Job Scheduling for Heterogeneous Computing Environments

P. Devaki and M. L. Valarmathi

***Abstract*—Scheduling the jobs of computationally intensive applications efficiently is one of the most essential and difficult issues when aiming at high performance in heterogeneous computing environments. A large number of scheduling heuristics have been presented in literature for homogeneous computing systems. The complexity of the problem increases when job scheduling is to be carried out in heterogeneous computing system. In this paper, we present a simple algorithm Heterogeneous Task Scheduling (HTS) for a bounded number of heterogeneous machines. The aim of our algorithm is to minimize the overall completion time of jobs (makespan) submitted. The methodology used is to maintain the dynamic queue (Ready Queue) in which the ready jobs are available. From that queue the job is selected for execution on a machine which is capable of completing that job quickly. The analysis and experiments have shown that this algorithm provides comparable results in some cases and even better results in most of the cases together with low complexity when compared with the existing algorithms Critical-Path On a Processor (CPOP) and Heterogeneous Critical Parent Trees (HCPT) .**

***Index Terms*—Scheduling, computationally intensive, heterogeneous system, makespan.**

I. Introduction

Recent developments in high-speed digital communication have made it possible to connect a distributed suite of different high performance machines in order to provide a powerful computing platform called a heterogeneous computing system. This platform is utilized to execute computationally intensive applications that have diverse computation requirements [1]. However, the performance of parallel applications on such systems is highly dependent on the scheduling of the application tasks onto these machines [2]. The objective of the scheduling algorithm is to map tasks onto machines such that it has to minimize the overall completion time (makespan) and order the tasks which should satisfy the precedence constraints [1,2]. When the structure of the parallel application in terms of its task execution times, task dependencies and size of communicated data is known a priori, the application is represented with the static model, and scheduling can be accomplished statically at compile time [3,4]. HTS scheduling is done in batch mode. In the general form of static task scheduling, the application is represented by the Directed Acyclic Graph (DAG), in which the nodes represent application tasks and the edges represent inter-task data dependencies [5]. Each node is labeled by the computation cost (expected computation time) of the task and each edge is labeled by the communication cost (expected communication time). Finding an optimal solution for the scheduling problem is NP-complete [6], [7]. Therefore, heuristics can be used to find a sub-optimal schedule rather than parsing all possible schedules. The task scheduling problem has been extensively studied, and various heuristics have been proposed in the literature [5], [8]–[11]. In static scheduling, these heuristics are classified into a variety of categories (such as list-based, clustering, and duplication-based). This paper deals with list-based scheduling.

Manuscript received August 30, 2012; revised October 10, 2012.

Ms. P. Devaki and M. L. Valarmathi are with Department of CSE, Kumaraguru College of Technology, Coimbatore Tamil Nadu, India (e-mail: devaki_cbe4@yahoo.com; ml_valarmathi@rediffmail.com).

II. Problem Definition

This section presents the model of the application used for static scheduling, the model of the heterogeneous computing environments, and the scheduling objective.

A. Application Representation

The application can be represented by a DAG, G(V,E) where: V is the set of v nodes, each node v_i in V represents an application task, which is a sequence of instructions that must be executed serially on the same machine, E is the set of communication edges. The directed edge $e_{i,j}$ joins nodes v_i and v_j , where node v_i is called the parent node and node v_j is called the child node. This also implies that v_j cannot start until v_i finishes and sends its data to v_j. A task without any parent is called an entry task and a task without any child is called an exit task. If there is more than one exit (entry) task, they may be connected to a zero-cost pseudo exit (entry) task with zero-cost edges, which do not affect the schedule.

B. Heterogeneous Environment Representation

The heterogeneous computing environment model is a set P of p heterogeneous machines connected in a fully connected topology. It is also assumed that: Any machine can execute the task and communicate with other machines at the same time. Once a machine has started task execution, it continues without interruption, and after completing the execution it immediately sends the output data to all child tasks in parallel. W is a v × p computation cost matrix in which each $w_{i,j}$ gives the estimated time to execute the task v_i on the machine p_j , also known as weight matrix as shown in Table I. The communication costs per transferred byte between any two machines are stored in matrix R of size p × p. The communication startup costs of machines are given in a p-dimensional vector S. The communication cost of edge $c_{i,j}$ for transferring μ bytes data from task v_i (scheduled on p_m) to task v_j (scheduled on p_n), is defined as:

$$c_{i,j} = S_m + R_{m,n} . \mu_{i,j}$$

DOI: 10.7763/IJMO.2012.V2.213

where: S_m is pm communication startup cost (in secs), $\mu_{i,j}$is the amount of data transmitted from task v_i to task v_j (in bytes), $R_{m,n}$ is the communication cost per transferred byte from p_m to p_n(in sec/byte). Before scheduling, each task is labeled with the average execution cost. The average execution cost of task v_i is defined as:

$$\varpi_i = \sum_{j=1}^{p} \frac{\omega_{i,j}}{p},$$

and each edge is labeled with the average communication cost. The average communication cost of edge $c_{i,j}$ is defined as:

$$c_{i,j} = \overline{S} + \overline{R}.\mu_{i,j}$$

where S is the average communication startup cost and R is the average communication cost per transferred byte.

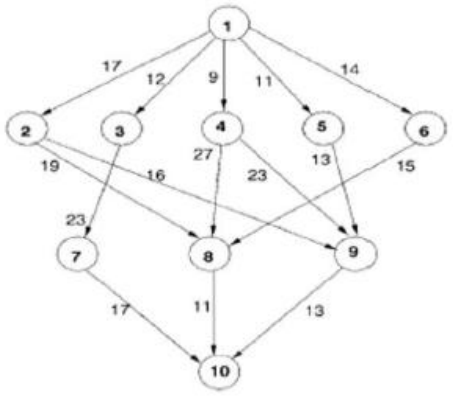

Fig. 1. Application Representation using Task Graph

TABLE I: WEIGHT MATRIX (EXPECTED TIME TO COMPUTE - ETC)

TASK	P1	P2	P3
1	14	16	9
2	13	19	18
3	11	13	19
4	13	8	17
5	12	13	10
6	13	16	9
7	7	15	11
8	5	11	14
9	18	12	20
10	21	7	16

III. PROPOSED ALGORITHM (HTS)

The proposed algorithm is maintaining the ready queue (RQ) in which there are tasks which are ready to execute. At first the entry task (task1) is ready to execute and it is in RQ. After task1 is completed, the RQ will be updated with current ready tasks. Select all the tasks in RQ one by one and check the EEFT for all the machines. The node Earliest Execution Finish Time on a machine EEFT is defined as :

$$EEFT(v_i, p_q) = \max_{v_n \in pred(v_i)} \{RT(P_q), FT(v_n) + k.c_{n,i}\} + w_{i,q},$$

where $RT(p_q)$ is the time when p_q is available. $FT(v_n)$ is the completion time of the parent node v_n and

k = 1 if the machine assigned to parent node v_n is not in p_q and,
= 0 otherwise.

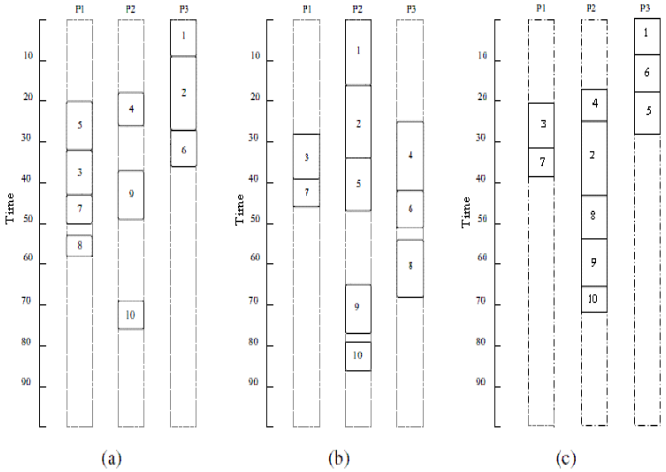

Fig. 2. Scheduling of Task Graph in the above Fig 1.using (a) HCPT (makespan = 76) (b) CPOP (makespan = 86) (c) HTS(makespan = 75) where P1,P2 and P3 represent processors.

Select the minimum EEFT (v_i,p_q) that have been calculated, and schedule the task v_i to the machine p_q .

Pseudocode of the proposed algorithm:

Initialize the Ready Queue (RQ) with entry task
While RQ is not empty do
For all tasks n_i in RQ do
Compute EEFT (n_i , p_j)
Select the task n_i which has minimum EEFT and assign it to the corresponding machine p_j.
Delete n_i from RQ
Update RQ with the successors of n_i ,
if they become ready tasks
End while

The proposed algorithm is compared with the existing algorithms Heterogeneous Critical Parent Trees (HCPT) [5] and Critical Path on a Processor (CPOP) [9]. The proposed algorithm (HTS) shows better results in terms of makespan compared to the above said algorithms.

IV. IMPLEMENTATION

Proposed algorithm is implemented in Java and input model is taken from the simulation model given in [12]. Comparison for all 12 combinations was done and the statistics given is average case of 1000 trials with different inputs.

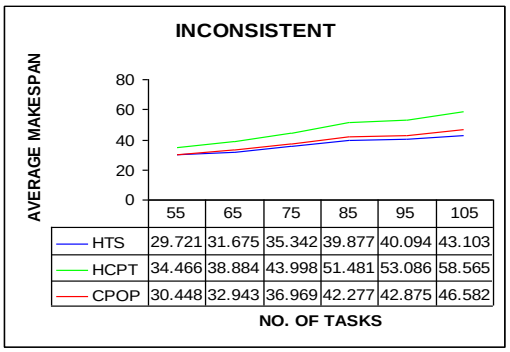

Fig. 3. Low task heterogeneity and Low machine heterogeneity

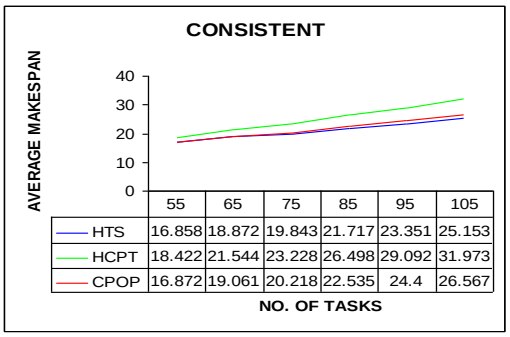

Fig. 4. Low task heterogeneity and Low machine heterogeneity

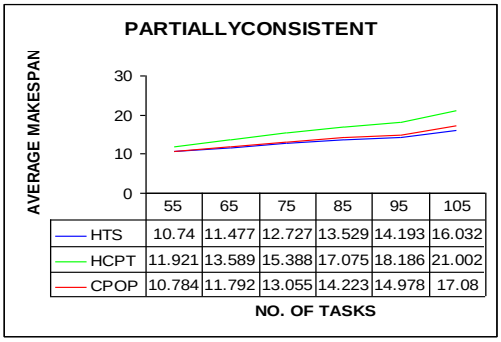

Fig. 5. Low task heterogeneity and Low machine heterogeneity

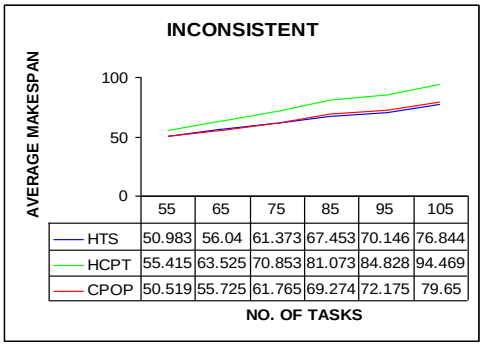

Fig. 6. Low task heterogeneity and High machine heterogeneity

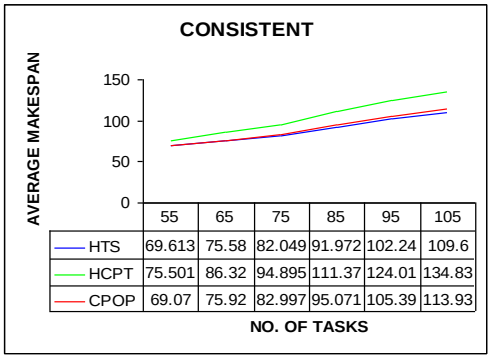

Fig. 7. Low task heterogeneity and High machine heterogeneity

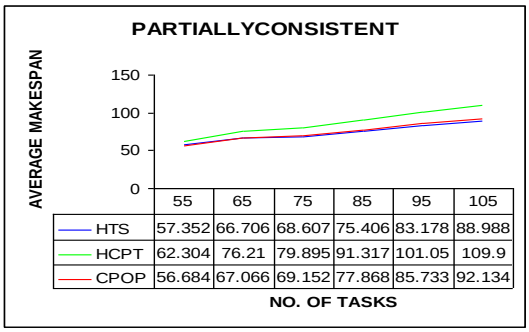

Fig. 8. Low task heterogeneity and High machine heterogeneity

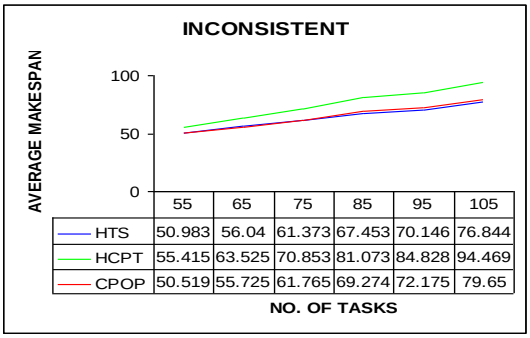

Fig. 9. High task heterogeneity and Low machine heterogeneity

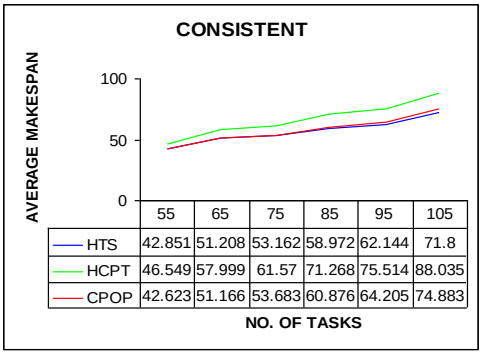

Fig. 10. High task heterogeneity and Low machine heterogeneity

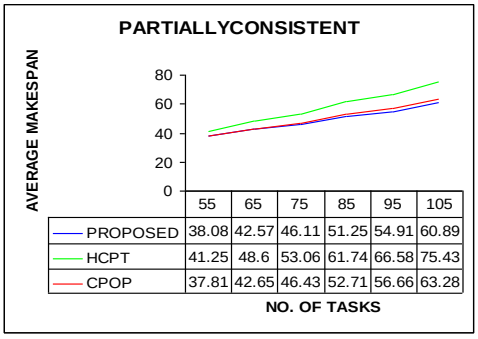

Fig. 11. High task heterogeneity and Low machine heterogeneity

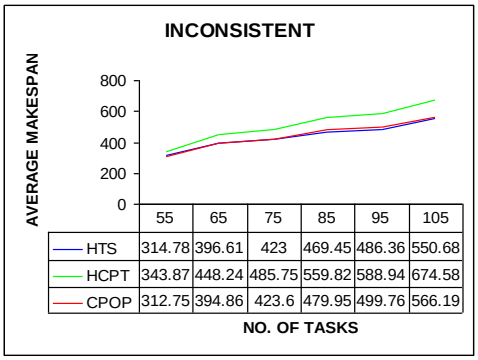

Fig. 12. High task heterogeneity and High machine heterogeneity

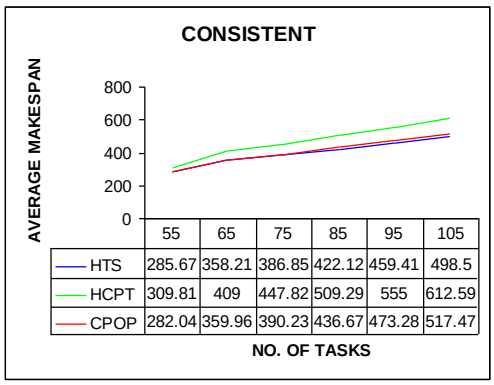

Fig. 13. High task heterogeneity and High machine heterogeneity

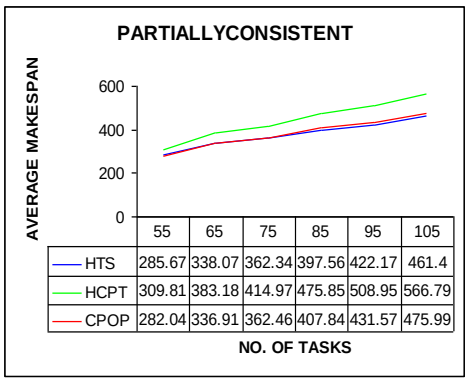

Fig. 14. High task heterogeneity and High machine heterogeneity

V. Conclusion and Future Expansion

Experimental study shows that the HTS outperformed the other algorithms in terms of performance, complexity, speedup and average makespan. In the output, for all combinations of heterogeneity, HTS performs better than the existing algorithms except for High task heterogeneity and High machine heterogeneity combination (4.4). In High task heterogeneity and High machine heterogeneity combination also when the number of tasks increases, we get better makespan compared to the existing ones. Among consistency combinations, our algorithm shows even better results for consistent one. HTS algorithm is a viable solution for the DAG scheduling problem with higher number of nodes on heterogeneous systems. HTS algorithm has given better performance and better running time results than the existing ones.

This scheduling work can also be extended by considering QoS parameters like memory and band width needed for the tasks.

References

[1] J. G. Webster, "Heterogeneous Distributed Computing," Encyclopedia of Electrical and Electronics Engineering, vol. 8, pp. 679-690, 1999.

[2] D. Feitelson, L. Rudolph, U. Schwiegelshohm, K. Sevcik, and P. Wong, "Theory and Practice in Parallel Job Scheduling," *JSSPP*, pp. 1-34, 1997.

[3] E. Ilavarasan, P. Thambidurai, and R. Mahilmannan "Performance Effective Task Scheduling Algorithm for Heterogeneous Computing System" in *Proc. 4th International Symposium on Parallel and Distributed Computing (ISPDC '05)* IEEE 2005.

[4] G. Falzon and M. Li, "Enhancing list scheduling heuristics for dependent job scheduling in grid computing environments," *J Supercomput* , March 2010.

[5] T. Hagras and J. Janecek, "A Simple Scheduling Heuristic for Heterogeneous Computing Environments," in *Proc. 2nd International Symposium on Parallel and Distributed Computing (ISPDC '03),*2003.

[6] A. Khan, C. McCreary, and M. Jones, "A Comparison of Multiprocessor Scheduling Heuristics", *ICPP*, vol.2, pp.243-250, 1994.

[7] E. U. Munir, J. Z. Li, Sheng-Fei Shi, and Q. Rasool, "Performance Analysis of task Scheduling Heuristics in Grid," in *Proc. 6th International Conference on Machine Learning and Cybernetics*, August 2007.

[8] A. Radulescu and A. VanGemund, "Fast and Effective Task Scheduling in Heterogeneous Systems," in *Proc. 9th Heterogeneous Computing Workshop*, pp.229-238, 2000.

[9] H. Topcuoglu, S. Hariri, and W. Min-You, "Performance- Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE trans. on Parallel and distributed Systems*, vol.13, no.3, pp. 260-274, 2002.

[10] E. Elavarasan and P. Thambidurai, "Low complexity performance effective task scheduling algorithm for heterogeneous computing environments," *J Comput. Sci.* vol. 3, no, 2, pp. 94-103, 2007.

[11] M. I Daoud and N. Kharma, "An efficient genetic algorithm for task scheduling in heterogeneous distributed computing systems," *IEEE Congress on Evolutionary Computation*, July, 2006.

[12] T. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen, and R. Freund. "A comparison study of static mapping heuristics for a class of meta tasks on heterogeneous computing systems," in *Proc. 8th IEEE Heterogeneous Computing Workshop (HCW'99),* pp. 15-29, April 1999.

Devaki P. was born in Erode, Tamil Nadu, on Oct 1, 1967. In 1989, she received the B. E. degree in Computer Technology and Informatics from Bharathiar University and M.E. degree in Computer Science and Engineering from Anna University, Chennai. Her research interests include parallel processing and multiprocessor scheduling. She is presently working as Associate Professor in Kumaraguru College of Technology, Coimbatore, Tamil Nadu

Dr. M. L. Valarmathi received her B.E. degree in Electrical and Electronics Engineering from ACCET, Karaikudi in 1983 and M.E. degree in Computer Science and Engineering from Government College of Technology, Coimbatore in 1990. She completed her Ph.D. in Computer Science and Engineering from Bharathiar University, Coimbatore in 2007. She is presently working as Associate Professor in Computer Science and Engineering Department, Government College of Technology, Coimbatore. Her research interests include Optimization Techniques, Image Processing and Data mining and Data Warehousing.