ISBN 978-93-5156-328-0
International Conference of Advance Research and Innovation (ICARI-2014)

Time and Cost Effective Task Scheduling In Grid Environment

Minakshi Saini *, Prashant Kumar
Department of Computer Science Engineering, Shri Venkateshwara University, Gajraula, U.P, India

Article Info

Article history:
Received 29 December 2013
Received in revised form
10 January 2014
Accepted 20 January 2014
Available online 1 February 2014

Keywords

Scheduling Techniques,
Grid Computing Environment,
Throughput,
Time Effective,
Cost Effective

Abstract

This paper titled **"Time and Cost Effective Task Scheduling In Grid Environment"** presents the time and cost effective scheduling technique followed by the scheduler determines the Grid system throughput and utilization of the resources in to the grid. Today's parallel and distributed systems are changing in the organization and the concept of Grid computing, a set of dynamic and heterogeneous resources connected via Internet and shared by many and different users, is nowadays becoming a reality.

The Grid system is responsible for the execution of jobs submitted to it. The advanced Grid system will include a task scheduler which automatically finds the most appropriate machines on which a given job is to run. This resource selection is very important in reducing the total execution time and cost of executing the tasks which depends on the task scheduling algorithm.

1. Introduction

Grid computing could be defined as any of a variety of levels of virtualization along a continuum. Exactly where along that continuum one might say that a particular solution is an implementation of grid computing versus a relatively simple implementation using virtual resources is a matter of opinion. But even at the simplest levels of virtualization, one could say that grid-enabling technologies are being utilized. The concept of a computational grid was first proposed by Ian Foster and Carl Kesselman in the mid 1990s. Since then the field has involved into one of the most exciting areas of study in the computing fraternity. Grid is a novel infrastructure for network computing on local or geographical scales that can dynamically represent heterogeneous computing resources.

Grid computing is broadly used in many scientific and engineering application fields. Grid application addresses collaboration, data sharing, cycle sharing and other modes of interaction that involves distributed resources and services.

The concept of grid computing is getting popular day by day with the emergence of the internet as a ubiquitous media and the wide spread availability of powerful computers and network as low cost commodity component. Resources could be computational system (such as traditional computers, clusters, or even powerful desktop machines), special class of devices (such as sensors) and even storage devices. A number of application need more computing power than can be offered by a single resource in order to solve them within a feasible reasonable time and cost.

Geographically distributed resources need to be logically coupled together to make them work as a unified resources. This led to the popularization of a field called Grid Computing. The field of Grid Computing is a manifestation of the development of distributed and cluster computing environments. Grid computing, most simply stated, is distributed computing taken to next evolutionary level. The goal is to create the illusion of a simple yet large and powerful self managing virtual computer out of a large collection of connected heterogeneous system sharing various combinations of resources. The standardization of communication between heterogeneous systems created the Internet explosion. The emerging standardization for sharing resources along with the availability of higher bandwidth, are driving a possibly equally large evolutionary step in grid computing.

2. Scheduling Techniques

2.1 Non Duplication Technique

*** Corresponding Author,**
E-mail address: mail2minakshi.7886@gmail.com

Two common approaches in non-duplication based scheduling are list scheduling and cluster-based scheduling.

List scheduling is one of most commonly used scheduling algorithms. In list scheduling, a weight is assigned to each task and edge, based on which an ordered task list is constructed by assigning priority for each task. Then, tasks are selected in the order of their priorities, and each selected task is scheduled to a computing host that can minimize a predefined cost function.

2.2 Duplication Technique

The proposed algorithm is a duplication-based static scheduling algorithm and it differs from the previous algorithms by addressing the minimization of the schedule length and the number of processors used as separate problems to be optimized in two distinct phases.

Real world problem in grid are multi objective means they required more than one objective. For e.g. Total execution time or make span, economy, reliability, trustworthiness, etc. We proposed a scheduling algorithm based on multi objective namely total economy cost/execution time.

3. Review Of Lierature

As two typical list scheduling heuristics, HEFT (Heterogeneous Earliest Finish Time) and CPOP (Critical Path on a Processor) are studied in [3]. The upward rank and downward rank of each task are computed at the beginning. HEFT algorithm always selects the task with the highest upward rank at each step. Then the selected task is assigned to a host that can minimize its earliest finish time. In contrast, CPOP algorithm always selects the task with the highest total rank (upward rank + downward rank) value. In order to minimize the total execution time, CPOP schedules all critical tasks onto a single host with the best performance. During execution, if a selected task is non-critical, it will be mapped to a host which could minimize its earliest finish time, as in HEFT. Both HEFT and CPOP have low complexity, i.e., lower algorithm execution time. However, the study in [4] observed that the performances of these two algorithms are affected dramatically by how to assign weights to the nodes and edges. In some extreme cases, different weight assignment approaches can lead up to 47.2% of performance difference. In another popular scheduling heuristic group scheduling, tasks are sorted into groups, under the constraint that tasks in the same group should be independent. Tasks then are scheduled group by group. The studies in [5] proposed a hybrid remapping heuristic. Tasks in a DAG are partitioned into levels so that there is no dependency among tasks at the same level. Then, tasks are mapped to computing hosts with task/host pairs using a static algorithm (e.g., baseline). The merit of this hybrid heuristic is to revise the static mapping result during job execution by two runtime factors, the availability of computing hosts and the completion time of tasks in previous levels. However, the task partition merely considers the task dependency relationships. It does not take task priority into account, which may result in that some tasks with lower priority are sorted into improper groups (levels). Clustering algorithms are proposed for the case of an unbounded number of computing resources, so they are not suitable for grid environment. Computational Grids computing systems are emerging as a new paradigm for solving large-scale problems in science, engineering and commerce [6, 7]. They enable the creation of virtual enterprises (VEs) for sharing and aggregation of millions of resources (e.g. SETI@Home [8]) geographically distributed across organizations and administrative domains. They comprise heterogeneous resources (PCs, workstations, clusters and supercomputers), fabric management systems (single system image OS, queuing systems, etc.) and policies, and applications (scientific, engineering and commercial) with varied requirements (CPU, I/O, memory and/or network intensive).

In [7, 9–11], we proposed and explored the usage of an economics-based paradigm for managing resource allocation in Grid computing environments. The economic approach provided a fair basis in successfully managing decentralization and heterogeneity that is present in human economies Competitive economic models provide algorithms/policies and tools for resource sharing or allocation in Grid systems. The models can be based on bartering or prices. In the bartering-based model, all participants need to own resources and trade resources by exchanges (e.g., storage space for CPU time). In the price-based model, the resources have a price, based on the demand, supply, value and the wealth in the economic system.

Most of the related work in Grid computing dedicated to resource management and scheduling problems adopt a conventional style where a scheduling component decides which jobs are to be executed at which site based on certain cost functions (Legion [12] (Legion [12], Condor [13], AppLeS [14], Netsolve [15]). Such cost functions are often driven by system-centric parameters that enhance system throughput and utilization rather than improving the utility of application processing.

4. Proposed Work

The main objective of this paper to develop a time and cost effective task scheduling technique for grid computing. Nodes in grid computing environment can be represented in form of Direct Acyclic Graph (DAG). This scheduling algorithm is proposed to optimize the time taken and economic cost of the schedule and minimizes the requirements of processors. The proposed algorithm is to be implemented to schedule different random DAGs onto different grids of heterogeneous clusters of various sizes.

Following are the objectives of the proposed work**:**

- We implemented a task scheduling method for grid computing environment to minimize the execution time of task. It is a scheduling method mainly based on DAG's workflow model.
- We present the high-level abstract language we developed for grid workflow application description, the Grid Application Modeling and Description Language (GAMDL).
- The algorithm has been implemented in JAVA for evaluation of time and cost of different random task graph.
- We generated computational results and graph showing time and cost effectiveness based on execution.

5. Methodology

GAMDL is XML language-based and the GAMDL syntax is developed as a set of XML-Schema. XML is the most widely used modeling language for workflow description in grid computing and has a very rich set of development tools. XML-Schema is used to define a set of rules to which an XML document must conform in order to be considered "valid". As a W3C standard, it provides a rich data model that allows us to express sophisticated structures and constraints used in GAMDL. The use of XML-Schema for GAMDL helps us easily develop a GAMDL parser using the open source XML development library, Apache XML Beans. XML Beans binds XML data with Java objects through the schema of the data expressed in XML-Schema. In our example, after we have designed the GAMDL XML-Schema, the XML Beans compiler takes the GAMDL schema and generates Java codes that access a GAMDL document. All the data types, XML documents and elements in GAMDL are mapped to Java classes. Using these automatically generated codes, we can easily develop a GAMDL parser in pure Java language.

In the context of grid computing, there exist several applications such as bioinformatics, Financial analysis etc. that can be constructed as workflows. A scheduling problem can be defined as the assignment of different grid services to different workflow tasks. Every workflow can be modeled as DAG, as shown in Figure represented by $W = (N, E, T, C)$, where N is a set of n computational tasks, T is a set of task computation volumes (one unit of computation volume is one million instructions), E is a set of communication arcs or edges that shows precedence constraint among the tasks and C is the set of communication data from parent tasks to child tasks (one unit of communication data is one Kbyte). The value of $\tau_I \in T$ is the computation volume for the task $n_i \in N$. The value of $c_{ij} \in C$ is the communication data transferring along the edge e_{ij}, $e_{ij} \in E$ from task n_i to task n_j, for n_i, $n_j \in N$. A task node without any parent node is called 'entry node', and a task node without any child node is called 'exit node'.

In this work, a fairly static methodology has been adopted for defining the weights of the computational tasks and communicating edges. In this study, we define the 'execution time (makespan)' as the total time between the finish time of exit task and start time of the entry task in the given DAG. Similarly, the 'economic cost' (EC) is the summation of the economic costs of all workflow tasks scheduled on different resources which can be defined as:

$$EC = \sum_{j=1}^{m} Dj$$

Where m is the total number of services (resources) available in grid and Dj is the execution cost due to tasks scheduled on resource j which can be calculated as:

$$Dj = PBTj \times \alpha(pj) \times Mj$$

where $\alpha(pj)$ is the processing capacity of resource j , Mj is the processing cost per MIPS (in grid dollar or g$) of executing task on resource j and $PBTj$ is the total busy time consumed by tasks scheduled on resource j [e.g., in Figure 1, PBT is three for resource P1 and eight for resource P2). In our model, the cost of the idle slots between the scheduled tasks on any resource is also considered in economic cost as it is difficult for the grid scheduler to schedule other workflow tasks in these idle slots. After assignment of all tasks in a DAG on grid resources, the makespan of the schedule will be the actual finish time of the exit task which can be computed as:

$$\text{makespan} = \text{AFT}(\text{task}_{exit}) - \text{AST}(\text{task}_{entry})$$

Where AFT and AST are the actual finish and start time of exit task and entry task respectively. In general, AST is zero except some cases where entry task has to wait for resource due to some local tasks scheduled on it. Since a large set of task graphs with different properties is used, it becomes necessary to normalize the schedule length (makespan) to a lower

bound, called the normalized schedule length (NSL) of a schedule which can be calculated as:

$$NSL = \frac{\text{makespan}}{\sum_{n \in CPmin} \min pj \in P\{\omega ij\}}$$

The denominator is the summation of the minimum execution costs of tasks on the *CP*min where computation cost of task n_i on resource p_j is ω_{ij}. The average values of NSL over several task graphs are used in the simulation.

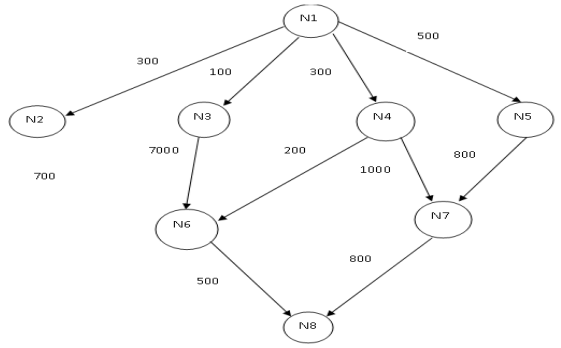


Fig: 1. Result: Direct Acyclic Graph Representation of Nodes

6. Algorithm

BLevelProcess()
[Get the current system time]
1. Set START_TIME := getCurrentSystemTime.
2. Print START_TIME.
3. Call GetBLevel(FINAL_BLEVEL) [Different Nodes]
4. Set NODES[] := {"T1", "T2", "T3", "T4", "T5", "T6", "T7", "T8"} [Nodes load in milliseconds which are assumed statically]
5. Set NODES_LOAD[] := {5000.0,4000.0,2000.0,1000.0,2000.0,4000.0,6000.0,3000.0} [Processing speed in million instruction per second]
6. Set PROCESSOR_SPEED[] := {25.0,10.0,2.0,1.0} [Machine Cost in million instruction per dollar]
7. Set MACHINE_COST[] := {1.0,2.5,3.0,2.0}
8. [Initialize] Set PROCESSOR_FREE[] := {TRUE, TRUE, TRUE, TRUE}
9. [Initialize] Set NODE_ASSIGN_TIME[] := {0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0}
10. [Initialize] Set NODE_FINISH_TIME[] := {0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0}
11. [Initialize] Set ASSIGN_PROCESSOR[] := {-1,-1,-1,-1,-1,-1,-1,-1}
12. [Initialize] Set PROCESSOR_ASSIGN_TASK[] := {-1,-1,-1,-1}
13. [Initialize] Set X := 0
14. [Initialize] Set FREE_NODE := 0
15. [Initialize] Set PROCESS_COMPLETE := FALSE
16. Repeat Steps 17 to 21 while PROCESS_COMPLETE ≠ TRUE
17. [Increment X] Set X := X + 1
18. Repeat For I = 0 to Length(NODES_LOAD) – 1

a. If NODE_ASSIGN_TIME[I] = 0.0, Then: [If node task is not started]
b. Repeat For J = 0 to Length(PROCESSOR_FREE) – 1

i. If PROCESSOR_FREE[J] = TRUE, Then:
ii. Set L := getCurrentSystemTime
iii. If I = 0, Then:
iv. Set NODE_ASSIGN_TIME[I] := L
v. Set PROCESSOR_FREE[J] := FALSE
vi. Set ASSIGN_PROCESSOR[I] := J
vii. Set PROCESSOR_ASSIGN_TASK[J] :=I
viii. Set NODE_FINISH_TIME[I] := NODE_ASSIGNTIME[I] + NODES_LOAD[I] / PROCESSOR_SPEED[J] Else:
ix. Set ASSIGN_ENGAGE_PROCESSOR:=FALSE
x. Set T := NODES_LOAD[I] / PROCESSOR_SPEED[J]
xi. Repeat For K = J To 0 Step -1
xii. Set ASSIGN_T := PROCESSOR_ASSIGN_TASK[K]
xiii. If ASSIGN_T ≠ -1 AND NODE_FINISH_TIME[ASSIGN_T] ≠ 0 AND NODE_FINISH_TIME[ASSIGN_T]< T, Then:
xiv. Set ASSIGN_ENGAGE_PROCESSOR := TRUE
xv. BREAK
[End of If Structure Step xiii] [End of Step xi Loop]
xvi. If ASSIGN_ENGAGE_PROCESSOR = FALSE, Then:
xvii. Set NODE_ASSIGN_TIME[I] := L
xviii. Set PROCESSOR_FREE[J] := FALSE
xix. Set ASSIGN_PROCESSOR[I] := J
xx. Set PROCESSOR_ASSIGN_TASK[J] := I
xxi. Set NODE_FINISH_TIME[I] := NODE_ASSIGN_TIME[I] + NODES_LOAD[I] / PROCESSOR_SPEED[J]
[End of If Structure Step xvi] [End of If Structure Step iii]
xxii. BREAK
[End of If Structure Step i][End of Step b Loop] Else:

c. Repeat For J = 0 to Length(PROCESSOR_FREE) – 1
d. If PROCESSOR_FREE[J] = FALSE AND ASSIGN_PROCESSOR[I] = J, Then:
e. Set L1 := getCurrentSystemTime

f. Set FN_TIME := L1 + NODES_LOAD[I] / PROCESSOR_SPEED[J]
g. If NODE_FINISH_TIME[J] <= FN_TIME, Then:
h. Set PROCESSOR_FREE[J] := TRUE
i. Set NODES_LOAD[J] := 0.0
[End of If Structure Step g]
[End of If Structure Step d]
[End of Step c Loop]
[End of If Structure Step a]
j. If NODES_LOAD[I] = 0.0, Then:
k. Set FREE_NODE := FREE_NODE + 1
[End of If Structure Step j]
[End of Step 18 Loop]

19. If FREE_NODE =Length(NODES_LOAD), Then:
20. Set PROCESS_COMPLETE := TRUE
21. BREAK
[End of If Structure Step 19]
[End of Step 16 Loop]
22. Set TOTAL_TIME := 0.0
23. Set TOTAL_MACHINE_COSE := 0.0
24. Repeat For J = 0 To Length(PROCESSOR_FREE) – 1
25. Set TOTAL_TIME := TOTAL_TIME + NODE_FINISH_TIME[J] – NODE_ASSIGN_TIME[J]
[End of Step 24 Loop]
26. Repeat For I = 0 To Length(NODES_LOAD) – 1
27. Set TOTAL_MACHINE_COST := TOTAL_MACHINE_COST + NODES_LOAD[I] * MACHINE_COST[ASSIGN_PROCESSOR[I]]
[End of Step 26 Loop]
28. PRINT TOTAL_TIME, TOTAL_MACHINE_COST
29. Set END_TIME = getCurrentSystemTime
30. PRINT (END_TIME – START_TIME)
31. PRINT (END_TIME – START_TIME) / 1000
32. Exit

GetBLevel(FINALBLEVEL)

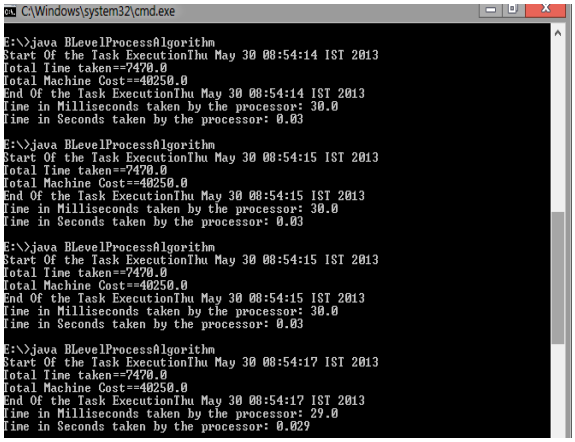
1. [Initialize] Set BLEVEL[] := {0,0,0,0,0,0,0,0}
2. [Initialize] Set MEAN_TIME[] := {1,3,4,3,6,2,5,2}
3. [Initialize] Set MEAN_P[] := (150+100+100+50+100+100)/6
4. [Initialize] Set NODE_CONNECTED[][] := {{1,2,3,4,5},{2,6},{3,6},{4,6,7},{5,7},{6,8},{7,8}}
5. [Initialize] Set NODE_LOAD[][] := {{1,300,100,300,500},{2,700},{3,700},{4,200,1000},{5,800},{6,500},{7,800}}
6. Repeat For I = 0 To Length(NODE_CONNECTED) – 1
7. Set MAX_B := 0
8. If I = 0, Then:
9. Set C := Length(BLEVEL) – 1
10. Set BLEVEL[C] := MEAN_TIME[C] + 0
11. Else:
12. Set NODE[] := NODE_CONNECTED[Length(NODE_CONNECTED) – I]
13. Set NODEL := NODE_LOAD[Length(NODE_CONNECTED) – I]
14. Set MAX_B := 0
15. Repeat For K = 1 To Length(NODEL) – 1
16. If K = 1, Then:
17. Set MAX_B := (NODEL[K] / MEAN_P) + BLEVEL[NODE[K] – 1]
18. Else:
19. If (NODEL[K] / MEANP) + BLEVEL[NODE[K] – 1] > MAX_B, Then:
20. Set MAX_B := (NODEL[K] / MEANP) + BLEVEL[NODE[K] – 1]
[End of If Structure Step 19]
[End of If Structure Step 16]
21. Set BLEVEL[Length(BLEVEL) – I – 1] := MEAN_TIME[Length(BLEVEL) – I – 1] + MAX_B
[End of Step 15 Loop]
[End of If Structure Step 8]
[End of Step 6 Loop]
22. Set FINALBLEVEL := BLEVEL
23. Return

7. Result

This algorithm is developed in Java. We have taken eight nodes and numbers of processor are four. On computing the above java code we have following output. We obtained different output on changing Node Load**(load in million instructions)** and changing on Processor Speed**(in million instruction per second)** .We observed that when load is same and processor speed varies, the machine cost remains same there is only change in total time taken by nodes. One important factor to be pointed out is the time taken by processor will be stable after a certain amount of compilation of code whether there is change in node load or in processor speed (as shown in figures below):

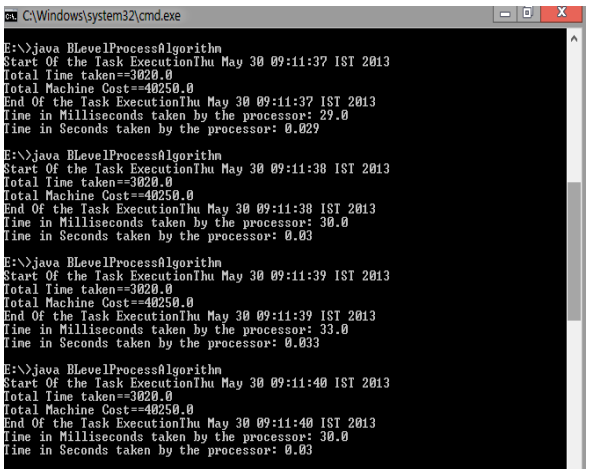
- **Execution Results**
  - **When Load on eight nodes(load in million instructions) :**

{5500.0,45000.0,2500.0,1500.0,2500.0,4500.0,6500.0,3500.0}
Processor Speed(in million instruction per second)={25.0,10.0,2.0,1.0}



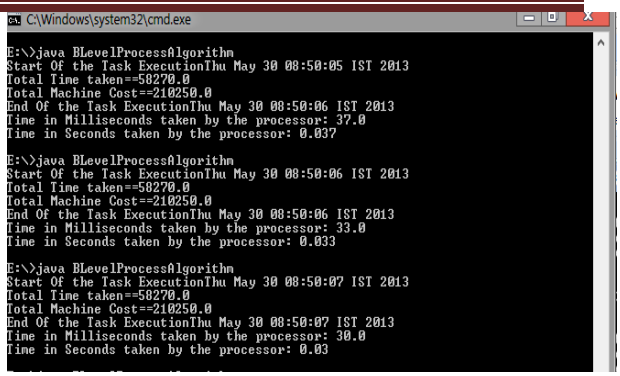
- **When Load on eight nodes(load in million instructions) :**

{5500.0,45000.0,2500.0,1500.0,2500.0,4500.0,6500.0,3500.0}
Processor Speed(in million instruction per second)={25.0,20.0,10.0,5.0}



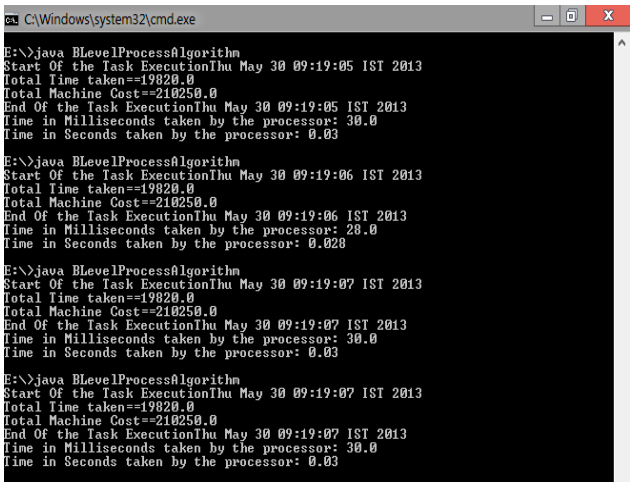
- **When Load on eight nodes(load in million instructions) :**

{25500.0,245000.0,22500.0,21500.0,22500.0,24500.0,26500.0,23500.0}
Processor Speed(in million instruction per second)={25.0,10.0,2.0,1.0}



- **When Load on eight nodes(load in million instructions) :**

{25500.0,245000.0,22500.0,21500.0,22500.0,24500.0,26500.0,23500.0}
Processor Speed(in million instruction per second)={25.0,20.0,10.0,5.0}



8. Conclusion

Computational Grids enable the creation of a virtual computing environment for sharing and aggregation of distributed resources for solving large-scale problems in science, engineering and commerce. The resources in the Grid are geographically distributed and owned by multiple organizations with different usage and cost policies. They have a large number of self-interested entities (distributed owners and users) with different objectives, priorities and goals that vary from time to time. The management of resources in such a large and distributed environment is a complex task. In this, a novel bi-criteria workflow scheduling approach has been presented and analyzed. We have proposed an efficient scheduling algorithm called 'Time and Cost effective Task Scheduling in Grid Environment' (TCTSGE) which optimizes the make span and economic cost of the schedule and

minimizes the requirements of processors. The algorithms have been implemented to schedule different random DAGs onto different grids of heterogeneous clusters of various sizes. The schedule generated by TCTSGE algorithm is better than other related bi-criteria algorithms in respect of both make span and economic cost.

References

[1] Fost, Ian, Carl Kesselman, Steve Tuecke, The Aanatomy of the Grid: Enabling Scalble Virtual Organization, International Journal of Supercomputer Application, 2001

[2] Klaus Krauter Rajkumar Buyya, Muthucumaru Maheswaran. A Taxonomy and Survey of Grid Resource Management System for Distributed Computing, Software: Practice and Experience (SPE) journal, Wiley Press, USA, 2001

[3] H. Topcuoglu,S. Hariri, M. Wu, Performance Effective and Low-complexity Task Scheduling for Heterogeneous Computing, IEEE Transactions on Parallel and Distributed Systems, 13(3), 260-274, 2002

[4] R. Sakellariou, H. Zhao, A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems, 13th Heterogeneous Computing Workshop (HCW'04), Santa Fe, New Mexico, USA, 26-30, 2004

[5] M. Maheswaran, H. J. Siegel, A Dynamic Matching and Scheduling Algorithm for Heterogeneous Computing Systems, in Proceedings of the Seventh Heterogeneous Computing Workshop, 1998

[6] D. Abramson, J. Giddy, L. Kotler, High performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid? Proceedings International Parallel and Distributed Processing Symposium (IPDPS 2000), Cancun, Mexico, 2000, IEEE Computer Society Press: Los Alamitos, CA, 2000

[7] R Buyya, D Abramson, J. Giddy, A Case for Economy Grid Architecture for Service-Oriented Grid Computing, International Parallel and Distributed Processing Symposium: 10th IEEE International Heterogeneous Computing Workshop (HCW 2001), 2001, San Francisco, CA. IEEE Computer Society Press: Los Alamitos, CA, 2001

[8] Sullivan III WT, D. Werthimer, S. Bowyer, J. Cobb, D. Gedye, D. Anderson, A New Major SETIP project Based on Project Serendip Data and 100,000 Personal Computers, 5th International Conference astronomy 1997 http://setiathome.ssl.berkeley.edu/woody paper.html

[9] R. Buyya, D. Abramson, Giddy J. Nimrod/G: Architecture for a Resource Management and Scheduling System in a Global Computational Grid". Proceedings 4th International Conference and Exhibition on High Performance Computing in Asia-Pacific Region (HPC ASIA 2000), Beijing, China, 2000, IEEE Computer Society Press: Los Alamitos, CA, 2000

[10] R. Buyya, D. Abramson, J. Giddy, An Economy Driven Resource Management Architecture for Global Computational Power Grids, 2000 International Conference on Parallel and Distributed Processing Techniques and applications (PDPTA 2000), 2000, Las Vegas. CSREA Press, 2000

[11] R. Buyya, J. Giddy, D. Abramson, An Evaluation of Economy-Based Resource Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications, Proceedings of the 2nd International Workshop on Active Middleware Services AMS 2000), Pittsburgh, PA, Kluwer Academic Press, 2000.

[12] S. Chapin, J. Karpovich, A. Grimshaw, The Legion Resource Management System, 5th Workshop on Job Scheduling Strategies for Parallel Processing , San Juan, Puerto Rico, Springer: Berlin, 1999

[13] Cactus Grid Computational Toolkit http://www.cactuscode.org

[14] The Globus Grid Project http://www.globus.org

[15] Common Component Architecture http://www.cca-forum.org/

[16] The Grid Computing Information Centre http://www.gridcomputing.com)

[17] Ninf network server project http://ninf.apgrid.org/