A Comparative Study of Grid Load Balancing

Anuj Kumar
Research Scholar
Department of Computer Science,
Gurukul Kangri Vishwavidyalaya,
Haridwar (UK), India

Heman Pathak
Associate Professor
Department of Computer Science,
Gurukul Kangri Vishwavidyalaya,
Haridwar (UK), India

ABSTRACT
Grid computing is an infrastructure for supporting complex computing. That is organized with the different scale of computational and network resources. In addition of that it is capable to process the request of multiple users. In this context the effective scheduling of resources according to the submitted tasks are required for efficient computational outcome. This paper provides an experimental study for the three popular load balancing techniques i.e. space shared, distributed and Hierarchical. The experiments are performed using GridSim technology and with help of JAVA based implemented scripts. The two kinds of experiments are reported in this work first with the increasing workload and secondly with the varying number of resources i.e. number of machines and number of processing elements. The different experiments show that the space shared is a promising algorithm for load balancing but the hierarchical load balancing algorithm comparatively enhances the performance of grid. Finally, the distributed load balancing algorithm demonstrates its superiority among all of them.

General Terms
Computational Grids, Grid Computing, Load Balancing, Distributed Grid, Hierarchical Grid

Keywords
Grid Computing, Load Balancing, Performance Evaluation, Distributed Algorithm, Space Shared, Hierarchical Algorithm.

1. INTRODUCTION
Grid computing enables the creation of a single IT infrastructure that can be shared by multiple business processes. Grid computing is a group of networked computers which work together as a virtual supercomputer to perform large tasks, such as analyzing huge sets of data or weather modeling. Through the cloud, you can assemble and use vast computer grids for specific time periods and purposes, paying, if necessary, only for what you use to save both the time and expense of purchasing and deploying the necessary resources yourself. Also by splitting tasks over multiple machines, processing time is significantly reduced to increase efficiency and minimize wasted resources. Unlike with parallel computing, grid computing projects typically have no time dependency associated with them. They use computers which are part of the grid only when idle and operators can perform tasks unrelated to the grid at any time. Security must be considered when using computer grids as controls on member nodes are usually very loose. Redundancy should also be built in as many computers may disconnect or fail during processing [1] [2].

Normally, grid works on different tasks in network, but that is also suitable to working on particular applications. That is designed for solving large problems with maintaining flexibility by dividing into number of smaller problems. Computing grids works on multiuser environment that offers discontinuous demands of huge information processing. Grids consist of a variety of resources such as software and hardware, computer languages, and frameworks. That is connected through a network or by using open standards to achieve a common goal [3].

1.1 Grid Computing System
Grid computing is an extension of distributed computing that incorporates coordinating and sharing of computational power, data storage and network resources across dynamic and geographically dispersed organizations. Rapid growth in use of computers has increased the number of applications which uses the shared hardware and software resources (e.g. memory, processor, files etc.) and ultimately increased the amount of submitted tasks across internet. Problem can be solved if we distribute the applications across different computer, in such a manner that it reduces the task response time and the overhead on a single computer. Proper distribution of applications across different available resources is termed as Load Balancing [4].
The Computational Grid category represents a system that has a higher aggregate computational capacity available for single applications. These can be additionally subdivided into circulated supercomputing and high throughput classes relying upon how the total limit is used. A circulated supercomputing Grid executes the application in parallel on different machines to decrease the finish time of a task.

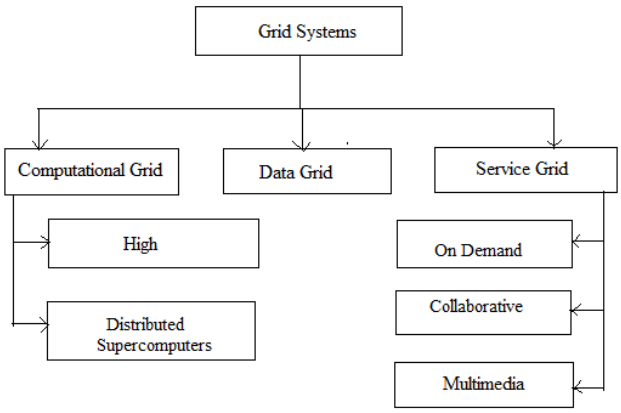

Figure 1.1 Grid System Taxonomy

The Data Grid classification is for frameworks that give a foundation for combining new data from information archives, for example, advanced libraries or information stockrooms that are conveyed in a wide territory arrange. Computational Grids additionally need to give information benefits however the significant contrast between a Data Grid and a computational Grid is the specific framework gave to applications to capacity administration and information get to.

The Service Grid category is for frameworks that give benefits that are not given by any single machine. This class is

additionally subdivided into On-Demand, Collaborative, and Multimedia Grid systems [5] [6].

1.2 Resource Management in Grid

Efficient resource management is one of the fundamental requirements in grid computing. Resource management is required in an environment where resources are quite limited and need to be utilized properly. The term resource management [7] refers to manage the different types of resources like bandwidth, processing power, etc. so that they can be efficiently utilized and satisfy the need of the users and as they are limited in the environment so to allocate to maximum number of sites. As resources are limited in the environment and number of users are far much greater than the number of resources, so we need to perform resource management so that effective utilization of resources take place. The various challenges that resource management in grid environment faces are as follows [8]:

- Satisfactory end to end performance through multiple domains
- Availability of computational resources needs to be checked in order to have knowledge regarding the resources present in the environment.
- Handle of conflicts between common resource demands.
- Fault tolerance.
- Inter domain compatibility.

The resources that are united in grid are geographically distributed and different individuals or organizations own each of them. Additionally they have their own access policies, processing cost, and mechanism. The resource administrators are responsible to manage and control using their desired management and scheduling system. That is also called local scheduler and grid users are expected to honor that and make sure they do not interfere with resource owners' policies. They may charge diverse costs for various framework clients for their asset utilization and it might differ every now and then [9]. As given in figure 2 the connection between different network segments are finished utilizing numerous means, which are as per the following [10]:

- The grid client pursues their application and investigation and determining their necessity, presents their undertakings to grid resource broker (GRB).
- GRB gathers all asset data and perform asset disclosure.
- After approving client and resource(s) GRB plan the undertaking to fitting resource(s) or registering hubs.
- Resource(s) execute the undertaking and return computational outcome to GRB.
- The GRB gathers result and give it to the matrix client.

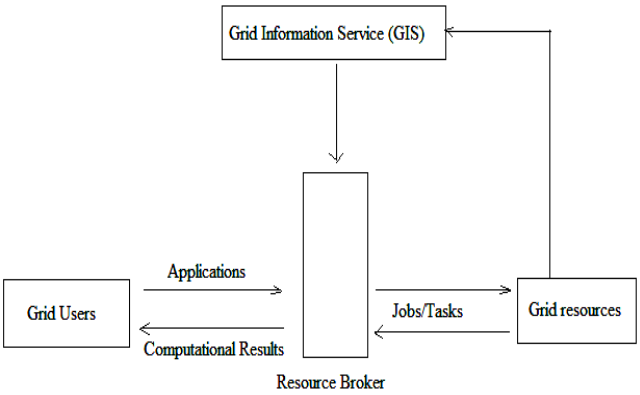

Figure 1.2 Grid Resource Management

1.3 Load Balancing in Grid

Grids functionally combine worldwide distributed computers and information systems for creating a universal source of computing power and information. A key characteristic of Grids is that resources are shared among numerous applications, and therefore, the amount of resources available to any given application is highly fluctuating over time. In present scenario, load balancing plays a key role. For applications that are Grid enabled, the Grid can offer a resource balancing effect by scheduling grid tasks on machines with low utilization. A proper scheduling and efficient load balancing across the grid can lead to improved overall system performance and a lower turnaround time for individual tasks. The main objective of load balancing is to minimize the make span time to enhance resource utilization, exploiting parallelism, maximize throughput and reduce response time through a suitable distribution of the application [11].

2. LITERATURE SURVEY

This section is organized for providing the investigational details about the different relevant concept of load balancing which contributes to the research directions and guidelines for design and development of a unique model for load balancing in a grid environment.

Karim Y. Kabalan et al. [12] discusses several adaptive load sharing algorithms for heterogeneous distributed computing systems. It proposes some modifications to existing algorithms that will account for the delay in transferring tasks from one node to another. It further verifies and validates those proposed changes with some of the simulation results obtained.

Ant colony is a meta-heuristic method that can be instrumental for grid load balancing. Mohsen Amini Salehi et al. [13] presents a reverberate arrangement of versatile fluffy ants. The ants in this condition can make new ones and may likewise submit suicide relying upon existing conditions. Another idea called Ant level load adjusting is exhibited here for enhancing the execution of the instrument. An execution assessment display is likewise determined. The experimental results reveal that the proposed system outperforms its antecedent.

Hongzhang Shan et al. [14] explored the impact of data migration under a variety of demanding grid conditions. They evaluated the grid scheduling algorithm through simulation of computing servers, different groupings of servers into sites, and inter-server networks, utilizing genuine workloads derived from real trace data collected at leading supercomputing centers. The study reveal that in the presence of input / output data migration, the sender initiated distributed approach resulted into reduction of average turnaround time by 60% as compared to the local approach.

Henrik Johansson et al. [15] performed a performance characterization of load balancing algorithms for parallel structured adaptive mesh refinement (SAMR). For a proficient parallel SAMR execution, dividing calculation must be powerfully chosen at run-time as to both application and PC state. Creators describe and analyze a typical apportioning algorithm and a large number of alternative partitioning algorithms. The results prove viability of dynamic algorithm selection and show benefits of using a large number of complementing partitioning algorithms.

In order to avoid the increase in waiting time and response time thereby causing reduction in performances of grid by reducing the resource utilization, an optimal resource sharing algorithm is required. D. Ramesh et al. [16] proposed a hybrid algorithm for streamlining the load sharing. It utilizes two major components viz. Hash Table (HT) and Distributed Hash Table (DHT). The experimental results reveal that the proposed algorithm enhances the performance in comparison to the existing ones.

The Artificial Bee Colony (ABC) algorithm is an optimization algorithm based on the intelligent behavior of honey bee swarm. Preeti Gulia et al. [17] has compared various dynamic load balancing techniques. This paper also reviews various load balancing techniques using swarm intelligence.

Load balancing is one of the critical issues considered for managing a grid environment. Sowmya Suryadevera et al. [18] proposed an Ant Colony Optimization algorithm for stack adjusting in framework figuring which will decide best asset to be dispensed to assignments, in view of asset limit and at the same time balances load of entire resources.

Expecting homogeneous arrangement of hubs connected with homogeneous and quick systems, different load balancing approached were proposed. On the basis of past outcomes, Prakash Kumar et al. [19] introduced a model to enhance the execution and throughput. It proposes proficient calculations with better booking strategies.

Volker Hamscher et al. [20] discuss scheduling structures for computational grids. Simulations were used to evaluate combinations of different Task and Machine Models. They use hierarchical scheduling as common scheduling structure and confirmed the benefit of Backfill.

Abbas Karimi et al. [21] presented an approach for implementing dynamic load balancing with fuzzy logic, that can handle uncertainty and inconsistency, this algorithm shows better response time than round robin and randomize algorithm respectively by 30.84% and 45.45%.

Yagoubi B et al. [22] proposed a hierarchical model for computational grids wherein the grid manager maintaining the global load information is vulnerable to become bottleneck.
Yagoubi B et al. [23] proposed a distributed model by mapping a grid into a forest based model wherein the local load balancing is preferred over the global load balancing.

3. PROPOSED WORK

The main aim of this paper is to provide experimental performance study of two proposed load balancing algorithms for heterogeneous grids viz. Hierarchical and Distributed. In view of the heterogeneity involved in the grid environment, we have chosen Processing Time as the metric for workload estimation which is calculated as the ratio of workload on a node to its processing speed. For efficiency purposes the Binary Heaps are used to maintain the workload information at the manager level as it enables to locate the source and destinations in a migration decision in O(1) time and can be reorganized in $O(\log_2 n)$ time thereby ensuring efficient solution.

The experiments are performed for finding the strengths of these algorithms on the basis of the following four performance parameters:

1. Average Consumed Time: the combination of time unit for different stages involved in computational grid i.e. waiting time, processing time and delay associated for allocating the task to resource.
2. Average Processing Cost: the distributed computing works on shared resources for that purpose the amount of cost is required for utilizing the server resources.
3. Average Waiting time: the amount of time required to start execution when it is in a task queue is known as waiting time.
4. Number of Task Migration: the tasks in a process queue are assigned with two parameters i.e. task length and priority flag. If the task is not processed within a threshold time then the task is migrated to other computational resource.

The performance of these algorithms is studied and analyzed on the aforesaid parameters through simulations using an application developed in Java using GridSim4.0, NetBeans IDE 8.1 and Apache Derby RDBMS on Intel® Core™ i3-5005U 2.00 GHz system. A comparison of the performance of these proposed algorithms is also done with the Space Shared Algorithm for grid load balancing used in GridSim4.0. The dataset for simulation experiments are downloaded from Parallel Workload Archive [24] and partitioned for experimental purposes.

Next section provides the detailed investigation of the experimentations and the obtained values.

4. RESULTS ANALYSIS

The experiments performed on the aforesaid three algorithms are described in this section. To demonstrate the experiments across different performance parameters, as discussed in Section III, are considered their obtained values and the observations are described as:

4.1 Average Consumed Time

The average time consumed is the total amount of time required to process the entire request by the grid resources. That is measured using the following formula:

$$\text{Average Consumed Time} = \frac{1}{N}\sum_{i=1}^{N} CPU_i$$

Where N is the total number of gridlets to be processed and CPU_i is the time for processing i^{th} gridlet.

Figure 4.1 and Table 4.1 show the average time consumed for processing the input gridlets by the configured grid server. In figure the x axis contains the length of different dataset and the corresponding time consumed is reported on Y axis. The average consumed time for space shared algorithm is reported using blue line, distributed algorithm is shown using red line and gray line is used for hierarchical algorithm. According to the observations the mean time consumption for the algorithm is initially higher and with the increased length of dataset it is continuously decreased. Results show that the distributed algorithm for load balancing is more effective than the other two algorithms.

Table 4.1 Average Consumed Time

Dataset Size	Space Shared	Distributed	Hierarchical
250	25816.79	18218.82	23182.77
500	22205.88	19315.63	20541.4
1000	18279.23	15915.15	16706.55
1500	18210.44	17141.09	17185.62
2000	17469.82	16406.33	16020.8
2500	16218.02	14692.92	15574.38

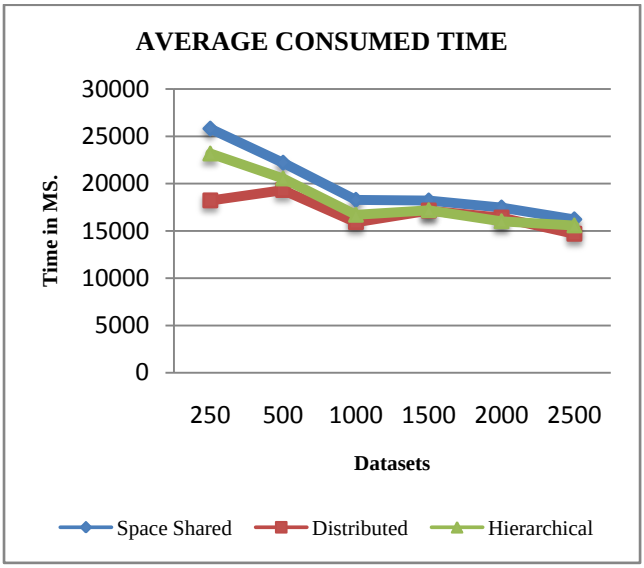

Figure 4.1 Average Consumed Time

4.2 Average Processing Cost

The amount of mean cost required to execute all the submitted tasks is termed as the Average processing cost. The Average processing cost can be computed using the following formula:

$$\text{Average Processing Cost} = \frac{1}{N}\sum_{i=1}^{N} \text{cost}_i$$

Where N is the total number of gridlets to be processed and cost_i is the cost for processing i^{th} gridlet.

Table 4.2 Average Processing Cost

Dataset Size	Space Shared	Distributed	Hierarchical
250	77450.32	54656.47	69548.3
500	66617.63	57946.9	61624.2
1000	54837.7	47745.45	50119.66
1500	54631.31	51423.27	51556.85
2000	52409.45	49218.98	48062.39
2500	48654.22	44078.75	46723.13

The average processing costs of the implemented algorithms are demonstrated using Figure 4.2 and Table 4.2. According to the obtained results, the space shared algorithm is costly as compared to both the Distributed and Hierarchical algorithms. But the distributed algorithm is cost effective than the hierarchical algorithm.

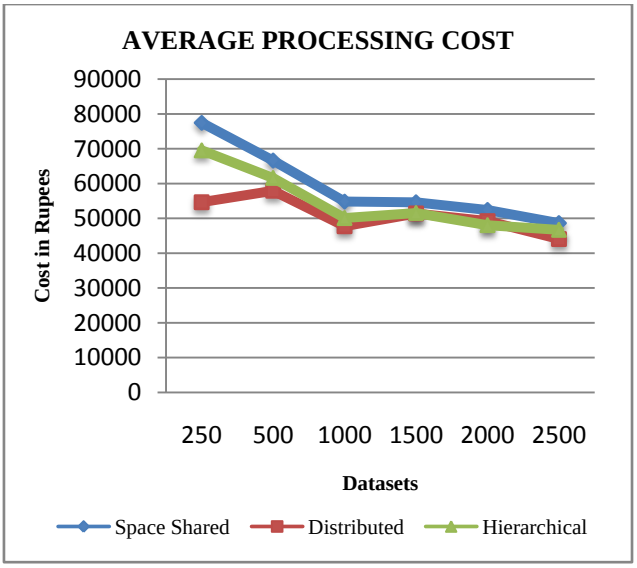

Figure 4.2 Average Processing Cost

4.3 Average Waiting Time

The time consumed before assigning a task to a server resource is termed as the waiting time. The Average Waiting Time is the mean time consumed for allocating the resource. The Average Waiting Time can be computed using the following formula:

$$\text{Average Waiting Time} = \frac{1}{N}\sum_{i=1}^{N} \text{WT}_i$$

Where N is the total number of gridlets to be processed and WT_i is the waiting time for i^{th} gridlet.

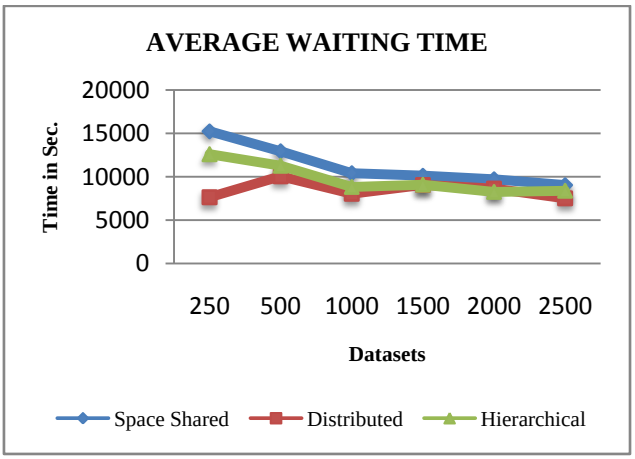

Figure 4.3 Average Waiting Time

Table 4.3 Average Waiting Time

Dataset Size	Space Shared	Distributed	Hierarchical
250	15242.16	7644.199	12608.14
500	12943.98	10053.74	11279.51
1000	10409.49	8045.404	8836.808
1500	10120.78	9051.432	9095.96
2000	9708.813	8645.322	8259.792
2500	9040.187	7515.031	8396.493

The Average Waiting Time for all the algorithms is shown in Figure 4.3 and Table 4.3. According to the experimental results distributed algorithm demonstrate a lower waiting time in comparison to the other two algorithms. Additionally, the hierarchical algorithm shows the efficiency as compared to the space shared algorithm.

4.4 Number of Task Migration

The task migration is an event when task is not executed with current assigned resource due to load or other reasons thus the task transferred to other resource for execution. The total number of tasks migrated for all the three algorithms are described in Figure 4.4 and Table 4.4. In this figure, the X axis contains the dataset size and the Y axis shows the total number of tasks migrated with respect to the size of the dataset.

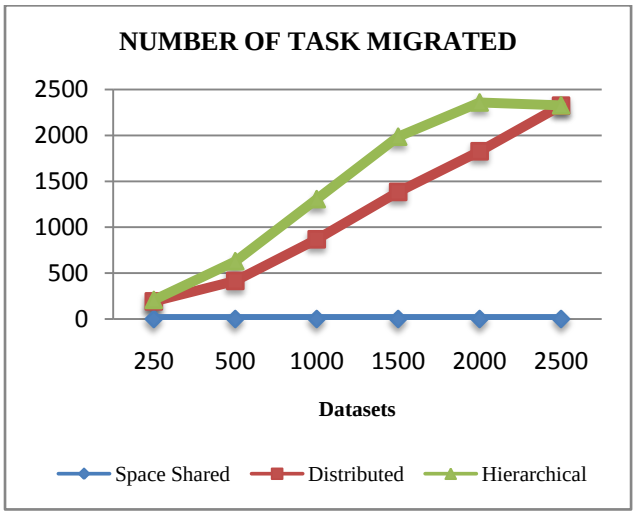

Figure 4.4 Number of Task Migration

Table 4.4 Number of Task Migration

Dataset Size	Space Shared	Distributed	Hierarchical
250	0	192	208
500	0	416	629
1000	0	869	1307
1500	0	1385	1987
2000	0	1828	2360
2500	0	2324	2329

According to the outcomes, the space shared algorithm demonstrates 0 task migration. On the other hand, both the algorithms undergo task migrations. Further, the hierarchical algorithm demonstrates comparatively higher task migration than the distributed algorithm.

5. RESULTS DISCUSSION

This section extends the analysis by interpreting the results obtained through various experiments with two datasets of different seizes (i.e. 250 & 2500) and variable number of Resources, PEs, and Machines.

5.1 Average Consumed Time

The evaluation of Average consumed time is demonstrated on section IV with the similar set of resources and fluctuating the dataset set size. In this section two datasets are considered first the set of 250 tasks and 2500 tasks and the different set of resources are considered to measure the effect of resource on load balancing.

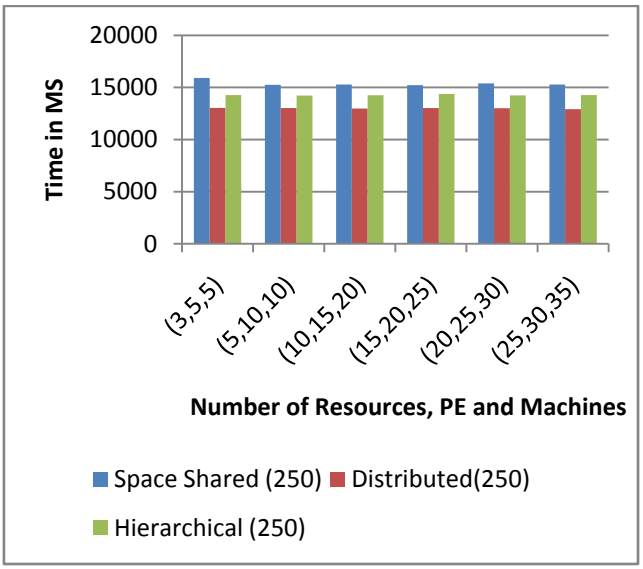

Figure 5.1(a) Average Consumed Time with 250 tasks

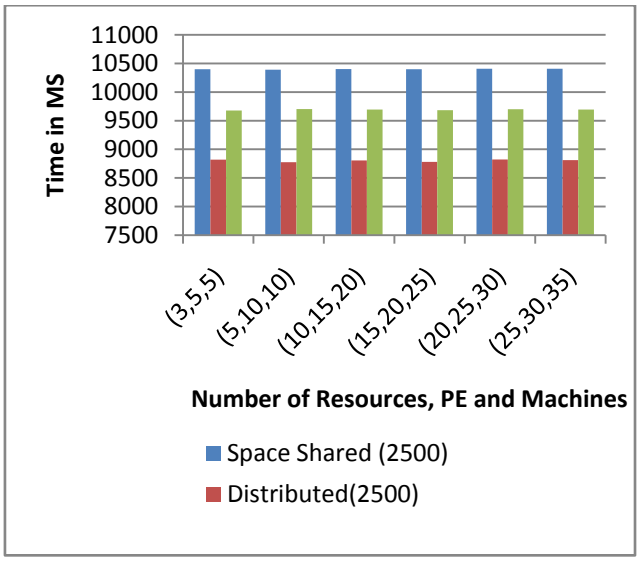

Figure 5.1(b) Average Consumed Time with 2500 tasks

The Figure 5.1(a) contains the Average Consumed Time for fixed length of tasks i.e. 250 and the Figure 5.1(b) contains for 2500 tasks. To demonstrate the Average Consumed Time, X axis contains the different server resources in terms of number of resources, number of machines and the processing elements and the Y axis contains the time required to process the tasks. According to the obtained results two conclusions are made; Firstly, the amount of task size reduces the amount of time of process execution. and Secondly, the large scale of resource configuration is not affecting much for executing the loads. The results of the experiments are shown in Table 5.1 under Appendix A.

5.2 Average Processing Cost

With the similar scenarios as described above for Average Consumed Time, the performance for Average processing cost is computed and the performance analysis is demonstrated in

Figure 5.2(a) and Figure 5.2(b) for 250 tasks and 2500 tasks respectively.

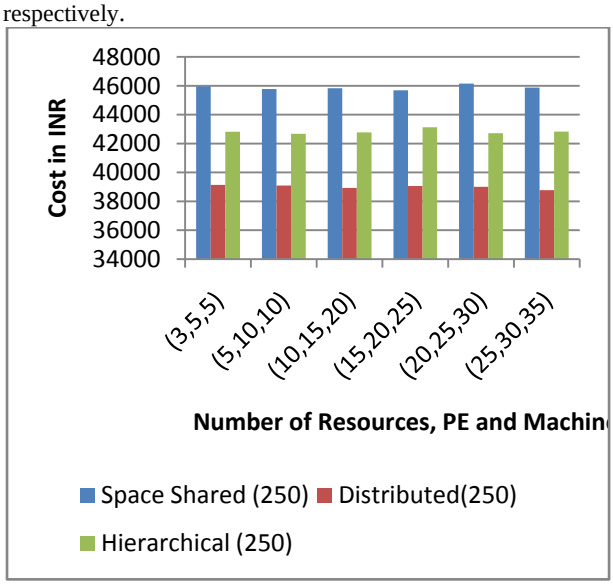

Figure 5.2(a) Average Processing Cost for 250 dataset

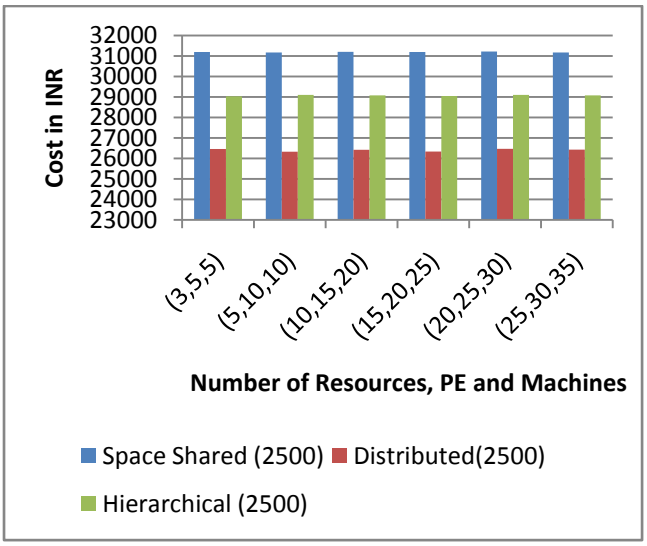

Figure 5.2(b) Average Processing Cost for 2500 tasks

In this context the X axis shows the grid configurations in terms of number of total resources, processing elements and the number of machines in each resource. According to the obtained results, as the number of tasks is increased with the similar resources, the processing cost is reduced and the grid performance is not much affected due to the increased size of dataset. The results of the experiments are listed in Table 5.2 under appendix A.

5.3 Average Waiting Time

This section analyzes the Average Waiting Time of a task for allocation of a resource after appearing in the process queue for the datasets of length 250 and 2500. The performance in both the scenarios is demonstrated in Figure 5.3(a) and Figure 5.3(b). The experiments are performed for finding the effect of increasing and decreasing the number of resources. Therefore the X Axis of the graph denotes the different combinations of resources and Y axis denotes the Average Waiting Time.

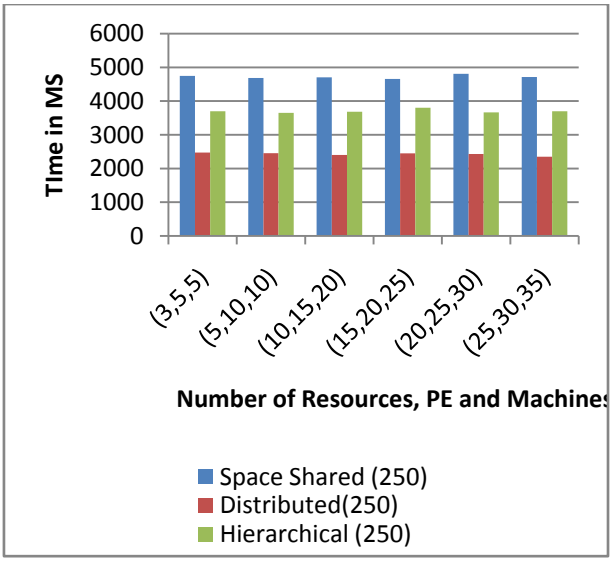

Figure 5.3(a) Average Waiting Time for 250 tasks

The performance graphs show that the Average Waiting Time is decreased when the larger datasets are executed on the similar configuration of resources in comparison to the smaller datasets. Further, on the front of Average Waiting Time, the distributed algorithm proves to be more efficient than the other two algorithms. The results are also shown in Table 5.3 under Appendix A.

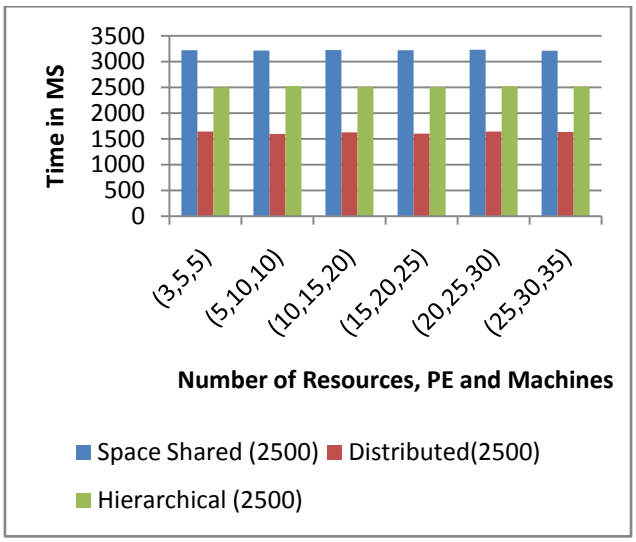

Figure 5.3(b) Average Waiting Time for 2500 tasks

5.4 Number of Task Migration

This section presents an analysis of the number of task migrations for the three algorithms. Figure 5.4(a) & Figure 5.4(b) presents the number of tasks migrated for the datasets comprising of 250 tasks and 2500 tasks respectively. The experimental results reveal that the length of dataset increases the number of task migrations.

Further, for smaller datasets the task migration increases with the increase of resources while for larger datasets, the task migration decreases with the increase of resources. Finally, in both the cases the Distributed load balancing algorithm demonstrates the higher task migration as compared to Hierarchical load balancing algorithm. The results are shown in the Table 5.4 under Appendix A.

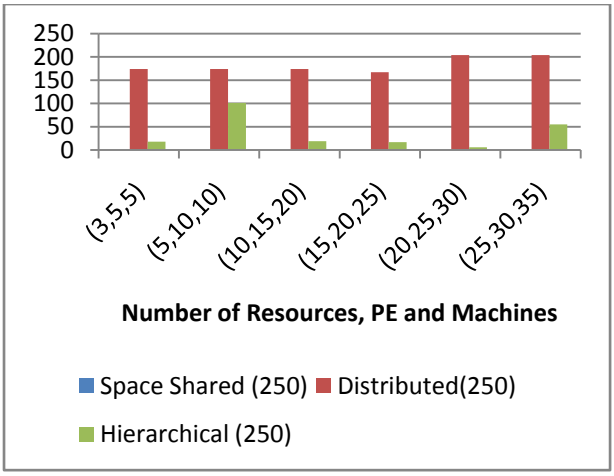

Figure 5.4(a) Number of Task Migration for 250 tasks

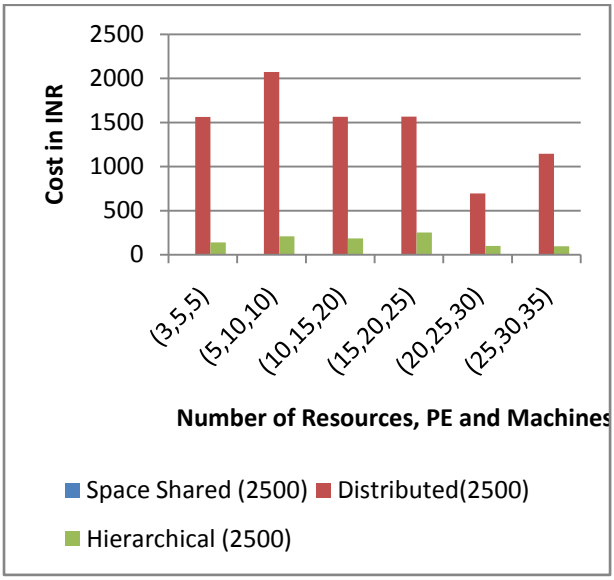

Figure 5.4(b) Number of Task Migration for 2500 tasks

6. CONCLUSION

The present work is dedicated to analyze and compare the three load balancing algorithms in computational grid environment viz. Space Shared, Distributed and Hierarchical in the following two perspectives:

First set of experiments are conducted to compute the performance parameters viz. Average Consumed Time, Average Processing Cost, Average Waiting Time and Number of Task Migration by increasing the length of datasets for fixed resource parameters. The results demonstrate that the values of all the aforesaid performance parameters are decreased with increase in length of datasets. In other words, with an increase in the number of tasks to be executed, the time and cost of resources is reduced.

In the next set of experiments the dataset size is fixed to 250 and the experiments are performed with the four aforesaid performance parameters with the varying set of resources. Thereafter, the similar experiments are conducted for the dataset size of 2500. These experiments help to analyze the effect of increasing resources for scheduling the tasks in a computational grid. Therefore, a set of different resources are prepared and the similar set of dataset is used for conducting experiments. The experiments demonstrate that the Average Consumed Time, Average Processing Cost, and Average Waiting Time are reduced with the increase in length of datasets and number of resources. On the other hand, the number of tasks migrated are increased with the increase in length of datasets and number of resources.

Finally, in terms of time and cost the distributed approach shows the efficient outcomes in comparison to other two approaches. On the other hand, in terms of the number of tasks migrated the hierarchical approach outperforms the distributed approach.
The future directions for research include further improvements in distributed and hierarchical algorithms for better results along with the study of fault tolerance of the proposed algorithms.

7. REFERENCES

[1] What is grid computing? Available online at: https://azure.microsoft.com/en-in/overview/what-is-grid-computing/

[2] B. Jacob, Brown, M., Fukui, K., & Trivedi, N. (2005), Introduction to grid computing. IBM redb, 2005

[3] Frederic Magoules, Kiat-An Tan and Abhinit Kuma, “Introduction to grid computing”, CRC press, 2009

[4] W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger, and K. Stockinger, Data Management in an International Data Grid Project, Proceedings of the first IEEE/ACM International Workshop on Grid Computing, India, 2000

[5] Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran, "A taxonomy and survey of grid resource management systems for distributed computing", Software: Practice and Experience 32, Number 2 (2002): pp. 135-164.

[6] Mark Baker, Rajkumar Buyya and Domenico Laforenza, “Grids and Grid technologies for wide-area distributed computing”, Software—Practice and Experience, 32, no. 15 (2002): 1437-1466.

[7] Karl Czajkowski, Ian Foster, Nick Karonis, and Steven Tuecke, "A resource management architecture for metacomputing systems", In Workshop on Job Scheduling Strategies for Parallel Processing, pp. 62-82. Springer, Berlin, Heidelberg, 1998

[8] Neeraj Mangla and Bhavya Bhatia, "Efficient Resource Management in Grid Computing", (2013)

[9] Buyya, Rajkumar, David Abramson, and Jonathan Giddy, "Grid Resource Management, Scheduling, and Computational Economy", In International Workshop on Global and Cluster Computing, Japan, Volume 21, pp. 2002-2040. 2000.

[10] Neeraj Pandey and Shashi Kant Verma, “Load Balancing Approaches in Grid Computing Environment”, International Journal of Computer Applications (IJCA), Volume 72– No.12, June 2013

[11] Belabbas Yagoubi and Yahya Slimani, “Dynamic Load Balancing Strategy for Grid Computing”, World Academy of Science, Engineering and Technology International Journal of Computer and Information Engineering Volume 2, Number 7, 2008

[12] Karim Kabalan Y., Waleed W. Smari, and Jacques Y. Hakimian, "Adaptive load sharing in heterogeneous systems: Policies, modifications, and simulation",

International Journal of Simulation, Systems, Science and Technology 3.1-2 (2002): pp. 89-100.

[13] Mohsen Amini Salehi, Hossein Deldari and Bahare Mokarram Dorri, "Balancing load in a computational grid applying adaptive, intelligent colonies of ants", Informatica 33.2 (2009)

[14] Hongzhang Shan, Leonid Oliker and Warren Smith, "Scheduling in heterogeneous grid environments: The effects of data migration", International Conference on Advanced Computing and Communication, Gujarat, India. 2004.

[15] Henrik Johansson and Johan Steensland, "A performance characterization of load balancing algorithms for parallel SAMR applications", Uppsala University, Department of Information Technology, Tech. Rep 47 (2006): 2006.

[16] D. Ramesh, and A. Krishnan, "Hybrid algorithm for optimal load sharing in grid computing", Journal of Computer Science, 2012.

[17] Preeti Gulia and Deepika Nee Miku, "Analysis and Review of Load Balancing in Grid Computing using Artificial Bee Colony", International Journal of Computer Applications (IJCA), Volume 71, Number 20, June 2013

[18] Suryadevera, Sowmya and Jaishri Chourasia, "Load balancing in computational grids using ant colony optimization algorithm", International Journal of Computer & Communication Technology 3.3 (2012): 20-23.

[19] Kumar, Prakash, Pradeep Kumar, and Vikas Kumar, "Computational Grid System Load Balancing Using an Efficient Scheduling Technique", International Journal of Computer Science and Network Security (IJCSNS) 15.8 (2015): 72.

[20] Volker Hamscher and Uwe Schwiegelshohn, "Evaluation of job-scheduling strategies for grid computing." Grid Computing—GRID pp. 191-202, 2000

[21] Abbas Karimi and Faraneh Zarafshan, "A New Fuzzy Approach for Dynamic Load Balancing Algorithm", (IJCSIS) International Journal of Computer Science and Information Security, Volume 6, Number 1, 2009.

[22] Yagoubi B., "Modele d'equilibrage de charge pour les grilles de calcul", Revue Africaine de la Recherche en Informatique et Mathematiques Appliquees: ARIMA, vol. 7, pages 1-19, 2007

[23] Yagoubi B., Meddeber M., "Distributed Load Balancing Model for Grid Computing", ARIMA, vol. 12, pages 43-60, 2010

[24] Parallel Workload Archive from http://www.cs.huji.ac.il/labs/parallel/workload/

8. APPENDIX

Appendix A (List of tables)

Table 5.1 Values for Average Consumed Time

Number of Resources, PE and Machines	Dataset size 250			Dataset size 2500		
	Space Shared	Distributed	Hierarchical	Space Shared	Distributed	Hierarchical
(3,5,5)	15919.48	13045.45	14270.72	10398.3	8820.48	9677.7
(5,10,10)	15257.94	13031.04	14222.69	10391.6	8775.53	9701.79
(10,15,20)	15278.21	12977.3	14256.26	10401	8806.48	9693.56
(15,20,25)	15229.34	13021.88	14376.82	10398.1	8779.93	9682.72
(20,25,30)	15387.97	13003.69	14238.24	10406.3	8822.25	9700.42
(25,30,35)	15290.36	12925.69	14274.18	10406.3	8809.9	9693.11

Table 5.2 Values for Average Processing Cost

Number of Resources, PE and Machines	Dataset size 250			Dataset size 2500		
	Space Shared	Distributed	Hierarchical	Space Shared	Distributed	Hierarchical
(3,5,5)	45958.44	39136.37	42812.18	31195	26461.4	29033.1
(5,10,10)	45773.82	39093.12	42668.01	31174.8	26326.6	29105.4

(10,15,20)	45834.63	38931.9	42768.79	31203	26419.5	29080.7
(15,20,25)	45688.03	39065.64	43130.46	31197.3	26339.8	29048.2
(20,25,30)	46154	39011.07	42714.73	31218.9	26466.8	29101.3
(25,30,35)	45871.09	38777.07	42822	31168.7	26429.7	29079.3

Table 5.3 Values for Average Waiting Time

Number of Resources, PE and Machines	**Dataset size 250**			**Dataset size 2500**		
	Space Shared	**Distributed**	**Hierarchical**	**Space Shared**	**Distributed**	**Hierarchical**
(3,5,5)	4744.85	2470.83	3696.1	3220.45	1642.59	2499.82
(5,10,10)	4683.31	2456.41	3648.04	3213.7	1597.65	2523.9
(10,15,20)	4703.58	2402.68	3681.63	3223.12	1628.6	2515.67
(15,20,25)	4654.72	2447.25	3802.19	3220.22	1602.05	2504.82
(20,25,30)	4810.35	2429.06	3663.62	3228.42	1644.37	2522.53
(25,30,35)	4715.73	2351.06	3699.56	3211.68	1632.02	2515.22

Table 5.4 Values for Number of Task Migration

Number of Resources, PE and Machines	**Dataset size 250**			**Dataset size 2500**		
	Space Shared	**Distributed**	**Hierarchical**	**Space Shared**	**Distributed**	**Hierarchical**
(3,5,5)	0	174	18	0	1564	140
(5,10,10)	0	174	101	0	2073	209
(10,15,20)	0	174	19	0	1565	185
(15,20,25)	0	167	17	0	1566	252
(20,25,30)	0	204	6	0	695	100
(25,30,35)	0	204	55	0	1146	97