Chapter 4

Taxonomy of Task Allocation Models and Related Work

Over the past two decades, a number of studies in optimization techniques and their applications to parallel and distributed applications have led to the identification of several challenging problems. One of these problems is optimally assigning the tasks of an application to the machines within the parallel and distributed system. This problem is known as task allocation problem or task assignment problem or scheduling. The Task Allocation Problem is NP-complete for most of its variants except for a few highly simplified cases [73] [79] [80] [102] [111]. As a result, it has attracted the attention of many researchers and has been extensively studied. Consequently, numerous approaches based on various constraints and assumptions have been reported to solve this problem. In a parallel and distributed system, it is essential to assign each task to the machine whose characteristics are most suitable to execute that task.

This chapter begins with a precise introduction to the task allocation problem. This is followed by an overview on the known task allocation models and their taxonomy based on the solution techniques used. Basic techniques used for task allocation have been reviewed with their essential characteristics and constraints.
4.1 The Task Allocation Problem

The problem being addressed in this thesis is concerned with allocating the tasks of a parallel program among machines of a parallel and distributed system in such a way that load on each machine remains balanced and some objectives under defined constraints are achieved.

A task allocation system consists of the following:

- Program Tasks
- Target System
- Allocation Schedule in which specific performance criterion is optimized.

Program Tasks

The characteristics of a parallel program can be defined as the system \((\tau, \Delta, ITC, EA)\) as follows:

- \(\tau = \tau_1, \tau_2, \ldots, \tau_n\) is a set of tasks to be executed.
- \(\Delta\), is partial order defined on \(\tau\), which specifies operational precedence constraints. That is \(\tau_i < \tau_j\) means that task \(\tau_i\) must be completed before task \(\tau_j\) can start execution.
- \(ITC\), is an \(n \times n\) matrix of communication data, where \(ITC_{ij} \geq 0\) is the amount of data required to be transmitted from task \(\tau_i\) to task \(\tau_j\), \(1 \leq i, j \leq n\).
- \(EA\), is an \(n\) vector of the computations i.e. \(EA_i \geq 0\) is a measure of the amount of computation at task \(\tau_i\), \(1 \leq i \leq n\).

The relationship among tasks in parallel and distributed system may or may not include precedence constraints. When some precedence constraints need to be enforced, the partial order \(\Delta\) is conveniently represented as a directed acyclic graph (DAG), called a task graph. In this case allocating these tasks is usually referred to as Precedence Constraints Allocation/Scheduling. A task graph \(G = (\tau, E)\) has
a set of nodes $\tau$ and a set of directed edges $E$. A directed edge $(i, j)$ between two tasks $\tau_i$ and $\tau_j$ specifies that $\tau_i$ must be completed before $\tau_j$ can begin. Associated with each node $\tau_i$ is its computational needs $EA_i$ (how many instructions or operations, for example). Associated with each edge $(i, j)$ connecting tasks $\tau_i$ and $\tau_j$ is the data size $ITC_{ij}$, that is, the size of a message from $\tau_i$ to $\tau_j$. Figure 4.1 shows a typical DAG.

![Figure 4.1: Directed Acyclic Graph](image)

When there are no precedence constraints among the tasks, the relationships are only communication among tasks, which are represented by a undirected graph called a Task Interaction Graph (TIG). Figure 4.2 shows a TIG. This work addresses the allocation of Directed Acyclic Graphs (DAGs) on parallel and distributed systems.

**Target System**

The target system consists of a set $M$ of $m$ heterogeneous machines connected using a high-speed message passing interconnection network. The connectivity of
4.1. The Task Allocation Problem

Figure 4.2: Task Interaction Graph

the machines can be represented using an undirected graph called the network graph. Associated with each edge \((i, j)\) connecting two processing elements \(M_i\) and \(M_j\) in the network graph is the transfer rate \(R_{ij}\), that is, how many units of data can be transmitted per unit of time over the link. As pointed out in Chapter 1, the target system addressed in this work is a heterogeneous multicomputer system.

The Allocation Schedule

An allocation schedule of the task graph \(G = (\tau, E)\) on a system of \(m\) machines is a function \(f\) that maps each task to a machine. Formally, \(f : \tau \rightarrow \{1, 2, \ldots, m\}\). If \(f(v) = i\), for some \(v \in \tau\), we say that task \(v\) is scheduled to be processed by machine \(M_i\).

The goal of any allocation schedule is to minimize the total completion time of a parallel program. This performance measure is known as the schedule length or
the maximum finishing time of any task.

**Execution and Communication Time**

Once the parameters of the task graph and the target system are known, the execution and communication times can be obtained as follows.

- The execution time, $e_{ij}$, of task $\tau_i$ when executed on machine $M_j$ whose computational speed is $s_j$ is given by:

$$e_{ij} = \frac{E A_i}{s_j}$$

The work addressed in this thesis assumes that task execution times are unknown.

- The communication delay (over a free link), $c_{ij}$, between tasks $\tau_i$ and $\tau_j$ when they are executed on adjacent machines $M_i$ and $M_j$ is given by:

$$c_{ij} = \frac{ITC_{ij}}{r_{kl}}$$

where $r_{kl}$ represents the bandwidth of like $(k,l)$.

**4.2 Task Allocation and Load Balancing**

In parallel and distributed systems, an application is divided into a fixed number of tasks that are to be executed in parallel. If these tasks are simply allocated to the available machines without any consideration of the types of processing elements and their speeds, there is a distinct possibility that some processing elements will complete their tasks before others and will become idle since the tasks are unevenly distributed or some processing elements may operate faster than others (or a combination of both situations). To achieve minimum execution time, all processing elements must operate continuously on the tasks allocated to them. When tasks are divided among the processing elements evenly with the goal of minimizing the application’s execution time, this is termed as load balancing [220].
Figures 4.3 and 4.4 explain how load imbalance leads to a larger application execution time. Two types of load balancing schemes have been reported in literature [220]:

**Static Load Balancing**

Static load balancing attempts to allocate tasks to the machines before execution begins. All parameters required to perform load balancing i.e. the characteristics of tasks, machines and interconnection network are known *a priori* and remain constant throughout the allocation process. Load balancing decisions are made at compile time either deterministically or probabilistically. This form of load balancing is usually referred to as mapping [47] or scheduling. These schemes are simple to implement and has less runtime overhead. Some well known static load balancing techniques are as follows [220]:

- **Round Robin Technique**: To distribute the load evenly on available machines, tasks are allocated to them in a round robin order i.e. in circular order without any priority. The scheduler assigns the \((i + 1)^{th}\) task to the first node if the \(i^{th}\) task is allocated to last node. Round robin techniques are simple and straightforward but under performs when tasks are of unequal processing time.

- **Randomization Techniques**: In these techniques, the task and machine are selected randomly, and then the selected task is allocated to the selected machine. Once again, the approach is simple but suffers from the same drawback as the round robin technique.

- **Partitioning Techniques**: In these techniques, the parallel application is divided into tasks of equal computational load whilst minimizing inter-task communication.

Partitioning techniques to map a two dimensional data matrix onto heterogeneous resources have been investigated by Crandall and Quinn [82] and
4.2. Task Allocation and Load Balancing

Kaddoura et al. [131]. The two papers are all based on Recursive Bisection algorithm.

- **Heuristics**: Heuristics make use of different approaches for allocating tasks of a parallel program to machines. Section 4.3.6 on page 110 presents various heuristic approaches used for task allocation.

The work addressed in this part of this thesis is related to task allocation with load balancing i.e. to implement load balancing whilst allocating tasks. So it is essential to have a taxonomy and literature review on various approaches used for task allocation. Section 4.3 on page 98 covers the same. Further the terms task allocation, task assignment, scheduling and mapping have been assumed to be synonyms within this thesis.

**Dynamic Load Balancing**

Dynamic load balancing schemes implement load balancing by transferring the load from heavily loaded machines to lightly loaded machines during program execution. They need not be aware about run time parameters of a program i.e. the characteristics of tasks, machines and interconnection network may not be known *a priori* and do not remain constant during program execution. Information policy, location policy and transfer policy are key terms used to describe a dynamic load balancing scheme. Information policy is used to update the machines on the number of tasks waiting in the queue; transfer policy determines whether a task should be processed on the same machine or should be transferred to some other machine to improve the performance whilst the location policy identifies the machine to which the task should be transferred. They have some advantages over static load balancing schemes but generate more runtime overhead compared to them.

Dynamic load balancing is beyond the scope this thesis hence details regarding this scheme is not discussed further.
4.2. Task Allocation and Load Balancing

Figure 4.3: Load Imbalance: Larger Execution Time

Figure 4.4: Perfect Load Balancing: Less Execution Time
4.3 Taxonomy of Task Allocation Models

Several strategies [202] [181] [47] [211] [187] [54] [233] [108] [44] [175] [63] [218] [208] [187] [53] [28] [30] [28] have been reported in literature to solve the task allocation problem both in the field of optimization and computer science. These strategies make use of numerous methods such as $A^*$-Algorithm, min-cut max flow, clustering, greedy approach, simulated annealing, tabu search etc. Though no general classification of these approaches exist, yet they may be grouped at different levels of hierarchy as shown in Figure 4.5.

At the first level of hierarchy these algorithms can be roughly classified into two categories namely, exact algorithms and approximate algorithms. [202] [181] [47] [211] [108] [44] [233] [28] [30] [232]. In the following subsections we briefly discuss each of them.

4.3.1 Exact Algorithms

Two categories of exact algorithms are - Restricted Exact Algorithms and Non-Restricted Exact algorithms. Restricted exact algorithms place some restrictions on the parallel program structure or on the interconnection network or on both and then lead to an exact solution in polynomial time.

Non-Restricted Exact algorithms on the other hand place no restrictions, either on the program structure or on the interconnection network and lead to an optimal solution but not in polynomial time.

4.3.2 Approximate Algorithms

Approximate algorithms may further be classifies as - Random Optimization and Heuristics. Random optimization algorithms use the same techniques as that used by the exact optimization algorithms to solve the problem. But, they restrict the solution search criteria by employing certain metrics which decide ”how good the solution is”. When a solution of certain satisfactory level is achieved the algorithm
4.3. Taxonomy of Task Allocation Models

Figure 4.5: Task Allocation Taxonomy
terminates, declaring it as a good solution.

Heuristics on the other hand make use of techniques which effect the entire process in an indirect fashion. For example, in a clustering technique, tasks with extremely high inter-task communication are clustered and then allocated to the same machine in order to reduce the communication overhead. In a greedy approach, some tasks from the task set are allocated in the first step and then each subsequent step allocates only one task without backtracking until the entire task set is allocated. In iterative approaches, the entire task set is allocated in the first step and then the allocation is improved by checking certain system parameters in subsequent steps. These steps may involve techniques like task migration (moving task from one machine to another) and pairwise exchange etc.

Looking down in the hierarchy in Figure 4.5 and based on the above discussion, it may be concluded that allocation algorithms may be broadly classified into following four categories:

- Mathematical Programming
- Graph Theoretic
- State Space Search
- Heuristics.

In the following subsections, each of them is reviewed along with related literature.

### 4.3.3 Mathematical Programming Techniques

Mathematical programming approach has been recognized as the most powerful approach for modeling and analyzing several kinds of problems. This approach formulates task allocation problem as an optimization problem and solves it with mathematical programming techniques. From this approach following techniques have been developed for the task allocation problem:
4.3. Taxonomy of Task Allocation Models

- Branch and Bound Technique
- Integer Programming Technique
- Dynamic Programming Technique

Branch and Bound Technique

One of the most popular techniques among researchers and widely investigated in distributing systems for task allocation is the branch and bound (BB) technique. This technique is an exhaustive search approach. Using this technique, numerous algorithms for task allocation in distributed system have been reported.

An optimal task allocation strategy with the goal of maximizing the reliability whilst considering communication cost as the constraint function for a distributed database management system has been reported by Verma [214]. The model is converted into a state space search tree and the Branch and Bound technique is used to achieve optimum results. This approach has been adopted for optimizing the distributed execution of join queries as reported by Reid [182]. Kartik and Murthy [133] used the idea of branch and bound technique and have presented efficient algorithms to maximize the system reliability in both redundant and non-redundant systems. Magirou and Milis [160] have presented an algorithm based on this technique for the problem of assignment of tasks to processors in a distributed processing system so that the sum of execution and communication cost is minimized.

Billionnet et al. [45] proposed another algorithm based on the BB technique to minimize the sum of inter-task communication and execution costs. In their approach the processors are heterogeneous while the communication links are identical. The use of identical links ensures that identical transmission times are generated when identical messages are transmitted through different links. The capacities of processors and links are assumed to be unlimited (uncapacitated network). The problem is first formulated to measure the inter-task communication and processing costs under an uncapacitated network. Then, the problem is refor-
Chang et al. [64] have proposed two algorithms based on the $BB$ technique to allocate files into a distributed system whilst minimizing the data transfer rate. The first algorithm called $OFA$ (Optimal File Allocation), uses the $BB$ technique with an evaluation function derived from critical cut concepts, and leads to an optimal solution. In second algorithm which is termed as $HFA$ (Heuristic File Allocation), the sub-optimal solution is obtained by terminating the $OFA$ search as soon as the first complete allocation is found.

Hagin [117] considered the allocation problem for assigning distributed multimedia applications into distributed computer systems. He has proposed an algorithm based on the $BB$ technique to solve two types of mapping problems.

### Integer Programming Techniques

Integer programming is a mathematical programming technique in which some of all the variables are restricted to be integers. It is directly applicable to the problem of task allocation in a distributed system. A task allocation model becomes more realistic when it incorporates real time constraints such as inter-processor communication, memory limitations of each processor etc. In the past, a significant number of studies have been devoted to the optimization in distributed systems using integer programming technique. For the task allocation problem, integer programming is a useful and exhaustive technique, as it is capable of reflecting real life situations of distributed processing and it is simple as well.

A number of researchers have worked on task allocation problem using integer programming technique to determine the optimal solution under given constraints [70][75][99]. A model based on this is developed by Chu [77] for optimum file allocation in a distributed system. A similar approach for data file allocation has also been proposed by Marcogliese and Novarese [163]. Lisper and Mellgren [152] have discussed various integer programming methods for the allocation in distributed real time systems. Using integer programming some scheduling tech-
4.3. Taxonomy of Task Allocation Models

Techniques have been developed by Tompkins [209] for task allocation in distributed systems.

Dynamic Programming Technique

Richard Bellman, first used the term dynamic programming to describe the solution procedure of the problems where one needs to find the best solution among a number of solutions. A dynamic programming approach examines the previously solved subproblems and combines their solutions to give the best solution for the given problem. It is both a mathematical optimization method and a computer programming technique. In terms of mathematical optimization, dynamic programming is a procedure to design algorithms for the problems wherein the solution is a result of a sequence of decisions.

A number of papers are available in literature proposing the solution to the task allocation problem using dynamic programming. Using this technique, Bokhari [47] formulated a shortest tree algorithm which runs in $O(mn^2)$ time for an optimal assignment of $m$ program modules to $n$ processors. Rosenthal [184] established a relationship between module allocation and non-serial dynamic programming. Dynamic programming technique was applied by Fernandez-Baca and Medepalli [108] for obtaining the optimal assignment through local search. Berman and Ashrafi [41] have developed optimization models for measuring the reliability of modular software systems using this technique.

Pros. and Cons. of Mathematical Programming Techniques

- These algorithms are more flexible compared to other allocation techniques.

- An optimal solution is guaranteed.

- System constraints like inter-task communication, network delay, memory limit, processor speed limit etc. can easily be formulated in terms of mathematical programming problem.
4.3. Taxonomy of Task Allocation Models

- These algorithms are time and space hungry. Their complexity grows exponentially as the parallel and distributed system is scaled up. So they are not suitable for large systems until some constraints are applied on them which in turn lead to randomized optimization.

4.3.4 Graph Theoretic Techniques

Graph theoretic approach is one of the most important mathematical techniques used in optimization. It has been extensively used in computer science research particularly in distributed systems, data mining and networking. This approach allows the use of graphical methods to represent and allocate program tasks to various processors in distributed processing system.

In most of the graph theoretical approaches, the solution begins with the abstraction of tasks and inter-task communication cost through graph model in which tasks are represented by nodes and inter-task communication cost as weights on bi-directional edges connecting these nodes. These approaches do consider load balancing, resource limitations etc. without giving much attention to the timing complexity. Graph theoretic approaches used in literature in the context of parallel and distributed systems can be further categorized as follows:

- Network Flow Techniques
- Shortest Path Techniques

Network Flow Techniques

In this approach the problem of allocating \( m \) tasks to \( n \) processors proceeds as follows:

- Denote the program by a graph consisting of a set of \( m \) nodes (tasks) and link these nodes with a set of edges where each edge represents communication among the tasks it is linking. The nodes of the graph are referred to as ordinary nodes.
4.3. Taxonomy of Task Allocation Models

- Convert the graph into commodity flow graph by adding \( n \) distinguished nodes in the graph, where each distinguished node corresponds to a processor.

- Connect every ordinary node with every distinguished node. The edge connecting ordinary node \( t_i \) with distinguished node \( P_s \) is given the weight \( w_{is} \), where \( w_{is} = \frac{1}{n} \sum_j e_{ij} - e_{is} \), for \( 1 \leq i \leq m \) and \( 1 \leq j \leq m \). Here \( e_{ik} \) denotes the execution cost of task \( t_i \) on processor \( P_k \).

- Apply an \( n \)-way cut on the graph. An \( n \)-way cut is defined as the set of edges which partition the graph into \( n \) disjoint subsets of nodes, each subset containing only one processor.

- Every subset obtained in previous step actually indicates an assignment. Sum of the weights on the edges in the cut is called the cost of the \( n \)-way cut and it is equal to the total sum of inter-task communication costs and execution costs \[153\].

Numerous techniques have been proposed in literature for solving the task allocation problem using network flow techniques. Stone \[202\] showed that the total sum of inter-task communication costs and execution costs can be minimized when program modules are assigned to a two processor distributed system. The complexity of this algorithm is \( O(m + 2)^3 \). He further extended his network flow techniques \[203\] and proved the existence of a critical load factor for each program task. However, this problem proved to be computationally intractable in the general case and thus for distributed systems.

To obtain a task allocation for an \( n \)-processor system where \( n \geq 2 \), the commodity flow graph is to be partitioned into \( n \) disjoint subsets of nodes, each subset containing only one processor. \( n \)-way partitioning is not possible with network flow algorithm alone. It has been attempted by repeatedly applying the network flow algorithm by Wu and Liu \[223\] \[224\] and also in combination with heuristics by Arora and Rana \[23\] for suboptimal solutions.
4.3. Taxonomy of Task Allocation Models

Rao et al. [181] again attempted to solve the same problem with an alteration in the system model within which one processor had limited memory while the other had unlimited memory. Using network flow techniques the authors demonstrated the method of constructing a $GH$-Graph, by finding the maximum flow between every pair of nodes of the original task interaction graph. In searching for a minimum cost assignment, they only produce the minimum cut and then reassign some subset of the tasks from one processor to another in order to satisfy the memory constraints. The algorithm had the complexity $O(n^3) + O(m - 1)$ for $m$ tasks and $n$ machines configuration. They further showed that the algorithm was $NP$-Hard for $n \geq 3$, so can’t be applied to distributed system allocation problem in general case.

Lee et al. [146] also considered the same problem using network flow techniques. They extended Ston’s work and generalized it from a two processor to a network of $n$ processors connected as a linear array. In their approach, the task allocation problem is first converted into a two terminal network flow problem and then solved using network flow technique in $O(n^2m^3 \log n)$ time. Again Lee and Shin [145] considered the allocation problem on a network of $n$-homogeneous processors. Each of them had its own memory. They first developed a modeling technique that transformed the assignment problem in a tree into a minimum-cut maximum flow problem and then solved the problem in $O(\sum_i(n_i - 1)m^3)$ time.

The allocation problem has also been considered by Hui and Chanson [124] for assigning $n$ tasks without precedence constraints (usually represented by a $TIG$) to a set of $m$ homogeneous and heterogeneous system connected through shared media, assuming access time to the shared media as zero. Initially they presented an algorithm for homogeneous system which had the complexity $O(n^2(n + E)\log \frac{n^2}{n + E})$. They then extended the algorithm for heterogeneous processors and presented an algorithm with complexity $O((n + E)n \log n)$.
4.3. Taxonomy of Task Allocation Models

Shortest Path Techniques

In a distributed processing system the task allocation problem can also be solved using the shortest path techniques. This approach has been widely investigated by the several researchers [47] [178] [211] to obtain an optimal solution.

The shortest path algorithm evaluates the set of nodes in the assignment graph corresponding to a program graph to select the best possible allocation of task to a processor. A shortest tree algorithm described by Bokhari [47] minimizes the total sum of the execution and communication costs for arbitrarily connected distributed systems with arbitrary number of processors, provided that the interconnection pattern of modules form a tree. The timing complexity of this algorithm is $O(mn^2)$.

Price and Pooch [178] claim that their shortest path method is applicable to all cases but, an optimal solution is possible in certain cases only. The shortest path algorithm proposed by them evaluates the set of nodes in the assignment graph corresponding to a program graph to select the best possible allocation of a task to a processor. A modification to the shortest path method is made using the non-backtracking branch and bound method and the complexity of algorithm is known to be roughly $O(mn)$.

The critical path method [91] and the ideas from the renewal theory along with the theory of large deviations [140] are also used for designing shortest path task allocation techniques.

Towsley [211] work considers special cases where the inter-task communication pattern is series parallel in nature. A search is made in the assignment graph for the parts of the program graph where inter-task communication patterns are series parallel or tree in nature. For such inter-task communication patterns, the shortest paths are derived and these shortest paths are combined to get the overall shortest path of the assignment graph. The timing complexity of the algorithm developed on this principle is $O(mn^3)$.

The problem of assigning the modules of chain structured programs and tree structured programs is presented in [48] [127] [138] [49] [128]. These programs
are considered for allocation on chain structured computer systems and single host, multiple satellite computer systems.

To derive an optimal allocation, an assignment graph is constructed which contains as many layers as the number of processors. The calculation of weights of edges depends on the nature of the program graph and the processor system. Each edge of the assignment graph is given two weights: sum weight and bottleneck weight. The sum and bottleneck weights of the path from source to terminal node indicate the time required by corresponding assignment. The optimal path should have the minimum sum bottleneck weight.

**Pros. and Cons. of Graph Theoretic Allocation Methods**

- These methods have been widely researched due to their simplicity.
- They are not meant for a generalized allocation of \( m \) tasks to \( n \) processors since such configurations have exponentially high complexities.
- Formulating resource constraints in graph theoretic approaches is a challenge.

**4.3.5 State Space Search Techniques**

State space search is a well-known approach to achieve optimal cost of the task allocation in a distributed system. In this approach, first the task allocation problem is converted in terms of a state space search tree and then a cost function is defined which is then used to guide the search. The search space tree is drawn as follows. For the task allocation in a distributed processing system, each state description is denoted by a node. Operators applicable to nodes are defined for generating successors of the nodes called node expansions. A solution path of a search problem is the path defined by a sequence of operators which leads a start node (i.e. initial node) to one of the goal nodes (i.e leaf nodes or external nodes). All the internal nodes in this state space search tree correspond to incomplete task allocations and all external (leaf) nodes correspond to complete allocations \[193\].
In this search tree, the job is to find the goal node i.e. a leaf node corresponding to the optimal task allocation. The well known $A^*$-algorithm is known to be the best tool for searching optimal assignment in state space tree.

A number of researchers have used the $A^*$-Algorithm to find the optimal allocations in distributed systems. Shen and Tasi [194] proposed the first $A^*$ based solution. They first translated the task assignment problem into a special graph known as weak homomorphism in graph matching. Then on this weak homomorphism they applied the $A^*$-algorithm to find the optimal solution. However, their solution was not designed for a generalized parallel and distributed system. They considered a point-to-point interconnection network in which tasks with inter-task communications were required to reside either on the same processor or on two directly connected processors.

Sinclair [199] has developed a method to reduce the size of the search tree. This method is termed as the reduction method. Two criteria proposed by the authors to implement the reduction method are known as Minimum Processor Cost Underestimate (MPCU) function and Minimum Independent Assignment Cost Underestimate (MIACU) function. The allocation algorithm developed based on reduction method is called "Branch and Bound with Underestimate (BBU)". BBU attempts to minimize the total sum of execution costs and inter-task communication costs while allocating the program modules to processors of a distributed system. The authors further proved that Minimum Independent Assignment Cost Underestimate when used with $A^*$, leads to less space and timing complexities. The solution proposed by Sinclair [199] does not take into account the task precedence constraints. The method was further improved by Wang and Tasi [218], by considering task precedence constraints. Tom and Murthy [208] improved the models proposed by Shen and Tasi [194] and Wang and Tasi [218]. They altered the order of nodes in the search tree in such a way that the independent tasks were restricted to be assigned last.

Kafil and Ahmed [132] have proposed a two phase algorithm for finding optimal solution of task allocation problem using $A^*$. The first phase of the algo-
4.3. Taxonomy of Task Allocation Models

Algorithm is known as “Optimal Assignment Sequential Search (OASS)” and is meant to produce an initial solution. The purpose of this phase is to reduce all those nodes which have higher costs than the resulting solution cost during search operation. The second phase of the algorithm is known as “Optimal Assignment Parallel Search (OAPS)” and its purpose is to accelerate the search process.

**Pros. and Cons. of State Space Search Techniques**

- They are flexible compared to their graph theoretic counterparts.
- System constraints like inter-task communication costs, network delay etc. are easy to formulate using these methods.
- For the task allocation problem an optimal solution is guaranteed by $A^*$.
- Like mathematical programming technique, these techniques are also constrained by the time and space complexities and, algorithms based on them require searching for an optimal solution.

4.3.6 **Heuristic Techniques**

These techniques have been developed to obtain the near optimal solution of task allocation problem in acceptable time. Although an optimal solution to task allocation problem can be obtained using exact algorithm, the general $n$-processor task allocation problem is $NP$-Complete. Therefore, finding an optimal solution with exact algorithm to large scaled task allocation problems is computationally prohibitive [132] [212] [233]. Therefore, the development of effective heuristics is gaining importance among researchers. Heuristics provide fast and effective alternatives for obtaining near optimal solution of large scaled task allocation problems. During the past two decades numerous heuristics have been reported in literature for the task allocation problem. They may be further classified as follows:

- Clustering Heuristics
4.3. Taxonomy of Task Allocation Models

- Greedy Heuristics

- Iterative Heuristics.

In the following subsection we briefly discuss some popular heuristics from each category.

**Clustering Based Heuristics**

Much of the focus in heuristic algorithms has been devoted to the minimization of inter-task communication cost. Task clustering technique is one of the heuristic approaches to reduce the total inter-task communication cost. In this technique, a set of communicating tasks is fused together to form a task cluster. If the number of created clusters is greater than the number of available processors then clusters are fused in such a manner that the number of clusters becomes equal to the number of processor. It may drastically reduce the size of the search space [216]. Finally, task clusters are allocated to the available processors.

Efe [96] presented a task-clustering algorithm called “two module clustering” that forces task pairs to be allocated to the same processor. This procedure is run until all the candidate task pairs are grouped together. Arora and Rana [22] proposed module assignment in two-processor distributed system. The authors later [24] extended the idea of task clustering and proposed the concept of clustering the tasks which exhibited certain particular behaviour to reduce the problem size. Sagar and Sarje [188] used the clustering technique to propose other models for task allocation in distributed systems. Bowen et al. [55] proposed a clustering algorithm for assignment problem of arbitrary process systems to heterogeneous distributed computing system.

The clustering technique used by Kim and Browne [135] iteratively applies a critical path algorithm to transform the graph into a virtual architecture graph which consists of a set of linear clusters and the interconnection between them. A different heuristic approach based on the concept of clustering to allocate the tasks for maximizing reliability has been proposed by Srinivasan and Jha [201].
4.3. Taxonomy of Task Allocation Models

Based on the clustering approach, a multiprocessor scheduling technique named “de-clustering” has been formulated by Sih and Lee [198]. The authors claimed that their de-clustering approach not only retains the clustering advantages but at the same time overcomes its drawbacks. Palis [171] presented task clustering and scheduling algorithm for distributed memory parallel architecture.

Abdelzaher and Shin [3] developed a graph-based model which involved recursive invocation of two stages: clustering and assignment. The clustering stage partitions tasks and processors into clusters while the assignment stage maps task clusters to processor clusters. The problem of minimizing the cost by clustering heavily communicating tasks and assigning the clustered tasks to appropriate processors has also been researched by [221] [177] [219].

Greedy Heuristics

A partial solution is essential to initiate a greedy heuristic. These approaches begin with a partial solution and repeatedly improve this solution until the complete allocation is done. At each stage one task allocation is made and this decision remains unchanged throughout all the subsequent stages. The allocation is done in a manner that a choice once made can never be reconsidered i.e. without any backtracking. The allocation of next task to be allocated say, \( \tau_k \) depends on the criteria chosen to allocate the previous \( k - 1 \) tasks. Generally these approaches are easy to implement and lead to near optimal solution in polynomial time, less than \( O(m^3) \), for \( m \) tasks in most of the cases.

Numerous solutions for task allocation problem using greedy approaches have been reported in literature [153] [144] [120] [162] [126]. Lo [153] merged a greedy algorithm with Stone [202] and thereby minimized the total execution and inter-task communication costs. The algorithm proposed by the authors consists of three phases namely Grub, Lump and Greedy. The first phase Grub, produces the initial partial solution required to run the greedy heuristics. The second phase Lump, is executed when the allocation is not completed by Grub. Similarly, the third phase is executed when the allocation is not completed by the second phase.
Another parameter used by this approach is interference cost. It is the measure of the amount of incompatibility among tasks.

The approach used by Lee [144] implements load balancing aspect through the greedy heuristic. The algorithm is termed as Largest Processing Time First (LPFT). The tasks are first arranged in descending order of communication cost and then allocated to the less loaded processors as per the order already defined.

Hluchy et al. [120] proposed static and dynamic greedy heuristics for multi-computer systems. An objective function was formulated by the authors to check the optimality of the solution in a static heuristic approach. A semi-distributed approach has been developed for dynamic allocations. For distributing programs and data files to networked multicomputers whilst maximizing the system reliability Hwang and Tseng [126] have proposed greedy heuristics based algorithms.

**Iterative Heuristics**

Like greedy heuristics, these approaches also start with an initial allocation and try to refine it further in subsequent steps. The initial allocation may be obtained using some faster heuristic or some random optimization technique. However, to improve the initial allocation they use techniques such as task migration or pairwise tasks exchange, which are generally not used by greedy approaches.

The following subsection presents some popular iterative heuristics found in literature:

**Genetic Algorithms (GA)**

It is a meta-heuristic optimization technique based on Darwin’s evolutionary theory of the survival of the fittest. It emulates the behaviour of reproduction in nature [215] [212]. A genetic algorithm can be applied to search large multi-objective and complex problem spaces in a distributed system. Genetic algorithms attempt to search a near optimal solution by fostering a population of strings (called chromosomes) using predefined genetic operators [151]. The main steps of a genetic algorithm are reproduction, selection, crossover and mutation. The
4.3. Taxonomy of Task Allocation Models

selection, crossover and mutation processes are repeated until the termination condition is satisfied [151].

In the last two decades a lot of work on the application of GA on a distributed processing system has been done by the researchers which is mostly related to the task allocation problem [12] [10] [212] [215] [123] [122] [231] [227] and task scheduling problem [222] [85] [167]. Task allocation in a distributed system is a challenging issue as it requires the system performance to be optimized. A technique based on problem-space genetic algorithm (PSGA) for static task allocation in heterogeneous distributed system has been proposed by Ahmad et al. [12], which combines the power of GA with the problem specific heuristic to search the best possible solution. For multiple task allocation, Tripathi et al. [212] developed a GA based method which is memory efficient and gave an optimal solution of the problem. Vidyarthi et al. [215] also used GA to maximize the reliability of a distributed computing system.

In distributed computing systems the hardware redundancy policy is of equal importance as the task allocation policy because it has direct impact on system cost and system reliability. In this context, researchers have addressed some algorithms based on GA [122] [227]. Levitin et al. [149] discussed a redundancy optimization problem for multistate systems. Deeter and Smith [89] used the similar approach to solve the all-terminal network design problem considering cost and reliability.

Applications of GA have revealed that they generate more efficient solutions than other heuristic approaches, which are applied to task scheduling in heterogeneous system. Page et al. [170] presented a GA based scheduling strategy to dynamically schedule a set of heterogeneous task onto a set of heterogeneous processors to minimize the total execution time.

Hill-Climbing (HC)

In the field of computer science, Hill-Climbing is an iterative technique for solving computationally hard problems. It starts with a sub-optimal solution to the
problem (i.e., start at the base of a hill) and iteratively improves the solution (climb up the hill) until some criterion is maximized (top of the hill is reached). The improvement to the current state (solution) is made gradually in small steps. Associated with each random move is a cost function which is evaluated after each step. If the change in the cost function is positive, the move is accepted and a new solution is generated; otherwise, no change is made to the current state. The process is repeated until there are no changes to the current state which lead to the reduction in the value of the cost function. When this condition occurs, it indicates that a local optimum has reached, instead of the required global optimum.

The problem with this algorithm is that it guarantees an optimal solution in case of convex problems and it is considered to be a good algorithm for finding a local optimum (a solution that can’t be improved by considering a neighbouring state). However, it doesn’t guarantee to find the best possible solution (the global optimum) out of all possible solutions (the search space).

Yanping and Haijiang [229] have developed a method of allocating tasks to welding robots using the hill climbing algorithm. Fattah et al. [105] have used the hill climbing techniques to rapidly find the appropriate start node in the application mapping of network-based many-core systems.

**Simulated Annealing (SA)**

Simulated Annealing is a global optimization technique which attempts to find the lowest point in an energy landscape. It emulates the physical concepts of temperature and energy to present and solve the optimization problems [30]. The objective function of the optimization problem is treated as the energy of a dynamic system while the temperature is introduced to randomize the search for a solution.

Over past few years, SA has been used by many researchers for solving the task allocation problem in a distributed system. In 2004, Attiya and Hamam [27] proposed a simulated annealing based optimal two-phase algorithm for task allocation problem in a heterogeneous distributed computing system with the goal of
maximizing the system reliability. The authors also proposed [30] another simulated annealing based optimal algorithm for the same problem and with same goal in 2006. Their hybrid algorithm first finds a sub-optimal allocation by applying the well known $SA$ algorithm and then finds an optimal allocation by applying the branch and bound ($BB$) technique by considering the solution provided by $SA$, as the initial solution. Further they used the same algorithm for load balancing in heterogeneous distributed computing system [29].

With the goal of minimizing the inter-task communication delays, Bollinger and Midkiff [52] have proposed a $SA$ based task allocation technique for a multi-computer network system. Hamam and Hindi [118] have used a similar approach for allocating program modules to a distributed system connected through a general purpose interconnection network.

**Mean Field Annealing ($MFA$)**

The $MFA$ technique was developed to solve combinatorial optimization problems. In this approach discrete variables, termed as *spins*, are used for encoding the combinatorial optimization problem. An *energy function* written in terms of *spins* is used as cost function. Then, using the expected values of *spins*, a gradient descent type relaxation scheme is used to find a configuration of the *spins* which minimizes the cost function.

Based on $MFA$ approach, Bultan and Aykanat [59] have developed a heuristic approach to solve the task allocation problem. They have shown that, the algorithms based on $MFA$ are faster than those based on the $SA$ but the later produces better solutions than the former.

Aykanat and Haritaoglu [34] have used the $MFA$ techniques mapping unstructured domain to a hypercube connected distributed memory architecture. Their goal is to find a mapping which minimizes the communication overhead while maintaining the same workload on processors (load balancing).
4.3. Taxonomy of Task Allocation Models

Other Heuristic Approaches

Heuristic approaches discussed above are not the only ones but are the most present in literature. [66] [191] [72] [61] [139] [18] describe some other heuristic approaches.

Chaudhary and Aggarwal [66] proposed a general algorithm for allocating a Task Interaction Graph (TIG) as well as a Directed Acyclic Graph (DAG) on a distributed system. The algorithm first finds an initial assignment and then improves it by pairwise exchange. The algorithm is very general so the complexity is high.

Selvakumar and Murthy [191] have proposed a heuristic algorithm for allocating programs into a distributed system with heterogeneous machines. An algorithm based on recursive divide-and-conquer was first developed for the case of two machines and then extended to the case of n machines.

Choi et al. [72] proposed three heuristic algorithms for allocating tasks of a linear task graph into a parallel system of heterogeneous processors. The complexities of first and second algorithms are $O(n^2m)$ and $O(n^4m^4)$ respectively. The third algorithm is based on greedy heuristic.

Cai and Lee [61] have proposed a three phase algorithm to distribute data files among heterogeneous server machines. The goal is to balance the response time between the servers i.e., minimize the difference in the response time. The first phase, selection phase, selects the set of servers that can participate in file allocation by eliminating those servers that have limit lower than a predefined value. The second phase, allocation phase, sorts the selected servers in increasing orders of their service rates, allocates files with higher access rate to a server until reaching its allocation limits and then the allocation moves onto a next server. The third phase, completion phase, allocates the remaining files into servers with higher capacity until all files are assigned.

Koziris et al. [139] presented a two phase heuristic algorithm to map tasks of a TIG into multiprocessor architecture to minimize the communication time. The algorithm first assigns the highly communicating tasks on adjacent nodes in
the processor's network. Then, without backtracking it maps the remaining tasks beginning from those close to the assigned tasks.

Altenbernd and Hansson [18] proposed an algorithm which is termed as slack method. This heuristic allocates communicating periodic hard real-time tasks onto a multiprocessor system.

**Pros. and Cons. of Heuristic Techniques**

- Heuristics provide efficient and effective methods of obtaining near optimal solutions.
- Their time and space complexities are less compared to exact techniques.
- It is very difficult to choose best values for the parameters required by them.
- Sometimes, they lead to a suboptimal solution that may be far away from the exact solution.
- Braun et al. [56] have proved that if they are used to get a solution close to optimal solution, their timing complexity may increase exponentially. They have further concluded that it is almost impossible to select a single heuristic technique as the best general method of solving different types of applications.

**Chapter Summary**

In this chapter a precise introduction to the task allocation problem is presented. Relation between task allocation and static load balancing schemes is explained. In literature survey various techniques available for task allocation problem are classified based on the solution techniques used: mathematical programming, graph theoretic, state space search and heuristics. It has been established that graph theoretic schemes are simple to implement and are acceptable as long as the number of processing elements in a distributed processing system are less than four. They
can not be used for the generalized problem of allocating $m$ tasks to $n$ machines due to their high computational time and inability of handling various system constraints. The mathematical programming and state space search techniques are found to be more suitable compared to graph theoretic techniques for the generalized problem of task allocation. They lead to an exact solution and can easily handle the system constraints but once again their time and space complexities grow exponentially as the system is scaled up. Heuristic approaches unlike the other three, are simpler and flexible. They are suitable for the applications where an exact solution is not possible in a reasonable time. However, solutions obtained by them may be far away from the exact solution. Their timing complexity may be problematic if a solution close to the exact solution is required. Moreover, it is not easy to find the best values for the parameters required by them and hence none of the heuristics can be designated as good approach for solving the task allocation problem.

The focus of next chapter is our contribution to the task allocation problem considering load balancing.