destination nodes. After the determination phase, instantaneous

load send requests are sent to the destination CN(s). Each node that gets an instantaneous load request, replies in accordance with the load information of its neighborhood. If the load amount of the inquired CN is less than that of its neighbors, it sends a positive acknowledgement message to the originator of the request message. After the approval, load immigration process takes place and the peak load is diffused through the system roughly. In case of a rejection, inquiring node waits for the other replies. After receiving all the replies and transferring the surplus load, node runs the hydrodynamic load balancing section of the RHLB method. As stated before, hydrodynamic load balancing section of the RHLB is a non-blocking version of the classic hydrodynamic approach.

[2] System Configuration

SINGLE-NODE INSTALLATION (Running Hadoop on Ubuntu (Single node cluster setup)

Distributed File System, running on Ubuntu Linux. Hadoop is a framework written in Java for running applications on large clusters of commodity hardware and incorporates features similar to those of the Google File System (GFS) and of the MapReduce computing paradigm. Hadoop's HDFS is a highly fault-tolerant distributed file system and, like Hadoop in general, designed to be deployed on low-cost hardware. It provides high throughput access to application data and is suitable for applications that have large data sets.

Before we start, we will understand the meaning of the following:

- **DataNode:** A DataNode stores data in the Hadoop File System. A functional file system has more than one DataNode, with the data replicated across them.
- **NameNode:** The NameNode is the centerpiece of an HDFS file system. It keeps the directory of all files in the file system, and tracks where across the cluster the file data is kept. It does not store the data of these file itself.
- **Jobtracker**: The Jobtracker is the service within hadoop that farms out MapReduce to specific nodes in the cluster, ideally the nodes that have the data, or at least are in the same rack.
- **TaskTracker:** A TaskTracker is a node in the cluster that accepts tasks- Map, Reduce and Shuffle operations – from a Job Tracker.
- **Secondary Namenode:** Secondary Namenode whole purpose is to have a checkpoint in HDFS. It is just a helper node for namenode.

Prerequisites

Java 6 JDK- Hadoop requires a working Java 1.5+ (aka Java 5) installation. Update the source list

user@ubuntu:~$ sudo apt-get update

or

Install Sun Java 6 JDK

Note:

If you already have Java JDK installed on your system, then you need not run the above command.

To install it

user@ubuntu:~$ sudo apt-get install sun-java6-jdk

The full JDK which will be placed in /usr/lib/jvm/java-6-openjdk-amd64 After installation, check whether java JDK is correctly installed or not, with the following command

user@ubuntu:~$ java -version

Adding a dedicated Hadoop system user

We will use a dedicated Hadoop user account for running Hadoop.

user@ubuntu:~$sudoaddgrouphadoop_group

user@ubuntu:~$ sudo adduser --ingroup hadoop_group hduser1

This will add the user hduser1 and the group hadoop_group to the local machine. Add hduser1 to the sudo group

user@ubuntu:~$ sudo adduser hduser1 sudo

Configuring SSH

The hadoop control scripts rely on SSH to perform cluster-wide operations. For example, there is a script for stopping and starting all the daemons in the clusters. To work seamlessly, SSH needs to be setup to allow password-less login for the hadoop user from machines in the cluster. The simplest way to achieve this is to generate a public/private key pair, and it will be shared across the cluster.

Hadoop requires SSH access to manage its nodes, i.e. remote machines plus your local machine. For our single-node setup of Hadoop, we therefore need to configure SSH access to localhost for the hduser user we created in the earlier. We have to generate an SSH key for the hduser user.

user@ubuntu:~$ su – hduser1

hduser1@ubuntu:~$ ssh-keygen -t rsa -P ""

The second line will create an RSA key pair with an empty password.

Note: P "", here indicates an empty password

You have to enable SSH access to your local machine with this newly created key which is done by the following command.

hduser1@ubuntu:~$ cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys

The final step is to test the SSH setup by connecting to the local machine with the hduser1 user. The step is also needed to save your local machines host key fingerprint to the hduser user's known hosts file.

hduser@ubuntu:~$ ssh localhost

If the SSH connection fails, we can try the following (optional):

- Enable debugging with ssh -vvv localhost and investigate the error in detail.
- .Check the SSH server configuration in /etc/ssh/sshd_config. If you made any changes to the SSH server configuration file, you can force a configuration reload with sudo /etc/init.d/ssh reload.

INSTALLATION

Now, I will start by switching to hduser

hduser@ubuntu:~$ su - hduser1

- Now, download and extract Hadoop 1.2.0
- Setup Environment Variables for Hadoop

Add the following entries to .bashrc file

```
# Set Hadoop-related environment variables
export HADOOP_HOME=/usr/local/hadoop
# Add Hadoop bin/ directory to PATH
export PATH= $PATH:$HADOOP_HOME/bin
```

Configuration

- ***hadoop-env.sh***

Change the file: conf/hadoop-env.sh

#export JAVA_HOME=/usr/lib/j2sdk1.5-sun to in the same file

export JAVA_HOME=/usr/lib/jvm/java-6-openjdk amd64 (for 64 bit)

export JAVA_HOME=/usr/lib/jvm/java-6-openjdk-amd64 (for 32 bit)

- **conf/*-site.xml**

Now we create the directory and set the required ownerships and permissions

hduser@ubuntu:~$ sudo mkdir -p /app/hadoop/tmp

hduser@ubuntu:~$ sudo chown hduser:hadoop /app/hadoop/tmp

hduser@ubuntu:~$ sudo chmod 750 /app/hadoop/tmp

The last line gives reading and writing permissions to the /app/hadoop/tmp directory

Error: If you forget to set the required ownerships and permissions, you will see a java.io.IO Exception when you try to format the name node.

Paste the following between <configuration>

- ***In file conf/core-site.xml***

```
<property>
  <name>hadoop.tmp.dir</name>
  <value>/app/hadoop/tmp</value>
  <description>A base for other temporary directories.</description>
</property>

<property>
  <name>fs.default.name</name>
  <value>hdfs://localhost:54310</value>
  <description>The name of the default file system. A URL whose scheme and authority determine the FileSystem implementation. The url's scheme determines the config property (fs.SCHEME.impl) naming the FileSystem implementation class. The url's authority is used to determine the host, port, etc. for a filesystem.</description>
</property>
```

- ***In file conf/mapred-site.xml***

```
<property>
<name>mapred.job.tracker</name>
  <value>localhost:54311</value>
  <description>The host and port that the MapReduce job tracker runs at. If "local", then jobs are run in-process as a single map and reduce task.
  </description>
</property>
```

- ***In file conf/hdfs-site.xml***

```
<property>
  <name>dfs.replication</name>
  <value>1</value>
  <description>Default block replication.
  The actual number of replications can be specified when the file is created. The default is used if replication is not specified in create time.
  </description>
</property>
```

Formatting the HDFS filesystem via the NameNode

To format the filesystem (which simply initializes the directory specified by the dfs.name.dir variable). Run the command

hduser@ubuntu:~$ /usr/local/hadoop/bin/hadoop namenode –format

Starting your single-node cluster

Before starting the cluster, we need to give the required permissions to the directory with the following command

hduser@ubuntu:~$ sudo chmod -R 777 /usr/local/hadoop

Run the command

hduser@ubuntu:~$ /usr/local/hadoop/bin/start-all.sh

This will start up a Namenode, Datanode, Jobtracker and a Tasktracker on the machine.

hduser@ubuntu:/usr/local/hadoop$ jps

Errors:

If by chance your datanode is not starting, then you have to erase the contents of the folder /app/hadoop/tmp The command that can be used

hduser@ubuntu:~:$ sudo rm –Rf /app/hadoop/tmp/*

You can also check with netstat if Hadoop is listening on the configured ports. The command that can be used

hduser@ubuntu:~$ sudo netstat -plten | grep java

Errors if any, examine the log files in the /logs/ directory.

Stopping your single-node cluster

Run the command to stop all the daemons running on your machine.

hduser@ubuntu:~$ /usr/local/hadoop/bin/stop-all.sh

ERROR POINTS:
If datanode is not starting, then clear the tmp folder before formatting the namenode using the following command
hduser@ubuntu:~$ rm –Rf /app/hadoop/tmp/*

Note:
The masters and slaves file should contain localhost.
In /etc/hosts, the ip of the system should be given with the alias as localhost.
Set the java home path in hadoop-env.sh as well bashrc.

MULTI-NODE INSTALLATION
Running Hadoop on Ubuntu Linux (Multi-Node Cluster)
We will build a multi-node cluster merge two or more single-node clusters into one multi-node cluster in which one Ubuntu box will become the designated master but also act as a slave , and the other box will become only a slave.

Prerequisites
Configuring single-node clusters first, here we have used two single node clusters. Shutdown each single-node cluster with the following command
user@ubuntu:~$ bin/stop-all.sh

Networking
The easiest is to put both machines in the same network with regard to hardware and software configuration. Update /etc/hosts on both machines .Put the alias to the ip addresses of all the machines. Here we are creating a cluster of 2 machines , one is master and other is slave 1
hduser@master:$ cd /etc/hosts
Add the following lines for two node cluster
10.105.15.78 master (IP address of the masternode)
10.105.15.43 slave1 (IP address of the slave node)

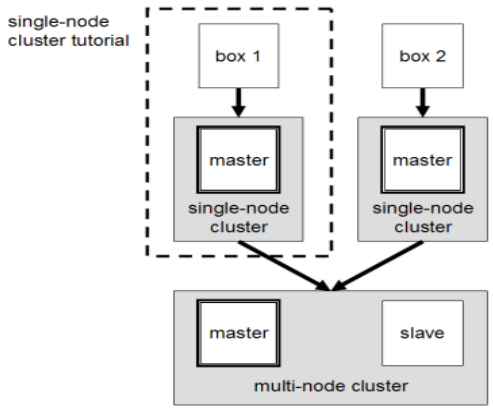

Fig. 2 .Master-Slave Architecture

SSH access
The hduser user on the master (aka hduser@master) must be able to connect: to its own user account on the master - i.e. ssh master in this context. to the hduser user account on the slave (i.e. hduser@slave1) via a password-less SSH login. Add the hduser@master public SSH key using the following command
hduser@master:~$ ssh-copy-id -i $HOME/.ssh/id_rsa.pub hduser@slave1

Connect with user hduser from the master to the user account hduser on the slave.
From master to master
hduser@master:~$ ssh master

From master to slave
hduser@master:~$ ssh slave1

Hadoop
Cluster Overview
This will describe how to configure one Ubuntu box as a master node and the other Ubuntu box as a slave node.
Configuration
conf/masters
The machine on which bin/start-dfs.sh is running will become the primary NameNode. This file should be updated on all the nodes. Open the masters file in the conf directory
hduser@master/slave :~$ /usr/local/hadoop/conf
hduser@master/slave :~$ sudo gedit masters
Add the following line
Master

conf/slaves
This file should be updated on all the nodes as master is also a slave. Open the slaves file in the conf directory
hduser@master/slave:~/usr/local/hadoop/conf$ sudo gedit slaves
Add the following lines
Master
Slave1

- ***conf/*-site.xml (all machines)***

Open this file in the conf directory
hduser@master:~/usr/local/hadoop/conf$ sudo gedit core-site.xml

Change the fs.default.name parameter (in conf/core-site.xml), which specifies the NameNode (the HDFS master) host and port.
conf/core-site.xml (ALL machines .ie. Master as well as slave)
<property>
<name>fs.default.name</name>
<value>hdfs://master:54310</value>
<description>The name of the default file system. A URI whose scheme and authority determine the FileSystem implementation. The uri's scheme determines the config property (fs.SCHEME.impl) naming the FileSystem implementation class. The uri's authority is used to determine the host, port, etc. for a filesystem.</description>
</property>

- ***conf/mapred-site.xml***

Open this file in the conf directory
hduser@master:~$ /usr/local/hadoop/conf
hduser@master:~$ sudo gedit mapred-site.xml

Change the mapred.job.tracker parameter (in conf/mapred-site.xml), which specifies the JobTracker (MapReduce master) host and port.
conf/mapred-site.xml (ALL machines)
<property>
<name>mapred.job.tracker</name>
<value>master:54311</value>
<description>The host and port that the MapReduce job tracker runs at. If "local", then jobs are run in-process as a single map and reduce task.
</description>
</property>

- ***conf/hdfs-site.xml***

Open this file in the conf directory
hduser@master:~$ /usr/local/hadoop/conf
hduser@master:~$ sudo gedit hdfs-site.xml
Change the dfs.replication parameter (in conf/hdfs-site.xml) which specifies the default block replication. We have two nodes available, so we set dfs.replication to 2.

- ***conf/hdfs-site.xml (ALL machines)***

Changes to be made
<property>
<name>dfs.replication</name>
<value>2</value>
<description>Default block replication.
The actual number of replications can be specified when the file is created. The default is used if replication is not specified in create time.
</description>
</property>

Formatting the HDFS filesystem via the NameNode
Format the cluster's HDFS file system
hduser@master:~/usr/local/hadoop$ bin/hadoop namenode –format

Starting the multi-node cluster
Starting the cluster is performed in two steps.
We begin with starting the HDFS daemons: the NameNode daemon is started on master, and DataNode daemons are started on all slaves (here: master and slave). Then we start the MapReduce daemons: the JobTracker is started on master, and TaskTracker daemons are started on all slaves (here: master and slave). Cluster is started by running the commnd on master
hduser@master:~$ /usr/local/hadoop
hduser@master:~$ bin/start-all.sh

By this command: The NameNode daemon is started on master, and DataNode daemons are started on all slaves (here: master and slave). The JobTracker is started on master, and TaskTracker daemons are started on all slaves (here: master and slave)
To check the daemons running , run the following commands
hduser@master:~$ jps

On slave, datanode and jobtracker should run.
hduser@slave:~/usr/local/hadoop$ jps

Stopping the multi-node cluster
To stop the multinode cluster , run the following command on master pc
hduser@master:~$ cd /usr/local/hadoop
hduser@master:~/usr/local/hadoop$ bin/stop-all.sh

ERROR POINTS:
Number of slaves = Number of replications in hdfs-site.xml
also number of slaves = all slaves + master(if master is also considered to be a slave) When you start the cluster, clear the tmp directory on all the nodes (master+slaves) using the following command
hduser@master:~$ rm -Rf /app/hadoop/tmp/*

Configuration of /etc/hosts , masters and slaves files on both the masters and the slaves nodes should be the same. If namenode is not getting started run the following commands:
To give all permissions of hadoop folder to hduser
hduser@master:~$ sudo chmod -R 777 /app/hadoop

This command deletes the junk files which gets stored in tmp folder of hadoop
hduser@master:~$ sudo rm -Rf /app/hadoop/tmp/*

[3] Experimental Results
All tests in this paper are performed on a cluster of workstations where each workstation has two dual cored Intel Xeon CPUs. Each single core is assumed to be a separate CN on logical ring topology. As an interprocessor communication tool, message passing interface (MPI) library is used. Processed workload is simple an acoustic normal mode calculation [3]. While some cases reflect the most difficult instances of the load distribution problem, some of them are used to demonstrate the state that has no need to use any load balancing method. The tests can be divided into two main groups according to purpose of each test. First test group aims to show the main differences between hydrodynamic based methods (HA and RHLB) and rest of the iterative methods. The second group intends to demonstrate the behavior of the hydrodynamic based solutions at particular occasions. All results are gathered from the average values of five test repetitions. First test group consists of three different

test cases each having the purpose of underlining the differences between the iterative methods that have been presented above. All the tests in the first test group are run on a 12 node cluster that has a logical ring topology.

This graph demonstrates that without an unexpected load addition and a peak in the load distribution gradient, iterative methods behaves similarly and the difference between the run times is minimal. A fair load distribution means that all processors can run their tasks in a fully utilized manner, without a load balancing mechanism. Therefore this case considers the additional load generated by processing the balancing algorithm of each iterative method itself. This graph emphasizes that in case of fair load distribution, maximum run time of the total process is directly proportional to the simplicity of the load balancing algorithm.

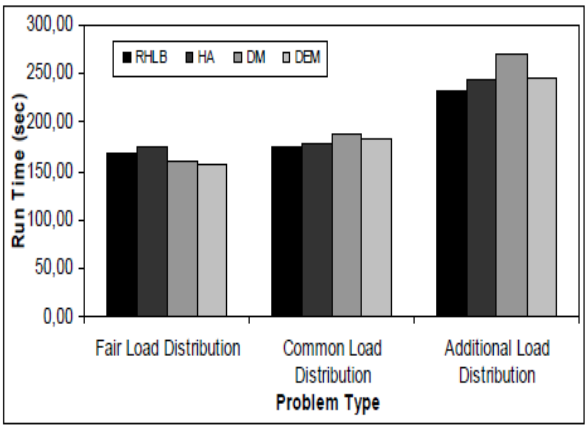

Fig. 3.Results for the first test group

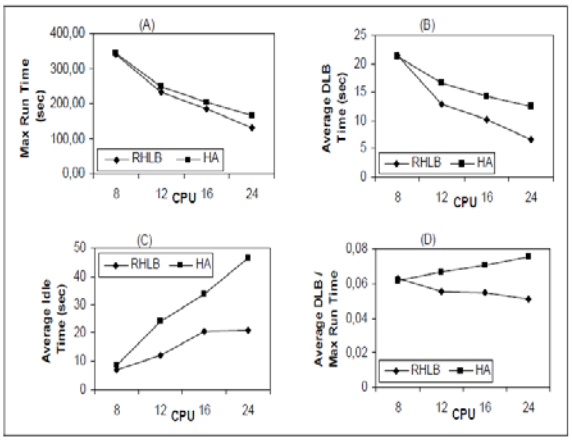

Fig. 4. (A) Maximum run time, (B) the average dynamic load balancing time, (C) the average idle time and (D) the ratio of average DLB time over maximum run time comparison between RHLB and HA methods.

The results of this case are shown in Figure 5. As it can be deduced from the figure, RHLB presents an improvement on the total run time with the use of randomized method. To sum up, the test results discussed above demonstrate that RHLB is the most effective load balancing technique not only among the hydrodynamic based models but also among all the mentioned iterative models. This fact stems from the randomized structure of the RHLB which makes it a hybrid method. Taking the advantage of randomization, RHLB distributes the unexpected peak load in a shorter amount of time than any other iterative method.

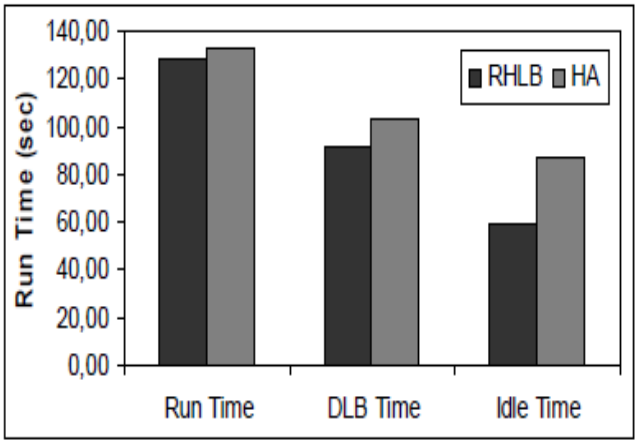

Fig. 5. Comparison of the hydrodynamic based methods in an exceptional load distribution situation.

Therefore total CN utilization is increased and maximum run time of the whole process is reduced.

[4] CONCLUSION

The load balancing problem is crucial in multiprocessor systems that have large number of CNs. In this study, a randomized hydrodynamic load balancing approach is presented as a solution to the dynamic load balancing problem for a network of heterogeneous CNs. RHLB can be considered as an extension of the hydrodynamic approach which converges to a balanced state geometrically for all configurations [15]. RHLB is invigorated by the use of non-blocking load balancing and the randomized load distributions. By this nature RHLB can also be considered as a hybrid method which combines the advantages of iterative and direct methods. RHLB is experimented for several test cases based on the problem of normal mode theory. As observed in experiments, the RHLB prevails over the existing iterative methods mentioned above, especially in extreme load distribution situations. To conclude, randomized hydrodynamic load balancing approach offers substantial improvements not only in the utilization of resources of the parallel system, but also the total time needed to run dynamic load balancing algorithm itself.

[5] Acknowledgement

I would like to express my special thanks of gratitude to my guide Mr. Sarvesh singh sir who gave me the golden opportunity to write this important paper on the topic load balancing on cloud data centers, which also helped me in doing a lot of research and I came to know about so many new things, I am really thankful to them, and also thankful to IJICSE team to give me this opportunity to publish this paper.

[6] References

1. Chengzhong XU, Burkhard MONIEN, Reinhard LÜNING, Francis C. M. LAU "Nearest neighbor algorithms for load balancing in parallel Computers"
2. Chang-Zhang XU, Francis C.M. LAU "Iterative dynamic load balancing in multicomputers", Journal of Operational Research Society, Vol. 45 No. 7, July 1994, pp. 786–796
3. Serkan AKSOY, Hüseyin A. SERİM "Technical report on normal mode method - Normal mod yötemi teknik analiz raporu", unpublished.
4. G. CYBENKO "Load balancing for ditributed memory multiprocessors", Journal of Parallel and Distributed Computing, Vol. 7, 1989, pp. 279–307
5. F.C.H. LIN, R.M. KELLER "The gradient model load balancing method", IEEE Transactions on Software Engineering, Vol. 13, 1987, pp. 32–38
6. W. SHU, L.V. KALE "A dynamic scheduling strategy for the chare kernel systems", In Proceedings of Supercomputing, 1989, pp. 389–398
7. Marta BELTRÀN, Jose BOSQUE "Information policies for load balancing on heterogeneous systems", IEEE International Symposium on Cluster Computing and the Grid, 2005, pp. 970 – 979
8. Mark H. WILLEBEEK-LEMAIR, Anthony P. REEVES "Strategies for dynamic load balancing on highly parallel computers", IEEE Transactions on parallel and distributed systems, Vol. 4 No. 9, 1993, pp. 979 – 993
9. Malcolm YOKE, Hean LOW "Dynamic load-balancing for BSP time warp", 35th Annual Simulation Symposium, 2002
10. Deyu QI, Weiwei LIN "TGrid: A next grid environment", First International Multi-Symposiums on Computer and Computational Sciences, 2006.
11. Wei JIE, Wentong CAI, Stephen J. Turner "Dynamic load-balancing using prediction in a parallel object-oriented system", IEEE, 2001
12. Giuseppe Di FATTA, Michael R. BERTHOLD "Dynamic load balancing for the distributed mining of molecular structures", IEEE Transactions on Parallel and Distributed Systems, Vol. 17 No. 8, August 2006.
13. Jacques M. BAHI, Sylvain CONTASSOT, Raphael COUTURIER "Dynamic load balancing and efficient load estimators for asynchronous iterative algorithms", IEEE Transactions on Parallel and Distributed Systems, Vol. 16 No. 4, April 2006.
14. Chi-Chung HUI, Samuel T. CHANSON "A hydrodynamic approach to heterogeneous dynamic load balancing in a network of computers", IEEE International Conference on Parallel Processing, 1996.