Research Journal of Applied Sciences, Engineering and Technology 7(7): 1456-1463, 2014
DOI:10.19026/rjaset.7.419
ISSN: 2040-7459; e-ISSN: 2040-7467

Submitted: June 05, 2013 Accepted: June 21, 2013 Published: February 20, 2014

Research Article

Grid Scheduling with QoS Satisfaction and Clustering

[1]Devaki Palaniappan and [2]Valarmathi Muniappan Lakshapalam
[1]Department of CSE, Kumaraguru College of Technology, Coimbatore-641049, Tamil Nadu, India
[2]Department of CSE, Government College of Technology, Coimbatore-641013, Tamil Nadu, India

Abstract: The objective of the study is to device an Adaptive Machine Scoring Technique with Cluster (AMSTWC) to schedule the jobs/tasks in a grid environment which reduces the overall completion time (make span) and increases the resource Utilization. It also minimizes the execution time of the algorithm and with QoS satisfaction. The scheduling is done for computational as well as data grids. There are many heterogeneous Gridlets/machines which are geographically distributed. So, the searching time of the appropriate Gridlets, most suitable for the given job is more. This algorithm clusters the Gridlets depending on their configurations which reduces the search time of the Gridlets/machines which satisfies QoS. Task requirements are matched against the Machine capabilities available in Grid and AMSTWC selects the machine which has the highest resource score. AMSTWC result is compared with the existing algorithms in terms of make span, Resource Utilization, Flow Time and Execution time. AMSTWC performs better than the existing algorithms in most of the cases.

Keywords: Execution time, flow time, gridlets, machine scoring, make span, resource utilization

INTRODUCTION

The engineering and science problems in real world are complex and involves various complicated computation and transferring of big volume of data through the network. In order to solve these problems we need more powerful computers. Utilizing and combining the resources scattered around the world is a good approach. Hence, the concept of grid computing was proposed. Grid computing has emerged as the next generation distributed computing that aggregates dispersed heterogeneous resources under different administrative domain, for solving various kinds of computational and data intensive applications. Grid makes a virtual organization by grouping heterogeneous computers for specific problem solving. To complete the job scheduled in different machines, the underlying network plays a major role. So, we need high network bandwidth and reliable network connection.

Matching the resources for the user request and scheduling the job to the matched resource is an NP complete problem (Taura and Chien, 2000). Monitoring the progress of the job assigned is also difficult since the resources are across different administrative domains (Khateeb *et al*., 2009) and they are dynamic (Zomaya and Teh, 2001). Job scheduling and resource management in grid is a challenging job. Lots of heuristic algorithms adjust the scheduling strategies according to the nature of job (Kobra and Naghibzadeh, 2007). This study concentrates on QoS satisfaction which is done by getting task requirements like RAM, Budget, network Bandwidth, Operating System and deadline from the user and search the resource suitable for the user's task. QoS satisfaction is needed for the following reasons:

- Multimedia applications require resources with high network bandwidth and RAM to transfer bulk of data. No need to have high computing power.
- Problems involving partial differential equations to solve need computing power.
- Any scientific/engineering problems involving complex computations need more computing power.

Load balance is also an important issue in grid scheduling. The main purpose of load balance is to balance the load of each resource in order to enhance the resource utilization and increase the system throughput. Many load balancing algorithms have been proposed in grid environment (Cao *et al*., 2005; Suri and Singh, 2010), but they may not be suitable for change in system status. Based on this opportunity for improvement, a new scheduling algorithm is proposed to balance the load of a grid system with adaptive machine scoring while trying to minimize the make span and flow time of job execution. We assign a job to a resource depending on the resource's characteristics while simultaneously considering the load of the machine and execution time of the algorithm. Execution

Corresponding Author: Devaki Palaniappan, Department of CSE, Kumaraguru College of Technology, Coimbatore-641049, Tamil Nadu, India

time of the algorithm is reduced in searching of resources by clustering the machines with same configurations.

The objective of the proposed methodology is to minimize the overall completion time of the submitted tasks (make span) to the grid lets. It also maximizes the resource utilization for efficient usage of the available grid lets and searching of an appropriate resource (that satisfies the task requirements of OS, budget, Network Bandwidth and RAM) for the given job is minimized.

LITERATURE REVIEW

Different types of scheduling based on different criteria, such as static versus dynamic environment, multi-objectivity, adaptability, etc., are identified and heuristic and meta-heuristic methods for scheduling in Grids are proposed. The study reveals the complexity of the scheduling problem in Computational Grids when compared to scheduling in classical parallel and distributed systems and shows the usefulness of heuristic and meta-heuristic approaches for the design of efficient Grid schedulers. The requirements for modular Grid scheduling and its integration with Grid architecture is also proposed (Ajith and Fatos, 2010). Workflow scheduling is proposed. The problem of satisfying the QoS requirements of the user as well as minimizing the cost of workflow execution is proposed. On-demand resource provisioning, homogeneous networks and the pay-as-you-go pricing model is proposed. A two-phase algorithm which first distributes the overall deadline on the workflow tasks and then schedules each task based on its sub deadline is proposed (Saeid *et al*., 2013).

The study proposes resource scheduling in grid computing using a global optimization algorithm which is Bacterial foraging optimization. Main objective is to minimize make span and cost (Rajni, 2012). Bacterial Foraging optimization is used to schedule the resources in grid and it is used for the practical application of protein sequence analyzer. The study proposed Optimization (BFO) for finding similar protein sequences in the existing databases. Usage of BFO reduces the time taken by a resource to execute the user's requests and also the resources utilized are balanced (Vivekanandan and Ramyachitra, 2012).

In order to utilize the power of the grid completely, an Adaptive Scoring Job Scheduling algorithm (ASJS) is proposed. The main objective is to minimize the make span. The computational and data intensive applications were used for scheduling. ASJS selects the fittest resource to execute a job according to the status of resources. Local and global update rules are applied to get the newest status of each resource. Local update rule updates the status of the resource and cluster which are selected to execute the job after assigning the job and the Job Scheduler uses the newest information to assign the next job. Global update rule updates the status of each resource and cluster after a job is completed by a resource. It supplies the Job Scheduler with the newest information of all resources and clusters such that the Job Scheduler can select the fittest resource for the next job. However, the resource discovery tree is constructed for each attribute will take more time to schedule (Ruay-Shiung *et al*., 2012). A reliable scheduling algorithm is proposed to overcome the hardware failure, program failure and storage failure. A hierarchical-driven scheduling is proposed (Xiaoyong *et al*., 2012).

A Fault tolerant hybrid load balancing strategy which takes into account grid architecture, computer heterogeneity, communication delay, network bandwidth, resource availability, resource unpredictability and job characteristics is proposed. Objective is to arrive at job assignments that could achieve minimum response time and optimal computing node utilization (Jasma and Nedunchezhian, 2012). The study focuses on computing grid. The system load is taken as a parameter in determining a balance threshold and the scheduler adapts the balance threshold dynamically when the system's load changes. First, the scheduling algorithm balances the system load with an adaptive threshold and second, it minimizes the make span of jobs (Yun-Han *et al*., 2011). A new Priority based Job Scheduling algorithm (PJSC) in cloud computing is proposed using multiple criteria decision making model (Shamsollah and Mohamed, 2012).

Architectural diagram: Users submit their jobs to grid portal. Task requirement block collects the requirements and gives them to Resource score calculator. Resource Score calculator gets Resource capability information from Grid Information Service (GIS) through Gird Broker and it calculates the resource score for all tasks of all resources. Then, Resource Score is passed to Grid job scheduler through Grid Broker to schedule. Finally, Grid broker assigns the task to the resources and execution is carried out in the resources. After completing the task, resource manager reports the result to the requested users (Fig. 1).

PROBLEM DEFINITION AND PROPOSED METHODOLOGY

The problem is to minimize the overall completion time as well as increase the resource utilization while allocating m resources to do n tasks where (n>m). Allocation is done in an offline manner. To formulate the problem, define Ti where i = {1, 2, 3,… n} as n independent tasks permutation and Rj where j = {1, 2, 3, … m} as m computing resources. Suppose that the Expected Time to Complete (ETCi, j) (Braun *et al*., 1999) is the processing for task i when computing

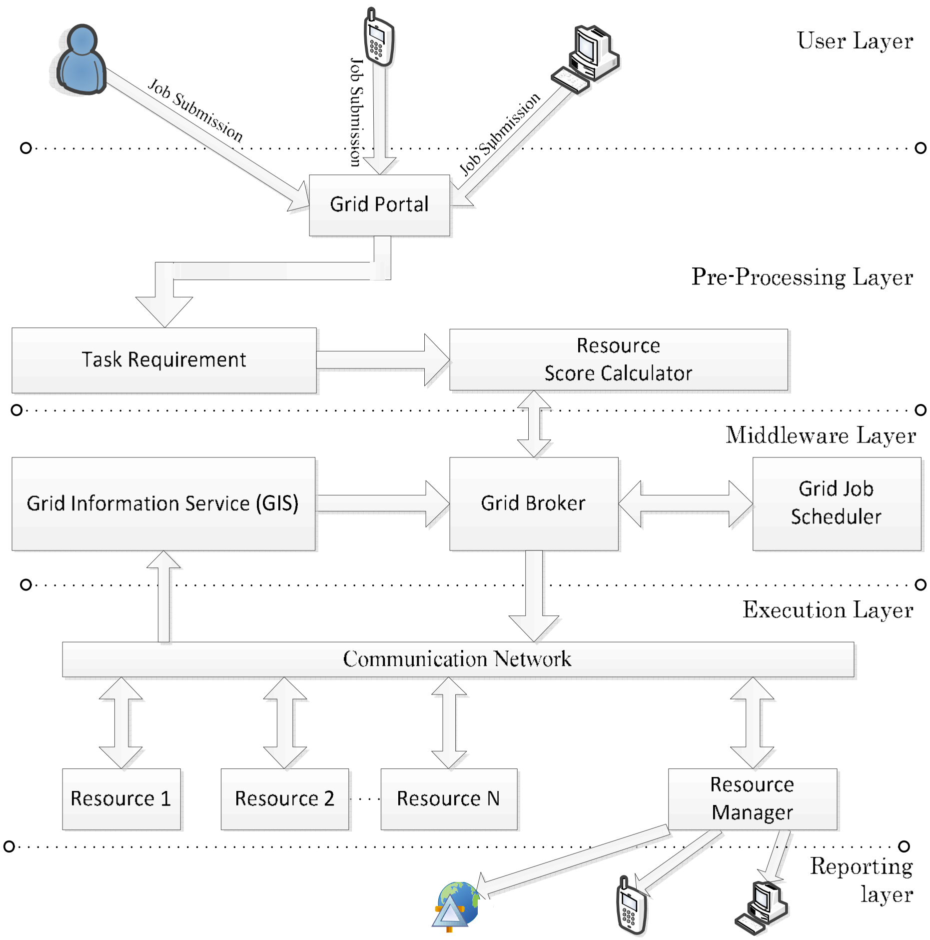

Fig. 1: Architectural diagram

time on resource j is known. The completion time C (x) represents the total time of completion of all n tasks. The objective is to minimize C (X) in Eq. (1):

$$C(x)_{\min} = \sum_{i=1}^{n} \sum_{j=1}^{m} ETC\,[i, j] \tag{1}$$

The minimal C (x) represents the length of schedule of whole tasks working on available resources.

Methodology: The resource is selected for a task using resource score. For each task the resource score is calculated as in Eq. (2):

$$\begin{aligned} \mathrm{Re}\,sourceScore_{i,j} &= \alpha.\Big(makespanScore_{i,j} \\ &+ (1-\alpha) NBScore_{j} + QoSScore_{i,j} \\ &+ LBScore_{j} + \frac{1}{\mathrm{Re}\,sourceaval_{j}} \end{aligned} \tag{2}$$

If the application is computationally intensive, then α = 1 else it is data intensive for which α = 0. The data intensive application can be found using the task requirement Network Bandwidth parameter. If the required network Bandwidth parameter is above the threshold, then the application is data intensive. Threshold is taken as 70% of the maximum bandwidth requested by the task. If the task network bandwidth request is above 70% of the maximum task request network bandwidth then α is 0 else α is 1. So, α decides whether the application is computationally intensive or data intensive.

Resource score is depending on make span Eq. (3). -if it is computational grid), network bandwidth satisfaction Eq. (4) -if it is data grid) QoS satisfaction Score Eq. (5) -QoS Score), Load Balance Score Eq. (6) LB Score) and Resource availability:

$$makespanScore_{i,j} = \frac{1}{etc_{i,j}} \tag{3}$$

Make span score is high if the expected time to complete is low, in turn the resource score is high:

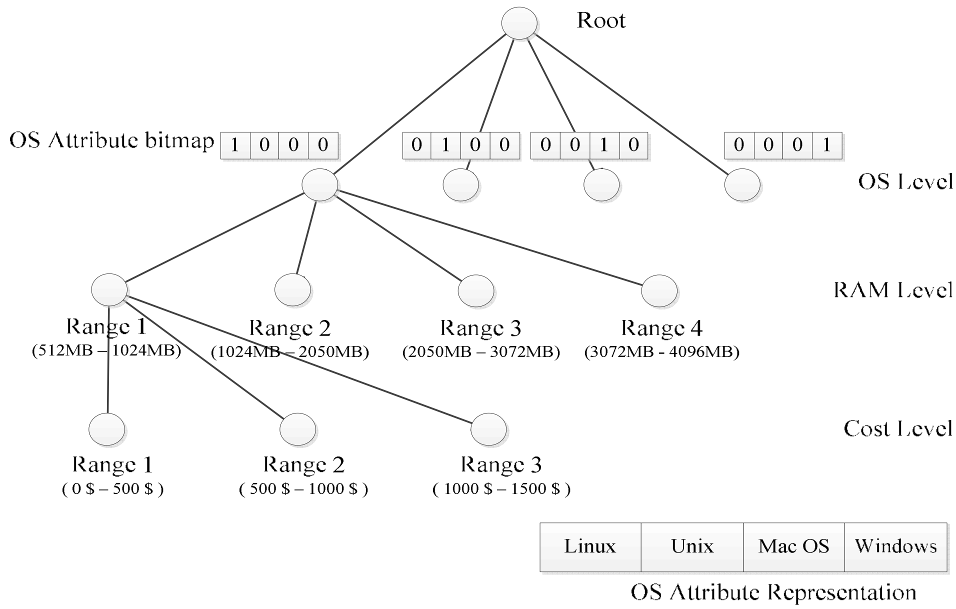

Fig. 2: Example of resource discovery tree

$$NBScore_j = RcapNetworkbandwidth_j \tag{4}$$

Network Bandwidth Score is purely depending on the Resource capability bandwidth score:

$$QoSScore_{i,j} = \frac{1}{RcapRAM_j - TReqRAM_i} + \beta.(TReqDL_i - RcapDL_j) + (1-\beta).\frac{1}{Rcapcost_j} \tag{5}$$

β is the weighted QoS and if β is high, then task is keen on deadline whereas if β is low, then the task is keen on Budget. QoS Score is depending on the RAM, Deadline and Budget requirements of the task satisfaction. If the resource capability of RAM is equal to Task Requirement of RAM, then the term $RcapRAM_j - TReqRAM_i$ in the above equation is 0. This can be avoided by replacing with the term:

$$\frac{1}{RcapRAM_j - TReqRAM}$$

with the value 1 which is the highest possible value:

$$LBScore_j = LBfactor_j \tag{6}$$

LB $Score_j$ is depending on the Load Balancing parameter of the $Resource_j$. The load balancing parameter of each resource is initially set to some value (may be 100). If any one $Task_i$ is assigned to the $Resource_j$, then the load balancing factor value of the $Resource_j$ is decreased such that in next selection the Resource Score inturn is reduced. For each task the resource with the highest resource score is selected. The execution time of the algorithm is reduced because the hierarchical tree is constructed by clustering the resources as groups for the resource known as resource discovery tree as in Fig. 2 for the attribute Operating system. Algorithm which is using this tree is known as AMST with Clustering (AMSTWC). AMSTWOC algorithm is not using this clustering tree for searching resources.

Bitmap representation for resource discovery tree: Let $\{R_1, R_{2\ldots} R_n\}$ be the set of resources and $\{A_1, A_2, \ldots A_n\}$ be the set of attributes of the resources (Ruay-Shiung and Min-Shuo, 2010). In each level of the tree, one attribute is checked for the task requirement and the remaining tree is pruned from checking for further attributes in searching of resources method. The Bitmap Data structure used for the tree is in Fig. 2.

In searching process for QoS satisfaction, if the task requirement is Unix OS then, the sub tree 2 and sub tree 3 searching is pruned in such a way that the searching time is reduced. This inturn reduces the algorithm's execution time.

Proposed algorithm

AMSTWC algorithm:

Step 1: Generate ETC matrix

Step 2: Get the task requirement matrix and resource capability matrix from the Gird Information Service

Step 3: Construct Resource capability tree

Step 4: For each $task_i$ do

Find whether the $task_i$ is data intensive or computational intensive
Calculate resource score of all resources for $task_i$
Select the $resource_j$ which has the highest resource score value
Assign the task to the selected resources.
Update the resource load
Until all tasks are assigned

Step 5: Calculate make span, flow time and resource utilization and record the algorithm execution time
Step 6: End

EXPERIMENTAL RESULTS

Performance metrics:

- **Make span:** Overall completion time which is defined in Eq. (1)
- **Resource utilization:** Resource utilization defined as the degree of utilization of resources with respect to the schedule. It is defined as follows:

$$\text{Re}\,sourceUtil\;ization = \frac{\sum_{i \in machines} Complete\;[i]}{makespan\;.m}$$

where Complete [i] is the completion time of last job on machine i and m is the number of machines. Objective is to maximize the resource utilization for all possible schedules
- **Flow time:** Flow-time is the sum of the finishing times of jobs. Objective is to minimize the flow time. It is defined as:

$$F = \sum C_j, j = 1,..., N$$

- **Algorithm execution time:** Objective is to minimize the execution time of the proposed algorithm

Benchmark description: The benchmark by Braun *et al*. (1999) is a frequently used benchmark that is very effective in simulating grid systems and capturing most important characteristics of the job scheduling problem. In it, instances are classified according to three parameters (job heterogeneity, machine heterogeneity and consistency) into 12 different types of ETC matrices, each of them consisting of 100 instances. All instances are composed from 512 jobs and 16 machines. They are labeled as u x yyzz where u means uniform distribution (in the matrix generation), x is the type of consistency (c-consistent, i-inconsistent and p means partially consistent), yy and zz indicate the job and machine heterogeneity (hi-high and lo-low). An ETC matrix is consistent when a machine is faster than others for all the jobs. Inconsistency means that a machine is faster for some jobs and slower for some others, while it is semi-consistent if it contains a consistent sub-matrix. The values are taken as an average for 100 runs for α = 1 (Computational Grid).

Table 1: Make span comparison

Instance	FCFS	AMSTWOC	AMSTWC
U C HIHI	1259.9880	1352.4290	1079.1540
U C HILO	675.5530	859.8783	790.0265
U C LOHI	663.8204	821.4559	743.5311
U C LOLO	322.5635	415.9882	383.7268
U I HIHI	2509.4260	2470.5580	1048.6060
U I HILO	1288.3190	1387.8020	625.7496
U I LOHI	1284.2090	1353.1150	610.6889
U I LOLO	634.4175	685.9567	317.0915
U P HIHI	1262.6720	1354.9960	1083.2330
U P HILO	674.8792	867.3299	787.2625
U P LOHI	665.8572	830.7629	745.0480
U P LOLO	323.4126	422.9728	386.1405

Table 2: Resource utilization comparison

Instance	FCFS	AMSTWOC	AMSTWC
U C HIHI	2.3577	5.8446	5.7578
U C HILO	2.2808	5.1903	4.8409
U C LOHI	2.3461	5.2623	5.0070
U C LOLO	2.3254	5.2711	4.9162
U I HIHI	2.5384	4.3202	6.3162
U I HILO	2.5332	4.0423	5.9551
U I LOHI	2.5904	4.0125	5.9949
U I LOLO	2.5985	3.9603	5.8555
U P HIHI	2.3194	5.9535	5.7237
U P HILO	2.2830	5.1039	4.8196
U P LOHI	2.8638	5.2171	4.9849
U P LOLO	2.3583	5.1521	4.8513

Table 3: Flow time comparison

Instance	FCFS	AMSTWOC	AMSTWC
U C HIHI	2642.5188	8010.8015	6081.4181
U C HILO	1419.9860	4206.9229	3627.9109
U C LOHI	1417.9228	4088.7806	3557.5905
U C LOLO	695.6284	2074.2671	1801.2943
U I HIHI	5974.4953	8983.8562	6407.8659
U I HILO	3101.3846	4698.0150	3651.0477
U I LOHI	3102.4057	4580.1369	3591.7916
U I LOLO	1550.2728	2308.4572	1818.1428
U P HIHI	2648.4907	7956.8925	6080.2258
U P HILO	1419.4616	4176.0677	3614.8729
U P LOHI	1554.1022	4066.2471	3553.0836
U P LOLO	697.0421	2054.3731	1795.7396

Table 4: Execution time comparison

Instance	FCFS	AMSTWOC	AMSTWC
U C HIHI	0.0320	14.5800	0.0052
U C HILO	0.0640	14.3120	0.6100
U C LOHI	0.0940	15.0020	0.6220
U C LOLO	0.0000	14.7700	0.7540
U I HIHI	0.0640	14.5140	0.4640
U I HILO	0.0940	15.2000	0.5300
U I LOHI	0.1900	14.2060	0.7220
U I LOLO	0.0640	15.0600	0.6040
U P HIHI	0.1280	14.0880	0.9080
U P HILO	0.0920	14.7720	0.5980
U P LOHI	0.2840	14.5000	0.8660
U P LOLO	0.0620	14.5940	0.5340

In Table 1, for the computational grids out of 12 combinations, our algorithm performs better for 6 combinations (Bold) whereas for inconsistent case, our algorithm performs better for all combinations of heterogeneity. Overall for 50% of the combinations our algorithm performs better. In Table 2, the resource utilization is better for all inconsistent cases. In Table 3, our algorithm performs poorly in all combinations of

Table 5: Make span comparison

	α = 0.25		α = 0.50		α = 0.75	
Instance	AMSTWOC	AMSTWC	AMSTWOC	AMSTWC	AMSTWOC	AMSTWC
U C HIHI	1426.30	1087.5539	1416.8047	1103.4926	1404.5912	1102.4630
U C HILO	846.20	773.5084	842.0430	770.8088	847.0673	775.0846
U C LOHI	852.00	779.8318	844.4138	775.2478	810.8959	739.6598
U C LOLO	432.50	395.5654	495.4829	454.8466	430.6891	398.6547
U I HIHI	2345.40	1072.8944	2290.7594	1033.6105	2502.9849	1062.7280
U I HILO	1210.10	609.9697	1238.5719	609.2914	1331.0644	628.8751
U I LOHI	1227.50	615.3519	1076.4004	618.5126	1326.2185	609.7131
U I LOLO	631.20	320.9927	707.8205	358.5829	650.5396	313.5738
U P HIHI	1425.70	1086.7048	1419.6963	1102.0741	1413.0685	1106.5907
U P HILO	852.50	772.9580	848.3231	766.5129	854.5760	772.3494
U P LOHI	860.30	777.9089	850.9616	774.3339	822.0156	740.1947
U P LOLO	435.36	395.1676	500.9713	455.5923	437.4567	399.6840

Table 6: Flow time comparison

	α = 0.25		α = 0.50		α = 0.75	
Instance	AMSTWOC	AMSTWC	AMSTWOC	AMSTWC	AMSTWOC	AMSTWC
U C HIHI	8178.9079	6211.47	8477.8669	6243.4972	7909.0243	6240.1572
U C HILO	4416.1781	3623.26	4230.8560	3607.6844	4161.2473	3597.1623
U C LOHI	4309.5603	3589.69	4238.9790	3627.6506	4043.1527	3545.9674
U C LOLO	2106.0420	1816.41	2405.2613	2074.3583	2125.7148	1820.8035
U I HIHI	8914.9038	6586.51	9209.1953	6537.7735	8869.0135	6628.4887
U I HILO	4721.3804	3633.80	4546.6512	3599.9739	4575.3259	3633.4899
U I LOHI	4630.4137	3610.82	4619.4274	3630.0330	4347.7126	3616.5828
U I LOLO	2282.9987	1840.68	2594.1742	2087.5657	2346.0925	1827.9746
U P HIHI	8138.0433	6205.57	8442.2686	6242.0354	7870.5897	6250.7389
U P HILO	4390.2415	3620.69	4204.9147	3602.7117	4137.5984	3585.5128
U P LOHI	4280.1268	3578.07	4218.3630	3624.3847	4021.4313	3537.7333
U P LOLO	2092.3840	1813.59	2394.9079	2073.5457	2115.1261	1819.1749

Table 7: Resource utilization comparison

	α = 0.25		α = 0.50		α = 0.75	
Instance	AMSTWOC	AMSTWC	AMSTWOC	AMSTWC	AMSTWOC	AMSTWC
U C HIHI	5.5641	5.8384	5.7860	5.7568	5.5222	5.7819
U C HILO	5.3548	4.8695	5.2582	4.9447	5.1632	4.8967
U C LOHI	5.2547	4.8410	5.1975	4.9125	5.1499	4.9881
U C LOLO	5.0609	4.7852	5.0899	4.7908	5.1227	4.8520
U I HIHI	4.3839	6.4185	4.5152	6.5074	4.1879	6.4807
U I HILO	4.3904	6.0945	4.2526	6.0408	3.9706	5.9364
U I LOHI	4.2249	6.0011	4.1985	5.9944	3.9050	6.0242
U I LOLO	4.0513	5.8310	4.1368	5.9640	4.1100	6.0072
U P HIHI	5.7045	5.8175	5.9054	5.7554	5.6235	5.7703
U P HILO	5.3033	4.8749	5.2101	4.9590	5.1435	4.9324
U P LOHI	5.2058	4.8545	5.1626	4.9193	5.0767	4.9685
U P LOLO	5.0279	4.7891	5.0283	4.8039	5.0301	4.8405

Table 8: Execution time comparison

	α = 0.25		α = 0.50		α = 0.75	
Instance	AMSTWOC	AMSTWC	AMSTWOC	AMSTWC	AMSTWOC	AMSTWC
U C HIHI	15.2740	0.0087	14.5400	0.0068	14.4500	0.0065
U C HILO	14.4920	0.6280	14.5620	0.4980	15.0580	0.4340
U C LOHI	14.9200	0.4700	14.7640	0.7140	14.7960	0.5600
U C LOLO	14.7720	0.9340	14.1220	0.5320	14.7940	0.4380
U I HIHI	14.5520	0.3160	14.5920	0.7800	14.7760	0.5600
U I HILO	14.6340	0.9120	14.1900	0.7080	15.4440	0.5280
U I LOHI	14.9680	0.6520	14.7180	0.8360	15.1540	0.5000
U I LOLO	14.6240	0.7140	15.2300	0.5280	14.8820	0.4960
U P HIHI	14.4420	0.6220	15.0060	0.6520	15.0920	0.6580
U P HILO	14.8940	0.6620	15.1220	0.5680	15.4920	0.4720
U P LOHI	15.0080	0.6780	14.7920	0.6760	14.7960	0.5980
U P LOLO	14.9840	0.6900	14.3100	0.7240	15.0840	0.4980

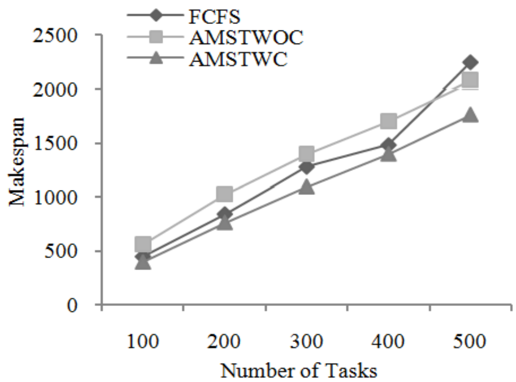

Fig. 3: Make span comparison for U C HIHI

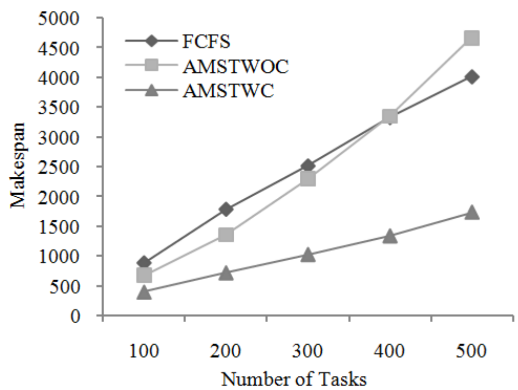

Fig. 4: Make span comparison for U I HIHI

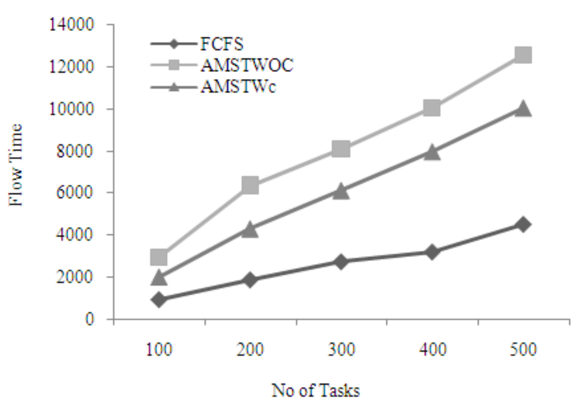

Fig. 5: Flow time comparison for U C HIHI

heterogeneity for flow time comparison. This is because the selection of resources is based on the task requirements. In Table 4, FCFS performs better in execution time since the machines are not checked for the requirement satisfaction. However, our algorithm performs 100% better than AMSTWOC.

For different values of α, for example 0.25, 0.50 and 0.75 (i.e., 75, 50 and 25% of data grids in the task request) the comparison is given in Table 5 to 8.

In Table 5 to 8, our algorithm performs 100% better than AMSTWOC in all the metrics for all

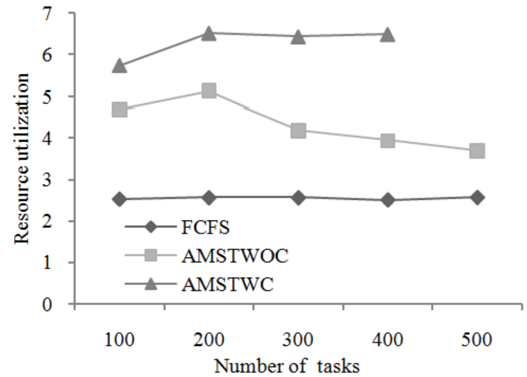

Fig. 6: Resource utilization comparison for U I HIHI

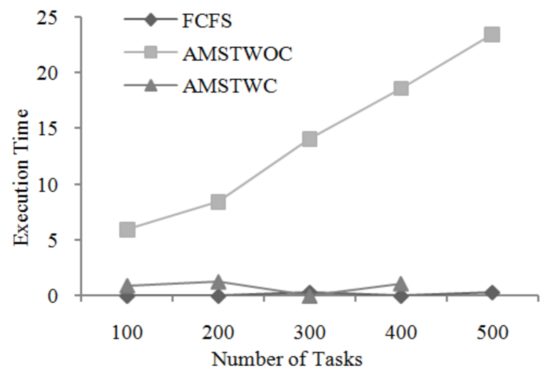

Fig. 7: Execution time comparison for U P HIHI

combinations of heterogeneity for various combinations of (75, 50 and 25%, respectively) data grids. Figure 3 and 4 show the make span comparison between the existing and the proposed algorithm (AMSTWC) for high task and high machine heterogeneity-consistent and inconsistent combinations. For both the combinations, as the number of tasks increases, our proposed algorithm gives reduced make span time. Compared to consistent combination, the inconsistent combination gives better performance in terms of make span. Figure 5 shows the flow time performance for high task and high machine heterogeneity-consistent combination comparison, our algorithm yields better results compared to FCFS. Figure 6 shows high task and high machine heterogeneity-inconsistent combination comparison; our algorithm has better resource utilization. Figure 7 shows the algorithm execution time comparison for high task and high machine heterogeneity-partial consistent combination, our method gives more or less the same performance as that of FCFS even though our algorithm has the matching process time of an appropriate resource for the task submitted.
Sample graphs are given in Fig. 3 to 7.

CONCLUSION AND RECOMMENDATIONS

The focus of our study is on QoS satisfied task scheduling with load balancing of resources. The experimental results show that the proposed algorithm is performing well in terms of execution time, resource utilization and make span. Our algorithm is poor in flow time because the selected resources are QoS satisfied resources. So, the jobs are waiting until it gets the QoS satisfied resource.

In future, we will adjust the score and add more parameters for the realistic environments. In real environments, because of more dynamic nature of grid, many more factors like reliability have impact on make span of the scheduling process. Reliability model may be included in future.

REFERENCES

Ajith, A. and X. Fatos, 2010. Computational models and heuristic methods for Grid scheduling problems. Future Gener. Comput. Syst., 26: 608-621.

Braun, T., H.J. Siegal, N. Beck and L.L. Boloni, 1999. A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems. Proceeding of the 8th IEEE Heterogeneous Computing Workshop (HCW'99), pp: 15-29.

Cao, J., D.P. Spooner, S.A. Jarvis and G.R. Nudd, 2005. Grid load balancing using intelligent agents. Future Gener. Comput. Syst., 21: 135-149.

Jasma, B. and R. Nedunchezhian, 2012. A hybrid policy for fault tolerant load balancing in grid computing environments. J. Netw. Comput. Appl., 35: 412-422.

Khateeb, A.A., R. Abdullah and A.N. Rashid, 2009. Job type approach for deciding job scheduling in Grid computing systems. J. Comput. Sci., 5(10): 745-750.

Kobra, E. and M. Naghibzadeh, 2007. A min-min max-min selective algorithm for grid task scheduling. Proceeding of the 3rd IEEE/IFIP International Conference in Central Asia (ICI 2007).

Rajni, I.C., 2012. Bacterial foraging based hyper-heuristic for resource scheduling in grid computing. Future Gener. Comput. Syst., 29(3): 751-762.

Ruay-Shiung, C. and H. Min-Shuo, 2010.A resource discovery tree using bitmap for grids. Future Gener. Comput. Syst., 26(1): 29-37.

Ruay-Shiung, C., L. Chih-Yuan and L. Chun-Fu, 2012. An adaptive scoring job scheduling algorithm for grid computing. Informat. Sci., 207: 79-89.

Saeid, A., N. Mahmoud and H.J.E. Dick, 2013. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. Future Gener. Comput. Syst., 29: 158-69.

Shamsollah, G. and O. Mohamed, 2012. A priority based job scheduling algorithm in cloud computing. Proc. Eng., 50: 778-785.

Suri, P.K. and M. Singh, 2010. An efficient decentralized load balancing algorithm for grid. Proceeding of the 2010 IEEE 2nd International Advance Computing Conference (IACC), pp: 10-13.

Taura, K. and A. Chien, 2000. A heuristic algorithm for mapping communicating tasks on heterogeneous resources. Proceeding of the 9th Heterogeneous Computing Workshop, Cancun Mexico, pp: 102-118.

Vivekanandan, K. and D. Ramyachitra, 2012. Bacteria foraging optimization for protein sequence analysis on the grid. Future Gener. Comput. Syst., 28(4): 647-656.

Xiaoyong, T., L. Kenli, Q. Meikang and H.M.S. Edwin, 2012. A hierarchical reliability-driven scheduling algorithm in grid systems. J. Parallel Distrib. Comput., 72: 525-535.

Yun-Han, L., L. Seiven and C. Ruay-Shiung, 2011. Improving job scheduling algorithms in a grid environment. Future Gener. Comput. Syst., 27(8): 991-998.

Zomaya, A.Y. and Y.H. Teh, 2001. Observations on using genetic algorithmsfor dynamic load-balancing. IEEE T. Parallel Distrib. Syst., 12(9): 899-912.