Comparative Study of Static Task Scheduling Algorithms for Heterogeneous Systems

Miss. Kalpana A. Manudhane[1]
ME(CSE) 2nd semester
G.H. Riasoni College of Engineering & Management
Amravati, Maharashtra, India
Email- kalpana.manudhane@gmail.com

Mr. Avinash Wadhe[2]
M-Tech(CSE)
G.H. Riasoni College of Engineering & Management
Amravati, Maharashtra, India
Email- aviwadhe@gmail.com

Abstract— **On the distributed or parallel heterogeneous computing systems, an application is usually decomposed into several interdependent sets of co-operating subtasks and assigned to a set of available processors for execution. Task scheduling is in general NP-compete problem. Static task scheduling algorithms are categorized as Heuristic based and Guided random search based scheduling algorithms. Heuristic algorithms guaranteed to find near optimal solution in less than polynomial time. Heuristic based list scheduling algorithms are Heterogeneous Earliest Finish Time (HEFT) and Critical-Path-On-a-Processor (CPOP). Whereas, Guided random search based scheduling algorithms have shown robust performance on verity of scheduling problems. Typical examples are Multiple Priority Queueing Genetic Algorithm (MPQGA), Tabu Search(TS), Ant Colony System (ACS). This paper gives comparative study of all these static task scheduling algorithms and compares them on the basis of average makespan, schedule length ratio (SLR) and speedup and running time of algorithm.**

Keywords- **Heterogeneous system; task scheduling; guided random search; heuristic list scheduling.**

I. INTRODUCTION

The *heterogeneous computing system* [1] is defined as a set of machines with different capabilities interconnected with different speed links. *Task scheduling* on heterogeneous computing systems has been well studied. Such systems are promising for fast processing of computationally intensive applications with diverse computation needs. In general, an originally large program can be decomposed into a set of smaller subtasks prior to parallel processing. These smaller subtasks almost always have dependencies representing the precedence constraints. Precedence constraints are represented as a directed acyclic graph (dag) consisting of nodes that represent computations and the directed edges that represent the dependency between the nodes. So, a task becomes ready for execution when all its immediate predecessors in dag get executed.

By decomposing a computation into smaller subtasks and executing the subtasks on multiple processors, the total execution time of the computation, namely makespan, can potentially be reduced. Hence, the goal of a task scheduling algorithm is to schedule all the subtasks on the given number of available processors so as to minimize makespan without violating precedence constraints.

It is a challenge on heterogeneous computing systems to develop task scheduling algorithms that assign the subtasks of applications to processors. Therefore, the task scheduling has been a well-studied problem on the distributed and parallel heterogeneous computing systems. Numerous algorithms have been proposed to minimize makespan.

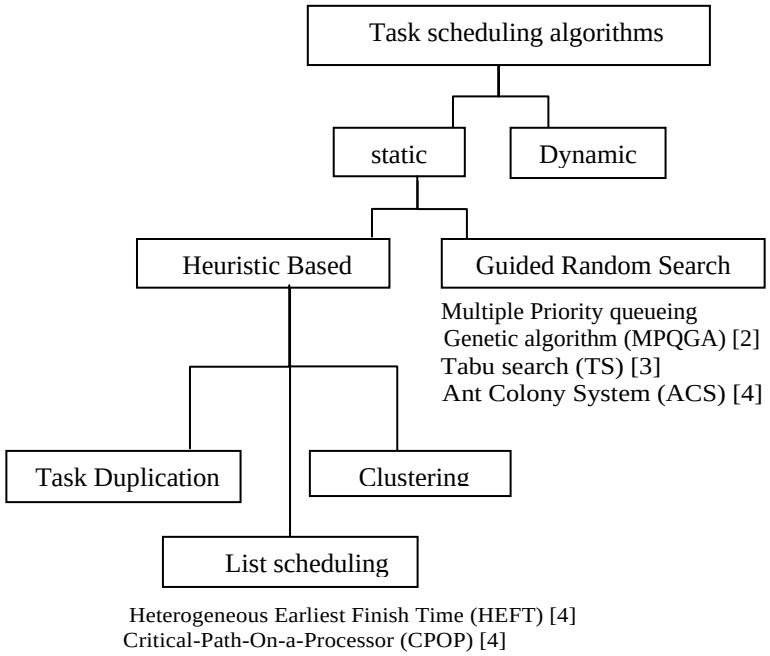

Figure 1. Classification of static task scheduling algorithms

Static task-scheduling algorithms can be divided into to main categories as shown in figure 1 [5], namely, heuristic based and guided random search based. The formal can be divided into three subgroups [5]: list scheduling, clustering and task duplication.

List scheduling Heuristics: List scheduling heuristic maintains the list of tasks according to their priorities. It has to phases: Task prioritizing – to assign priority to each task according to some criteria and processor selection- for selecting a suitable processor that minimizes a predefined cost function. Examples of algorithm for heterogeneous system are Heterogeneous Earliest Finish Time (HEFT) and Critical-Path-On-a-Processor (CPOP).

Clustering Heuristics: An algorithm in this group maps the tasks in a given graph to an unlimited number of clusters. At each step, the selected tasks for clustering can be any task, not necessarily a ready task. Each iteration refines the previous clustering by merging some clusters. If two tasks are assigned to the same cluster, they will be executed on the same processor. A clustering heuristic requires additional steps to generate a final schedule: a cluster merging step for merging the clusters so that the remaining number of clusters equal the number of processors, a cluster mapping step for mapping the clusters on the available processors, and a task ordering step for ordering the mapped tasks within each processor.

Task duplication Heuristics: The concept behind duplication based algorithms is to schedule a task graph by mapping some of its tasks redundantly, which reduces the interprocess communication overhead.

The performance of these algorithms is heavily dependent on the effectiveness of the heuristics. Therefore, they are not likely to produce consistent results on a wide range of problems [2], especially when the complexity of the DAG task scheduling problem increases.

Guided random search based algorithms: Guided random search techniques [2] use random choice to guide themselves throughout the problem. Guided random search based algorithm have robust performance on variety of scheduling problems, however they are less efficient and generate much higher computational cost than heuristic based algorithms. Genetic algorithms have been widely used to evolve solutions for many task scheduling problems. Some other examples are Tabu search (TS) and Ant Colony System (ACS).

The remainder of this paper is organized as follows: Section II gives literature review describing system and task model. In Section III, static task scheduling algorithms, namely, HEFT, CPOP, MPQGA, ACS and TS are described. Section IV gives comparison metrics and a comparative study of these task scheduling algorithms. Finally, in Section V, conclusion is presented that tells which algorithm is suitable for what requirement.

II. LITERATURE RIVIEW

The target system [2] consists of a set P of k heterogeneous processors that are fully interconnected with high-speed network. The communication time between two dependent subtasks should be taken into account if they are assigned to different processors. We also assume a static computing model in which the dependence relations and the execution times of subtasks are known a priori and do not change over the

course of the scheduling and subtask execution.

In this study, a parallel task [2] can be decomposed into an entry subtask, an exit subtask and several intermediate subtasks. For a pair of dependent subtasks, T_i and T_j, if the execution of T_j depends on the output from the execution of T_i, then T_i is the predecessor of T_j, and T_j is the successor of T_i. We use pred(T_i) and succ(T_i) to denote the set of predecessor subtasks and successor subtasks of the task T_i, respectively.

In general, the task can be represented by a weighted dag on a distributed and parallel heterogeneous computing system. Dag is a directed acyclic graph with nodes representing subtasks and edges representing execution precedence between subtasks. A weight is associated with each node and edge. The node weight denoted as $W(T_i)$ represents the subtask T_i execution time. Whereas the edge weight denoted as $C(T_i, T_j)$ represents the inter-subtask communication time between subtask T_i and subtask T_j.

In addition, in the underlying study, the computation time of the subtask T_i on a processor P_k is denoted as $W(T_i, P_k)$ and its average computation time is denoted as $W(T_i)$ is defined as

$$\overline{W(T_1)} = \frac{\sum_{j=1}^{k} W(T_i, P_j)}{k} \quad (1)$$

Communication time is only required when two subtasks are assigned to different processors. In the other words, the communication time when the subtasks are assigned to the same processor can be ignored.

The $EST(T_i, P_k)$ and $EFT(T_i, P_k)$ [5] are earliest execution start time and the earliest finish time of the node or subtask T_i on processor P_k. EST for entry task is zero.

$$EST(T_i, P_k) = \max_{T_j \in pred(T_i)} \{EFT(T_j, P_m) + C(T_j, T_i)\} \quad (2)$$

$$EFT(T_i, P_k) = EST(T_i, P_k) + W(T_i, P_k) \quad (3)$$

The upward-ranking of task T_i can be denoted as $Rank_u(T_i)$, as shown in Equation

$$Rank_u(T_i) = \overline{W(T_1)} + \max_{T_j \in succ(T_i)} \left(\overline{C(T_1, T_j)} + Rank_u(T_j) \right) \quad (4)$$

The downward-ranking of subtask T_i can also be denoted as $Rank_d(T_i)$, as shown in Equation

$$Rank_d(T_i) = \max_{T_j \in pred(T_i)} \left(\overline{W(T_1)} + \overline{C(T_1, T_j)} + Rank_d(T_j) \right) \quad (5)$$

Figure 2 shows how upward ranking and downward-ranking are computed. Note that, both computation and communication costs are averaged over all nodes and links. The downward-ranking of a subtask is defined as the summation of the computation and communication costs along the longest path of the node from the entry subtask in the task graph. The subtask itself is excluded from the computation. As shown in Figure 2, $rank_d$ Of T_4 involves node T_1 and T_2 which are along the longest path.

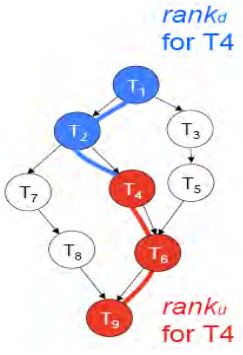

Figure 2. Downward and upward ranking

In contrast, the upward-ranking of a subtask is computed by adding the computation and communication costs along the longest path of the subtask from the exit subtask in the task graph (including the subtask). As shown in Figure 2, $rank_u$ of T4 involves T_4, T6 and T_9.

III . TASK SCHEDULING ALGORITHMS

In this section we study various static task scheduling algorithms.

A. Heuristic List scheduling algorithms

Heuristic algorithms guaranteed to find near optimal solution in less than polynomial time. We now study HEFT and CPOP heuristic list based algorithms.

- *Heterogeneous Earliest Finish Time (HEFT):*

The HEFT algorithm is an application scheduling algorithm for bounded number of heterogeneous processors. It has to major phases, namely, task prioritizing and processor selection phase.

Task prioritizing phase:

This phase computes priority of each task. The priority is nothing but the upward rank value, $rank_u$. The task list is generated by sorting the tasks by decreasing order of $rank_u$. The tie-breaking is done randomly to avoid complexity. It is clear that the decreasing order of $rank_u$ provide topological order of tasks, which is the linear order that preserves the precedence constraints.

Processor selection phase:

HEFT algorithm has insertion based policy which considers possible insertion of a task in an earliest idle time slot between two already scheduled tasks on a processor. The length of an idle time slot, i.e., the difference between execution start time and finish time of two tasks that were consecutively scheduled on the same processor, should be at least capable of computation cost of the task to be scheduled.

A task become ready for execution when its immediate predecessors in task graph get executed. In HEFT algorithm, the search of an idle time slot of a task T_i on processor P_j starts at the ready time of T_i on P_j.

- *Critical-Path-On-a-Processor (CPOP):*

Although CPOP algorithm has task prioritizing and processor selection phases, as in HEFT, it uses different attributes for assigning task priorities and different strategy for determining the best processor for each selected task.

Task prioritizing phase:

In this phase, upward rank, $rank_u$ and downward rank, $rank_d$ values of all tasks are computed. The CPOP uses the critical path of given application graph. The length of this path, |CP|, is sum of the computation costs of the tasks on the path and inter-task communication costs along the path.

The priority of each task is assigned as summation of $rank_u$ and $rank_d$. Priority queue is maintained (with key $rank_u + rank_d$) to contain all ready tasks at any given instant. A binary heap was used to implement the priority queue. At each step, the task with highest priority is selected from priority queue.

Initially, the entry task is the selected task and marked as a critical path task. An immediate successor that has the highest priority value is selected and it is marked as a critical path task. This process is repeated until the exit node is reached. For tie-breaking, the first immediate successor which has the highest priority is selected.

Processor selection phase:

The critical path processor, pcp, is one that minimizes the cumulative computation costs the tasks on the critical path. If the selected task is on the critical path, then it is scheduled on pcp; otherwise, it is assigned to a processor which minimizes the Earliest Finish Time, EST, of the task. Both cases consider an insertion based scheduling policy.

B. Guided Random search based algorithms

Guided random search based scheduling algorithms have shown robust performance on verity of scheduling problems. Let us study now MPQGA, ACS and TS.

- *Multiple Priority queueing Genetic algorithm (MPQGA):*

Genetic algorithm [6] is a random search method, which is widely used for solving combinatorial optimization problems. It executes in generations, producing better and better solutions using crossover and mutation

operators in each generation by "weeding" out poor solutions in each generation and randomly producing new solutions (offspring) for the next generation based on the solutions (parents) in the current solution.

The Multiple Priority queuing Genetic algorithm (MPQGA) comprises two key components: (1) a genetic algorithm to generate multiple priority queuing for task scheduling on distributed and parallel heterogeneous computing systems and (2) a heuristic based heterogeneous earliest finish time (HEFT) approach to search for a solution for mapping tasks to processors.

GA use a collection of solutions which evolves through genetic operators to obtain better solutions. New solutions (offspring) for the next generation are obtained by applying the following two genetic operators :

- Crossover, which aims to take the best features of each parent and mix the remaining features in forming the offspring.
- Mutation, which aims to introduce variations into the individuals.

Fitness value plays an important role in deciding which individuals would be used to generate the next-generation population while the genetic operators realize the concrete evolution.

makespan is the largest finish time among all subtasks, which is equivalent to the actual finish time of the exit node T_{exit}. For the task scheduling problem, the goal is to obtain subtask assignments that ensure minimum makespan to ensure that the precedence of the subtasks is not violated. Hence, the Fitness function value is defined as

$$Value_{Fitness} = makespan = EF\ T(T_{exit}) \quad (6)$$

The HEFT algorithm is used to map the subtasks to the processors. The subtasks have been assigned to the processors in order of their priority. At each step of the assignment, the selected processor provides the earliest finish time for the subtask under consideration, taking into account all the communications from the sub- task's parents.

- *Ant Colony System (ACS):*

The elementary idea of ACS is to simulate the foraging behavior of ant colonies. When a group of ants sets out from the nest to search for the food source, they use a special kind of chemical to communicate with each other. The chemical is referred to as pheromone. Once the ants discover a path to food, they deposit pheromone on the path. By sensing pheromone on the ground, an ant can follow the trails of the other ants to the food source. As this process continues, most of the ants tend to choose the shortest path as there have been a huge amount of pheromones accumulated on this path as shown in Figure 3. This collective pheromone-depositing and pheromone-following behavior of ants becomes the inspiring source of ACS that is applied to task scheduling in heterogeneous system.

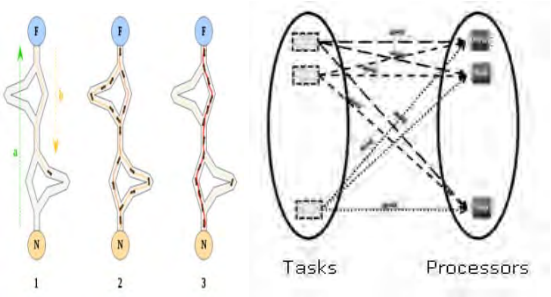

Figure 3. Behavior of ant colony and applying it to task scheduling

Informally, the algorithm can be viewed as the interplay of the following procedures:

1) Initialization of algorithm: All pheromone values and parameters are initialized at the beginning of the algorithm.
2) Initialization of ants: A group of M artificial ants are used in the algorithm. In each iteration, each ant randomly selects a constructive direction and builds a sequence of tasks.
3) Solution construction: M ants set out to build M solutions to the problem based on pheromone and heuristic values using the selection rule of the ACS algorithm.
4) Local pheromone updating: Soon after an ant maps a service instance s_i^j to task T_i, the corresponding pheromone value is updated by a local pheromone updating rule.
5) Global pheromone updating: After all ants have completed their solutions at the end of each iteration,

pheromone values corresponding to the best-so-far solution are updated by a global pheromone updating rule.

6) Terminal test: If the test is passed, the algorithm will be ended. Otherwise, go to step 2) to begin a new iteration.

The flowchart of the algorithm is given in Figure 4.

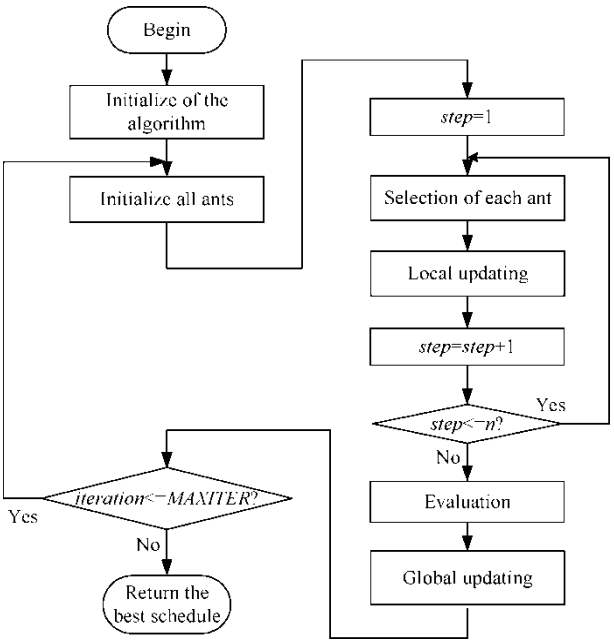

Figure 4. Flowchart of the ACS algorithm

- *Tabu Search :*

Tabu search [6] is the technique that keeps track of the regions of the solution space that have already been searched in order to avoid repeating the search near these areas. A tabu list is constructed by short-hop and long-hop procedures, and the best solution from the list is produced as the Tabu algorithm solution.

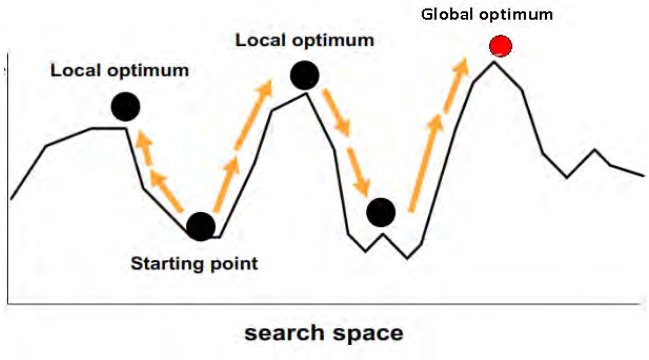

Figure 5: Behavior of Tabu search algorithm

A task scheduling algorithm using the Tabu method [3] is the neighborhood search technique that tries to avoid local minima and attempts to guide the search towards a global minimum. Tabu search starts with an initial solution, which can be obtained by applying a simple one-pass heuristic, and scans the neighborhood of the current solution—that is, all the solutions that differ from the current one by a single move. For the multiprocessor task-scheduling problem, a move consists of moving a task from one processor to some other processor, or changing the order of execution of a task within the list of tasks scheduled to a processor. This technique considers all the moves in the immediate neighborhood, and accepts the move which results in the best makespan.

III. COMPARATIVE STUDY

The comparisons of the algorithms are based on following metrics [5]:

1.Average Makespan: The makespan is the largest finish time among all subtasks (formula 6), which is equivalent to the actual finish time of the exit node . Texit. We consider average makespan obtained from various random task graphs

2. Schedule length ratio(SLR): The main performance measure of scheduling algorithm on a graph is the schedule length (makespan) of its output schedule. Since the large set of task graphs with different properties is used, it is necessary to normalize to schedule length to a lower bound, which is called the Schedule Length Ratio (SLR). The SLR value of an graph is defined by

$$\text{SLR} = \frac{\text{makespan}}{\sum_{n_i \in CP_{MIN}} \min_{p_j \in Q}\{W_{i,j}\}} \tag{7}$$

The denominator is the summation of the minimum computation costs of tasks on the CP_{MIN}.
For an unscheduled dag, if the computation cost of each node n_i is set with minimum value, then the critical path will be based on minimum computation costs, which is represented as CP_{MIN}. The SLR cannot be less than one since the denominator is the lower bound.

3. Speedup: The speedup value for a given graph is computed by dividing the sequential execution time by parallel execution time (i.e. the makespan of output schedule. The sequential execution time is computed by assigning all tasks to a single processor that minimizes the computation costs.

$$\text{Speedup} = \frac{\min_{p_j \in Q}\left\{\sum_{n_i \in V} W_{i,j}\right\}}{\text{makespan}} \tag{8}$$

4. Running time of algorithms: The running time of an algorithm is its execution time for obtaining output schedule for a given task graph.

Now we compare these algorithms in pair or groups.

A. Comparison between HEFT and CPOP:

The HEFT and CPOP are compared [5] for different task graphs on heterogeneous system on the basis of average SLR, Average speedup and running time of algorithm.

- Average SLR of HEFT is better than CPOP by 7 percent.
- Average speedup of HEFT is better than CPOP by 6 percent.
- Running time of HEFT is faster than CPOP by 10 percent.

So, HEFT gives better performance than CPOP. CPOP gives better results as compared to other algorithms (Dynamic Level Scheduling, Mapping Heuristic) for graphs with higher CCRs than graphs with lower CCRs. CCR is nothing but communication to computation ratio.

B. Comparison between MPQGA and HEFT:

The MPQGA and HEFT are compared [2] for different task graphs on heterogeneous system on the basis of metric average makespan that gives following results:

- If the number of processors is considered as fixed value (say 32), CCR considered as 1, as the subtask numbers is increased, MPQGA always outperforms HEFT.
- If CCR value is smaller (0.1) and the processor numbers is increased, the results illustrate that the makespan reduces very fast.

As a result, this algorithm can cover a large search space than deterministic scheduling approaches without incurring high computational cost. The MPQGA algorithm outperforms HEFT with higher speedup on subtask execution.

C. Comparison of GA and TS:

Random search based algorithms – Genetic algorithm (GA) and Tabu Search (TS) yield better solutions with shorter makespan than HEFT. In this group best solutions were obtained by both -Genetic algorithm and Tabu Search [3].

D. Analysis of ACS:

An ACS algorithm [4] a large scale workflow scheduling problem in computational grids has been proposed. In the algorithm different QoS parameters are considered, including reliability, time and cost. Users are allowed to

define QoS constraints to guarantee the quality of schedule. Moreover, the optimizing objective of the algorithm is based on the user defined QoS preferences.

IV. CONCLUSION

In this paper, we studied static task scheduling algorithms for heterogeneous system. We mainly studied Guided Random Search based algorithms- MPQGA, TB , ACS and Heuristic list based scheduling- HEFT and CPOP. This paper also compared their performance on the basic of metrics - average make-span, Schedule Length Ratio (SLR), speed up and running time of algorithm.

HEFT is better than CPOP on the basis of average SLR, average speedup and running time of algorithm. Whereas, Genetic algorithm such as MPQGA outperforms HEFT on the basis of average makespan and speedup. In case of Ant Colony System, users are allowed to define QoS constraints to guarantee the quality of schedule.

So, this paper conclude that performance of algorithms differ according to comparison metric chosen. Algorithm should be chosen as per ones metric requirement.

V. REFERENCES

[1] Tarek Hagras, Jan Janecek, "A Near Lower-Bound Complexity Algorithm for Compile-Time Task-Scheduling in Heterogeneous Computing Systems", Proceedings of the ISPDC/HeteroPar'04 IEEE, 2004.

[2] Yuming Xu, Kenli Li, Tung Truong Khac, Meikang Qiu, "A Multiple Priority Queueing Genetic Algorithm for Task Scheduling on Heterogeneous Computing Systems", IEEE 14th International Conference on High Performance Computing and Communications, pp. 639-646, 2012

[3] Shiyuan Jin, Guy Schiavone , Damla Turgut, "A performance study of multiprocessor task scheduling algorithms", J Supercomput (2008) 43,pp. 77–97, 2008.

[4] Wei-Neng Chen, "An Ant Colony Optimization Approach to a Grid Workflow Scheduling Problem With Various QoS Requirements", IEEE transactions on system, man and cybernetics, part C: Applications and reviews, Vol. 39, NO. 1, pp. 29-43, 2009.

[5] Haluk Topcuoglu, Salim Hariri, Min-You Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing", IEEE transactions on parallel and Distributed systems, Vol. 13, No. 3,pp. 260-274, 2002.

[6] Sang Cheol Kim, Sunggu Lee and Jaegyoon Hahm, "Push-Pull: Deterministic Search-Based DAG Scheduling for Heterogeneous Cluster Systems", IEEE Transactions on Parallel and Distributed Systems, vol. 18, pp. 1489-1502, November 2007.

[7] Andre R. Brodtkorb, Christopher Dyken, Trond R. Hagen, Jon M. Hjelmervik a and Olaf O. Storaasli, "State-of-the-art in heterogeneous computing", IOS Press, Scientific Programming 18 (2010),pp. 1–33 , 2010.

[8] R.Eswari and S.Nickolas,"Path-based Heuristic Task Scheduling Algorithm for Heterogeneous Distributed Computing Systems", 2010 International Conference on Advances in Recent Technologies in Communication and Computing, pp. 30-34,2010.

[9] Srishti Srivastava, Nitin Sukhija, Ioana Banicescu and Florina M. Ciorba, "Analyzing the Robustness of Dynamic Loop Scheduling for Heterogeneous Computing Systems", 2012 11th International Symposium on Parallel and Distributed Computing,pp. 156-163, 2012.

[10] Tong Li, Paul Brett, Rob Knauerhase, David Koufaty, Dheeraj Reddy and Scott Hahn, "Operating System Support for Overlapping-ISA Heterogeneous Multi-core Architectures" , Intel Corporation, pp.1-12.

AUTHORS PROFILE

Kalpana Manudhane**:** Received the B.E in computer engineering from North Maharshtra University, SSB's COET, Bambhori, Jalgaon (khan) in 2005. She is currently an Assistant Professor in CSE department .in COET, Akola from SGBAU University. She is pursuing for ME CSE from G.H. Raisoni, Amravati. Her research interest include task scheduling, compiler techniques, operating system and programming.

Prof. Avinash P.Wadhe**:** Received the B.E and from SGBAU Amravati university and M-Tech (CSE) From G.H Raisoni College of Engineering, Nagpur (an Autonomous Institute). He is Currently an Assistant Professor with the G.H Raisoni College of Engineering and Management,Amravati SGBAU Amravati university. His research interest include Network Security , Data mining and Fuzzy system .He has contributed to more than 20 research paper. He had awarded with young investigator award in international conference**.** .