CHAPTER 1

INTRODUCTION TO TASK SCHEDULING

1.1 INTRODUCTION

Parallelism arises frequently in everyday life. Juggling is a parallel task that humans can perform. Humans are made up of various parts like ears, eyes, hand, legs, brain, etc. Eyes see, ears hear, skin feels, hands work, and all these jobs can take place simultaneously without affecting one another. They are coordinated and organized by the brain. In human beings, the brain acts as the center for parallelism and initiates the various functions of the respective organs. Parallel computing is an evolution of serial computing that attempts to emulate what has always been the state of affairs in the natural world. Many complex, interrelated events are happening at the same time, yet within a sequence. Examples include galaxy formation, planetary movement, weather and ocean patterns, rush hour traffic, automobile assembly line, building a space shuttle and the like. The Data Warehouse researchers compare parallel computing to building a house. The house corresponds to the problem to be solved and the workers are the CPUs. There are many different tasks involved in building a house, and to get the job done efficiently, the workers must work on the separate tasks in a proper order. In general, the Real World is massively parallel. Historically, parallel computing has been considered to be “the high end of computing” (Blaise Barney 2010).

Increasing the computational capabilities is essential for the continued progress of modern Science and Technology in all fields. Parallel computing has been recognized as one of the fundamental approaches to achieve high-performance computing and it values short execution time. It is
the simultaneous use of multiple computer resources to solve a computational problem. Here the problem is broken down into discrete parts that can be solved concurrently and each part is further broken down to a series of instructions. Instructions from each part are executed simultaneously on different CPUs.

In the past, only a limited number of government laboratories and large businesses had access to high-performance parallel systems, primarily because of the tremendous cost of these systems, both to purchase and to operate.

Nowadays, with the rapid growth of microprocessor power, reduction in memory prices, emergence of high-speed networks, and availability of parallel system and application software, building and maintaining parallel systems have become cost effective and thus are attractive and accessible to a considerably wider range of users. Improving the performance of parallel systems has been a major goal since the inception of parallel processing.

1.2 PARALLEL COMPUTING

Parallel computing is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently (“in parallel”). It is the simultaneous execution of the same task (split up and specially adapted) on multiple processors or multiple computers in order to obtain results faster.

Many scientific problems exceed the capabilities of the single processor. In such cases, parallelism is the only means by which the large
computational runs can be performed. Parallelism can be achieved through the following ways:

- An Application is to be run using multiple CPUs.
- A problem is broken into discrete parts that can be solved concurrently.
- Each part of the problem is further broken down to a series of instructions.
- Instructions from each part are executed simultaneously on different CPU's.

An essential aspect of parallel programming is scheduling which is the spatial and temporal assignment of the tasks of a program to the processors of the target system. For the efficient utilization of a parallel system, a task must be divided into sub-tasks and these must be assigned to the system’s resources. This assignment of tasks to the computing and communication resources of a parallel system is known as the scheduling or task scheduling problem.

In the last few years, the trends in parallel processing system design and deployment have been moving away from single isolated powerful supercomputers to cooperative networked distributed systems such as commodity-based **cluster computing** systems and **grid computing**. When a large number of computers are to be shared by a variety of users, proper scheduling and resource allocation becomes a critical issue. The system may consist of powerful machines, interconnected by a high-speed network. However, the poor choice of a scheduling algorithm will lead to poor system performance and utilization.
1.3 CLUSTER COMPUTING ENVIRONMENT

A cluster is a group of linked computers, working together closely so that in many respects they form a single computer. The components of a cluster are commonly connected to each other through fast local area networks. They are usually deployed to improve performance and/or availability over that provided by a single computer. The key to making cluster computing work well is the middleware technologies that can manage the policies, protocols, networks, and job scheduling across the interconnected set of computing resources.

Cluster computing (Rajkumar Buyya and Mark Baker 1999) has emerged in recent years as one of the major growth areas in applied computer science. The traditional model of national HPC facilities is also being eroded by cluster computing as research groups find more value in a flexible manner. Approaches for managing scheduling, reliability and interoperability on clusters often originated in the distributed computing community.

The cluster computing systems can be a single cluster system or multicluster systems. A single cluster system is formed from a set of independent workstations that are interconnected by a Local Area Network (LAN). Multicluster computing systems are formed from a set of independent clusters interconnected by a Wide Area Network (WAN). Job scheduling is a complex problem, yet it is fundamental to sustaining and improving the performance of parallel processing systems. However, multicluster systems, compared to the classical parallel computers, pose several technical challenges that introduce an additional degree of complexity to the scheduling problem while amplifying the existing ones.

For example, resources in cluster computing are: (1) distributed (2) heterogeneous and (3) highly shared in both time and space. Therefore,
continuously arriving jobs and dynamically changing available CPU capacity make traditional scheduling algorithms difficult to apply in clustered systems. Therefore, it is believed that effective scheduling heuristics promise better results by adapting schedules to changing situations.

1.4 PARALLEL PROGRAMMING PARADIGMS

A few programming paradigms are used repeatedly to develop many parallel programs. Each paradigm is a class of algorithms that have the same control structure. The choice of paradigm is determined by the available parallel computing resources and by the type of parallelism inherent in the problem. The computing resources may define the level of granularity that can be efficiently supported on the system. The type of parallelism reflects the structure of either the application or the data and both types may exist in different parts of the same application. Parallelism arising from the structure of the application is named functional parallelism. In this case, different parts of the program can perform different tasks in a concurrent and cooperative manner. But, parallelism may also be found in the structure of the data. This type of parallelism allows the execution of parallel processes with identical operation but on different parts of the data.

1.4.1 Choice of Paradigms

Most of the typically distributed computing applications are based on the very popular client/server paradigm. In this paradigm, the processes usually communicate through Remote Procedure Calls (RPCs), but there is no inherent parallelism in this sort of applications. They are, instead, used to support distributed services, and it is not considered in the parallel computing area. The popular parallel programming paradigms are Task-Farming, Single-Program Multiple-Data, Data Pipelining, Divide and Conquer and Speculative Parallelism. Among the parallel programming paradigms, the
Task-Farming, Single-Program Multiple-Data, Divide and Conquer and Work Pool are used in this proposed research work.

1.4.1.1 Task-Farming (or Master/Slave)

The Task-Farming paradigm consists of two entities namely, master and multiple slaves. The master is responsible for decomposing the problem into small tasks (and for distributing these tasks among a farm of slave processors), as well as for gathering the partial results in order to produce the final result of the computation. The slave processors work in a very simple cycle: get a message with the task, process the task, and send the result to the master. Usually, the communication takes place only between the master and the slaves.

Task-Farming may either use static load balancing or dynamic load balancing. In the first case, the distribution of tasks is all performed at the beginning of the computation, which allows the master to participate in the computation after each slave has been allocated a fraction of the work. The allocation of tasks can be done once or in a cyclic way. Figure 1.1 presents a schematic representation of this first approach.

The other way is to use a dynamically load balanced Master/Slave paradigm, which can be more suitable when the number of tasks exceeds the number of available processors, or when the number of tasks is unknown at the start of the application, or when the execution times are not predictable, or when one may be dealing with unbalanced problems. An important feature of dynamic load balancing is the ability of an application to adapt itself to changing conditions of the system, not just the load of the processors, but also a possible reconfiguration of the system resources.
Due to the reconfiguration of the system resources characteristic, the Task-Farming paradigm can respond quite well to the failure of some processors, which simplifies the creation of robust applications that are capable of surviving the loss of slaves or even the master. The Task-Farming paradigm is widely used in the present, proposed heuristics research work.

1.4.1.2 Single-Program Multiple-Data (SPMD)

The SPMD paradigm is the most commonly used paradigm. Each process executes basically the same piece of code but on a different part of the data. This involves the splitting of application data among the available processors. This type of parallelism is also referred to as geometric parallelism, domain decomposition, or data parallelism. Figure 1.2 presents a schematic representation of this paradigm.
Many physical problems have an underlying regular geometric structure, with spatially limited interactions. This homogeneity allows the data to be distributed uniformly across the processors, where each one will be responsible for a defined spatial area. Processors communicate with neighbouring processors and the communication load will be proportional to the size of the boundary of the element, while the computation load will be proportional to the volume of the element. It may also be necessary to perform some global synchronization periodically among all the processes. The communication pattern is usually highly structured and extremely predictable. The data may initially be self-generated by each process or may be read from the disk during the initialization stage.

1.4.1.3 Divide and Conquer

The Divide and Conquer approach is well known in sequential algorithm development. A problem is divided into two or more sub problems. Each of these sub problems is solved independently and their results are combined to give the final result. Often, the smaller problems are just smaller
instances of the original problem, giving rise to a recursive solution. Processing may be required to divide the original problem or to combine the results of the sub problems. In parallel divide and conquer, the sub problems can be solved at the same time, given sufficient parallelism. The splitting and recombining process also makes use of some parallelism, but these operations require some process communication. However, because the sub problems are independent, no communication is necessary between processes working on different sub problems.

One can identify three generic computational operations for divide and conquer and they are split, compute, and join. The application is organized in a sort of virtual tree. Some of the processes create subtasks and have to combine the results of those to produce an aggregate result. The tasks are actually computed by the compute processes at the leaf nodes of the virtual tree. Figure 1.3 presents this execution.

The Task-Farming paradigm can be seen as a slightly modified, degenerated form of Divide and Conquer; i.e., where problem decomposition is performed before tasks are submitted, the split and join operations are only done by the master process and all the other processes are only responsible for the computation.

In the Divide and Conquer model, tasks may be generated during runtime and may be added to a single job queue on the manager processor or distributed through several job queues across the system.
1.4.1.4 Work Pool

This paradigm is used in the shared variable model. A pool of work is released in the global data structure shown in Figure 1.4. A number of processes are created, but initially there can be only a single piece of work in the pool. Any free process fetches the work from the pool and executes the same. It also can produce one or more work pieces into the pool. The program ends when the work pool becomes empty. The Resource Aware Scheduler developed in the present research work makes use of the Work Pool paradigm for processing free processor requests.
1.4.2 Description of the problem

Task scheduling is to find a mapping of the set of m tasks to n processors such that the total completion time is minimized. There are $n^m$ possible assignments, and finding the optimal assignment after exhaustive search is known to be a NP – Complete problem. The objective of task scheduling problem is to minimize the schedule length, which is given by

$$\text{Schedule Length} = \max (\text{AFT} (V_{\text{exit}}))$$

where
\begin{align*}
\text{AFT} &\quad \text{Actual Finish Time} \\
V_{\text{exit}} &\quad \text{Exit Task}
\end{align*}

There are huge numbers of heuristics for finding optimal makespan. It is necessary to propose metaheuristics approaches for the task scheduling problem since it avoids local minima and guide the search towards global optima. Hence, in this work, various heuristics have been applied to the task scheduling problem for finding an optimal makespan considering the non-preemptive and preemptive tasks with precedence constraint in cluster of workstations.

1.5 LITERATURE SURVEY

Task scheduling can be performed at compile-time or run-time. It has been proven to be NP-Complete (Haluk et al 2002) for which the optimal solutions can be found only after an exhaustive search.

As finding an optimal solution is not feasible, a large number of algorithms are proposed which attempt to obtain a near-optimal solution for many variants of the multiprocessor task scheduling problem. Hence, a need arises for developing an efficient algorithm for scheduling highly
communication intensive tasks on the available parallel system i.e., Cluster Of Workstations. The general approaches for task scheduling are shown in Figure 1.5.

**Figure 1.5  Simplified Taxonomy of the Task Scheduling Problem Approaches**

1.5.1  Related Works in Classical List Scheduling Algorithm

A number of works in the literature deal with the classical list scheduling heuristics.

Tomasz Kalinowski et al (2000) proposed a list scheduling of general task graphs under LogP model. They analyzed the performance bound of the proposed approaches. However, the problem encountered during construction of program schedules for LogP model is the proper insertion of
receive and send tasks. The proposed algorithm was found to reduce the number of possible task assignments simplifying the algorithms but possibly eliminated some good solutions and also the communication time was not found to be greater than the average computation time. It is strongly recommended to use this approach when the number of processors is smaller than the graph parallelism and the number of task successors is low.

Haluk TopCuoglu et al (2002) presented two novel scheduling algorithms for a bounded number of heterogeneous processors with an objective of meeting high performance and a fast scheduling time which are called Heterogeneous Earliest Finish Time (HEFT) and Critical Path On Processor (CPOP). They observed that the CPOP algorithm resulted in a better performance and running time results than the other existing algorithms. A new method has been introduced in this paper for the processor selection phase with an effort to minimize the earliest finish time of the critical-child task of each selected task. Their policies for the task prioritizing and processor selection phases are adopted in the present research work. Moreover, their work has been extended and enhanced in the proposed approach of the present work for contention aware task scheduling.

Bajaj and Agrawal (2004) developed a task duplication based scheduling algorithm for network of heterogeneous systems (TANH). This algorithm proved to be optimal for DAGs by reducing the makespan and its performance has been observed to be better than the scheme of the BIL (i.e., the best existing scheduling algorithm for heterogeneous systems).

Ishfaq Ahmad et al (2004) evaluated and compared the algorithms for scheduling and clustering. These algorithms allocate a parallel program represented by an edge-weighted Directed Acyclic Graph (DAG), also called a task graph or macro-dataflow graph, to a set of homogeneous processors, to minimize the completion time. They examined several classes of such
algorithms and compared the performance of a class of algorithms known as the Arbitrary Processor Network (APN) scheduling algorithms. Their discussion included the design philosophies and principles behind these algorithms and assessed their merits and deficiencies. Experimental results had been obtained by testing the algorithms with a large number of test cases. Global and pair-wise comparisons were made within each group whereby these algorithms were found to be ranked according to their performance.

Illavarasan and Thambidurai (2005) proposed a list scheduling heuristics for scheduling DAG structured applications onto a heterogeneous computing system. The algorithm proved to perform better than the existing methods in terms of the average schedule length ratio, speedup, efficiency, running time and the frequency of the best results. The performance of this algorithm was observed experimentally by using a large set of randomly generated task graphs with various characteristics. The complexity of the algorithm was less when compared with other scheduling algorithms. This work can be extended for arbitrary connected networks.

Comparing with all these works, the proposed list scheduling algorithm in this thesis is different in many aspects. First, the task scheduling heuristics is based on insertion-based policy and making use of priority queue based calculation for priority. Then the tasks are scheduled based on the minimum EFT value of the processors. Moreover, this scheduling heuristic can be scaled to any number of the nodes in the cluster. This heuristic is used for scheduling any high communication intensive applications.

1.5.2 Works on Hybrid PSO Approaches for Task Scheduling

A lot of research is being carried out for the hybrid PSO approaches for the task scheduling problem.
1.5.2.1 Related Works in PSO for Task Scheduling

Tingwei Chen et al (2006) proposed a task scheduling mechanism, which expressed each possible task scheduling scheme as a Task-Resource Assignment Graph (T-RAG) and thus mapped the task scheduling problem onto a graph optimal selection problem. A task scheduling algorithm based on Particle Swarm Optimization (PSO) was proposed which considered the longest path of the T-RAG as fitness value and encodes every task-resource assignment as a particle. The automatic adjustment during run-time, a problem which was not addressed in the dynamic property of grid resources. Their experimental results indicated that the proposed PSO-based algorithm could be a viable alternative for solving the grid scheduling problem.

1.5.2.2 Related Works in Tabu Search Heuristics for Task Scheduling

Mahmood et al (2002) investigated a Tabu Search algorithm for non-preemptive static scheduling of real time tasks where the tasks could be periodic and have arbitrary deadlines, precedence and exclusive constraints in a distributed computing environment.

Graham Ritchie (2003) described several novel approaches using the Ant Colony Optimization (ACO). Metaheuristic and local search techniques including Tabu Search to the problem of static scheduling of independent jobs on to homogenous and heterogeneous processors were proposed but were found to be significantly longer to run.

Adil Bokasoglu (2006) discussed applying the multi objective Tabu Search to continuous optimization problems with a simple neighborhood strategy. In this paper, the authors suggested that there was no need to apply a weighting scheme, the objective functions were not combined into a single function but by using some kind of utility functions. The advantage of this
approach could be found in the guidelines given for the determination of the Tabu Search parameters which is adapted in the present research work.

Gabriella Stecco et al (2009) introduced a Tabu Search heuristic for a production scheduling problem with a sequence dependant and time dependant set-up times on a single machine. The two neighborhoods defined by the swap and complete relocate moves were evaluated for an appropriate surrogate objective function to identify a set of promising solutions.

1.5.2.3 Related Works in Hybrid Scheduling on Distributed Systems

Peng-Yeng Yin et al (2006) proposed a hybrid PSO algorithm for finding a near-optimal task allocation within reasonable time. Experimental results showed that the HPSO was found to be robust against different problem sizes, task interaction density and network topology besides maximizing the Distributed System Reliability (DSR) of executing successfully a task consisting of several modules.

Shinichi Nakano et al (2007) presented a new Particle Swarm Optimization based on the concept of Tabu Search, TS-PSO. The proposed method was validated through numerical simulations with several functions which are well known as optimization benchmark problems comparing to the conventional PSO methods. Constraint conditions were not found covered under this method.

Premalatha and Natarajan (2009) presented the hybrid approaches of Particle Swarm Optimization (PSO) with Genetic Algorithm (GA). PSO and GA are population based heuristic search techniques which can be used to solve the optimization problems modeled on the concept of the Evolutionary Approach. The modification strategies were proposed in the PSO using the
GA. Experiment results were examined with benchmark functions and the results showed that the proposed hybrid models outperform the standard PSO.

1.5.2.4 Related Works in PSO and Tabu Search for Task Scheduling

A lot of research work has been carried out on the hybrid PSO based task scheduling.

Ali Allahverdi and Fawaz (2006) addressed the assembly flow-shop scheduling problem with respect to the due-date based performance measure (i.e.) maximum lateness. The computational analysis indicated that Tabu outperformed the others for the case where the due-dates range was relatively wide. For tight due-dates, the PSO outperforms the others. They executed the above algorithm for distributed database application. Their objective was to schedule the queries on the servers so that the maximum lateness of all ‘n’ queries would be minimized. They proposed three heuristics such as EDD, PSO and the Tabu Search. They showed that the PSO could be the best for tight due dates and the Tabu as the best for loose due dates.

Tzu-Chiang Chiang et al (2006) proposed a constrained scheduling problem in display system operation. The constrained scheduling problem not only satisfied the resource constraint but also the timing constraint. There were barriers that must be overcome in applying energy function to solve multi-processor task scheduling problems. The sigmoid function had been utilized to produce a probability threshold from the velocity of each threshold particle, then generating new candidate solutions. They showed that the proposed approach was capable of obtaining higher quality solution efficiently in constrained scheduling problems.

Xiaohong Kong et al (2006) incorporated an efficient population-based search technique (PSO) with List Scheduling. They also
compared a few assigning rules to select a target machine with different processing speeds for different tasks. An HPSO first constructs a better scheduling list through the guided particles search technique and the cooperation among the particles then duplicates the critical parent thereby improving the performance further.

Chen Ai-Ling et al (2006) developed a hybrid approximation algorithm for the capacitated vehicle routing problem. It combined global search and local search to search for the optimal results and Simulated Annealing (SA) using a certain probability to avoid being trapped in a local optimum. The DPSO had been used to assign the customers on routes and it was found to be feasible especially for large scale problems.

Fatih Tasgetiren et al (2007) proposed a discrete PSO to solve the TSP on a set of benchmark instances with symmetric distances. It was also hybridized with the Local Search (LS) and variable neighborhood descend algorithm, to further improve the solution quality.

Bassem Jarboui et al (2008) proposed a new algorithm for solving the permutation FSP, namely the combinatorial PSO. Computational experiments conducted with the makespan criterion proved the superiority of the CPSO over the continuous PSO but almost identical in terms of average values. They introduced also an improved phase based on the SA procedure, called the H-CPSO.

Lei Zhang (2008) proposed a heuristic approach based on the PSO to solve the TS problem in grid environment. They considered a finite number of processors in a small scale grid environment and assumed that the processing speeds of each processor and the cost time of each task were
known. In this paper, the Variable Neighborhood Search (VNS) method presented by Hansen has been adopted.

Xiaohong Kong et al (2008) incorporated a miscellaneous population-based search technique PSO with list scheduling for the multiprocessor task scheduling problem. The framework of the hybrid PSO algorithm was developed according to the permutation based solution representation to find the optimal-task-machine pairs. The proposed model considers arbitrary networks in more realistic environments and the scheduling method takes communication delays into account. In this method, the SA has been used to select the next generation and it is utilized to improve the solution quality. In the present research work, the proposed approach is almost the same as these techniques but TS is used in the place of SA.

Hesam Izakian et al (2009) presented a metaheuristic technique for scheduling metatasks in a distributed heterogeneous computing system. The initial population was generated using two scenarios such as random generation of particles and one particle using min-min heuristic.

A TS heuristic is embedded in the PSO iteration to expedite the convergence. The EPSO aims at a global search by taking the concept of the mutation to the PSO, and the HPSO has improved the convergence performance to a better quality solution by taking the concept of natural selection (Kennedy and Eberhart 2001), (Schutte and Groenwold 2005). A few hybrid methods combining TS and PSO had been proposed earlier. For example, in every iteration of the PSO procedure, the algorithm can search the near territory of excellent particle, and it is named as the Tabu-IPSO hybrid algorithm. This procedure simply switches between the PSO and the sequential quadratic programming (SQP) (Boggs and Tole 1995) which is applied to the
unit commitment problem named as the hybrid TS-PSO-SQP (Aruldoss and Ebenezer 2006).

Comparing with all these works present in the literature reviewed, it is noted that the hybrid PSO approach proposed in this present work is different in the following aspects. The Tabu Search Metaheuristic has been applied to the task scheduling problem combined with the PSO. To avoid the local optima of the PSO, the local search technique has been embedded. The proposed PSO-TS approach has shown the improved makespan when compared to the existing techniques like the GA and the PSO.

1.5.3 Works on Contention Aware Scheduling

A lot of research is going on in contention aware task scheduling.

Oliver Sinnen and Sousa (2004) evaluated the accuracy of the task scheduling algorithms and also proposed an evolution methodology. The authors performed an experimental evaluation of the task scheduling accuracy. They showed that the Bubble Scheduling and Allocation algorithm (BSA) was suitable to the contention aware task scheduling than the Dynamic List Scheduling (DLS) approach irrespective of the network topology and target architecture of the computing system.

Oliver Sinnen et al (2005) investigated the incorporation of contention awareness in task scheduling. They proposed a new system model for task scheduling which could capture both end point and network contentions. They had shown how the classic list scheduling could be easily extended to a more accurate system model. This approach could preserve the theoretical basis of task scheduling but had a very small impact on the complexity of scheduling algorithms. Experimental results demonstrated the improvement in accuracy and efficiency of the schedules if link contention is
considered. Contention awareness was achieved by scheduling the edges of the DAG on to links of the topology graph. Their contributions are analyzed and enhanced in the proposed system which deals with the communication contention of the DAG through edge scheduling and the searching technique is improved with Tabu Search.

Oliver Sinnen et al (2006) investigated the involvement of the processor in communication, its impact on task scheduling and also how to make task scheduling aware of it. First, they investigated the different types of processor involvement and their characteristics. They proposed a new system model which could extend the scheduling of the edges on the links to their scheduling on the processors. This technique reflected three types of processor involvement and the distinction between overhead and direct involvement. Scheduling the edges on the processors has an impact on the operating techniques of scheduling heuristics. This challenge is investigated and two solutions are proposed: provisional scheduling and using a given processor allocation. Based on these solutions, two scheduling algorithms are proposed for the new involvement-contention model: an adapted list scheduling and a Genetic Algorithm. Extensive experiments demonstrated that the involvement-contention model significantly improves the accuracy and the execution time of the produced schedules.

Shiyuan Jin et al (2008) made a comparative study with realistic assumptions. They studied 9 algorithms which are frequently used widely in general. They made a detailed analysis of the scalability, advantages and disadvantages of each algorithm.

Tang Xiano Yang et al (2009) investigated the incorporation of contention awareness into task scheduling. They proposed a dynamic scheduling of edges to links through the earliest finish communication time search algorithm based on the shortest-path search method. Their scheduling
priority was based on recursive rank computation on an arbitrary heterogeneous processor network. The present research work implements the contention awareness through edge scheduling and priority is based on bottom level schemes.

Comparing with all these works and assumptions made in these reviewed literature, the proposed approach differs in many aspects. The proposed approach makes use of the BSA and embeds Metaheuristic like TS, SA into it. The priority scheme adopted is bottom level priority and it is adaptive to any dynamic nature of the target architecture and the network topology. The quality of the schedules generated by the proposed approach is better than the existing heuristics like DLS and BSA.

1.5.4 Works on Memetic Algorithm

Availability of literature is not much regarding the job and flow-shop scheduling problem based on the Memetic Algorithm approach.

Edmund and Alistair (2002) proposed a Memetic Algorithm for the maintenance scheduling problem. The effectiveness of the method was tested through its application to real scale problems. Extensive testing suggested that the optimum parameters for this problem were smaller population, combined with a reduced tabu list length and a smaller number of iterations when encountering non-improving solutions during the local search.

Lin Jiannong and Wu (2004) proposed a scheduling algorithm using the Genetic Algorithm. They used indirect decoding techniques. The chromosome length was equal to the number of tasks. The gene’s position was the task serial number assignment to the task. However, GA required more interaction for convergence.
Bo Lu et al (2007) proposed an effective Particle Swarm Optimization (PSO) based Memetic Algorithm (MA) for the Permutation Flow-Shop Scheduling Problem (PFSSP) with the objective of minimizing the maximum completion time. Both PSO-based searching operators and some special local searching operators were designed to balance the exploration and exploitation abilities. Here, SA based local search was designed and incorporated into the MA and an effective meta-Lamarckian learning strategy was employed to decide which neighborhood could be used.

Sang Cheol Kim et al (2007) presented a new static scheduling algorithm based on a guided search strategy, termed Push-Pull, for heterogeneous cluster systems. This algorithm searched for an improved solution starting from the initial solution generated by a baseline algorithm. By iteratively pushing or pulling tasks in a given schedule, the Push-Pull Algorithm attempted to improve the initial solution. This algorithm performed particularly well in improving the initial solutions generated by list scheduling algorithms such as HEFT or HCPT because the pull operation could fix a major shortcoming of list scheduling algorithms. They claimed that the proposed Push-Pull Algorithm provided a practical deterministic method for effective task scheduling in large heterogeneous distributed computing environments.

Antonie Jougl et al (2009) proposed the hybrid flow-shop scheduling problem with multiprocessor tasks. They employed a constraint programming based branch and bound algorithm as the local search engine of the Memetic Algorithm. Experimental results demonstrated that the developed Memetic Algorithm produced better quality solutions when compared to GA and Critical Path based Branch and Bound.

Kamrul Hassan et al (2009) developed a Memetic Algorithm for solving job-shop scheduling problem with an objective of minimizing the
makespan while satisfying the number of hard constraints. Three priority rules were designed namely partial re-ordering, gap reduction and restricted swapping and used local search techniques. They showed that the performance of GA could be improved by incorporating different search and heuristic techniques.

Comparing with all these works present in the literature, the Memetic Algorithm approach proposed in the present research work is different in the following aspects. The Metaheuristic like Tabu Search, Hill Climbing and SA has been applied to the task scheduling problem combined with the GA. To avoid the local optima of the GA, the local search techniques have been embedded. The proposed approach has shown the improved makespan when compared to the existing techniques like the GA and the classical list scheduling heuristics.

1.5.5 Works on Dynamic Resource Aware Scheduling

There are many works available in the literatures that discuss Dynamic Scheduling algorithms.

1.5.5.1 Related Works in Dynamic Task Scheduling

Min Fan et al (2003) proposed a distributed task assignment dynamic algorithm with which IPU\textsuperscript{s} (Independent Processing Units) could adjust their tasks in an adaptive fashion and in turn help the system to get optimal. Their concept was based on the self learning idea of Tung and Li’s.

Beaumont et al (2003) focused on the execution of a complex application, on a heterogeneous “grid” computing platform. In this paper, they dealt with the implementation of mixed task and data parallelism onto heterogeneous platforms. They showed as to how to determine the best steady
state scheduling strategy for a general task graph and for a general platform graph, using a linear programming approach. The objective was to determine the optimal steady state scheduling policy for each processor, (i.e.,) the fraction of time spent computing, and the fraction of time spent sending or receiving each type of tasks along each communication link, so that the (averaged) overall number of problems processed at each time-step was maximal.

Vincent Boudet et al (2003) proposed an original algorithm for mixed data and task parallel scheduling. The main specificities of this algorithm were to simultaneously perform the allocation and scheduling processes, and avoid data replication. The idea was to base the scheduling on an accurate evaluation of each task of the application depending on the processor grid. Applications are represented by task graphs and were composed of dense linear algebra operations. The principle of this algorithm was to associate a list of execution platforms to each node of the task graph. They applied the proposed algorithm to two applications: the complex matrix multiplication and the Strassen decomposition. The performances achieved by the schedules both in the homogeneous and the heterogeneous environment were compared.

Illias and Tahar Kechadi (2004) developed a dynamic scheduling algorithm based on the divide and conquer principle. They modeled the cluster as hyper-grids and then balanced the load among them. But the study did not consider the preemptive tasks and their uses both mixed centralized and decentralized load balancing policies. Also all the nodes in the cluster were almost equally balanced.

Beaumont et al (2004) dealt with the implementation of mixed task and data parallelism onto heterogeneous platforms. Due to space limitations, they mainly concentrated on complexity results. They showed that the
problem of optimizing the steady-state throughput is NP-Complete in the general case. They had shown how to determine in polynomial time the best steady-state scheduling strategy for a large class of application graphs and for an arbitrary platform graph, using a linear programming approach. In particular, they provided positive results for all task graphs whose dependency depth was bounded (which included atomic and tree shaped task graphs, among others).

Ligang He et al (2006) developed workload allocation strategies for multicenter architectures (a heterogeneous cluster of clusters). They had proposed two average-based workload allocation strategies (ORT and OMR) that dealt with different types of job. The ORT strategy could optimize the mean response time of non-real-time jobs, while the OMR strategy could optimize the mean miss rate of a soft real-time job stream. These proposed workload allocation strategies were combined with job dispatching strategies based on the Weighted Random and the Weighted Round-Robin policies. Both ORT and OMR strategies took the mean arrival rate as input and calculate the workload proportions for each cluster. When a change in the mean arrival rate reaches certain thresholds, the proportions of workload were updated dynamically.

Naga Vydyanathan et al (2006) presented an approach to determine the appropriate mix of task and data parallelism, i.e., the set of tasks that should be run concurrently and the number of processors to be allocated to each task. iCASLB, is an iterative coupled processor allocation and scheduling strategy for mixed parallel applications. iCASLB makes intelligent allocation and scheduling decisions based on the global structure of the application task graph and the scalability curves of its constituent tasks. The look-ahead mechanism avoided the local minima and the backfill scheduling improves processor utilization. Experimental results using synthetic task
graphs and those from applications showed that iCASLB achieved good performance improvement over schemes like CPR, CPA, TASK and DATA.

Anca and Dick (2007) proposed four scheduling policies for processor co-allocation in multicruster, and they assessed with simulations their performance under a wide variety of parameter settings. In particular, in their simulations they used synthetic workloads and workloads derived from the logs of actual systems and from runtime measurements. They concluded that although co-allocation makes scheduling more difficult and the wide-area communication critically impacts the performance, there was a wide range of realistic applications that may benefit from co-allocation.

Visalakshi and Sivanandam (2009) presented a HPSO for solving the Task Assignment Problems (TAP). They developed this to dynamically schedule heterogeneous tasks onto heterogeneous processors in a distributed setup. They used PSO for scheduling with fixed and variable inertia. The limitation with this approach was that the natures of the tasks were independent and non-preemptive. In the present research work, their contribution is analyzed and further enhanced with a local search technique and the tasks are dependent.

1.5.5.2 Related Works in Dynamic Scheduling on Multi-Processor Tasks

Janez Brest et al (2002) developed a dynamic scheduling algorithm for the heterogeneous system that could decompose the program workload into computationally homogeneous subtasks, which could be of different sizes, depending on the current load of each machine in the heterogeneous computing system.

Sascha Hunold et al (2008) presented an algorithm RePA for the dynamic scheduling of DAGs of M-tasks onto clusters of clusters. The main objective of the scheduling approach was to keep the redistribution costs
between subsequent tasks small by reusing processors. They had then compared the makespan of RePA and M-HEFT for a large number of DAGs and for various grid configurations.

1.5.5.3 Related Works in Scheduling Preemptable Task on Parallel Processors

Jacek Blazewicz et al (2000) analyzed problems of scheduling preemptable tasks on parallel processors with non-availability periods. They showed that such a problem was strongly NP-hard in case of in-tree precedence constraints, identical processors and makespan criterion. The problem with chain-like precedence constraints could be solved by the algorithm developed for independent tasks. A polynomial time algorithm based on linear programming had been proposed for the maximum lateness criterion. An algorithm based on network flows and the two-phase methods had been proposed for problems with independent tasks scheduled on uniform and unrelated processors.

Stewart Edgar and Alan Burns (2001) developed a method for statistically estimating task WCET based on measurements. They also showed how to statistically estimate the feasibility of a task set, but did not address the problem of finding highly or maximally robust schedules.

John Ragehr (2002) showed that the preemption threshold logic for mapping tasks to as few threads as possible could rule out the schedules with the highest critical scaling factors. He also proposed two novel scheduling abstractions such as task clusters and task barriers that could overcome the limitations of the existing work on preemption threshold scheduling. But CPU reservations were not taken into account.
1.5.5.4 Related Works in MapReduce Scheduling

Literature is available for the MapReduce Framework that discusses the importance of it for the task scheduler.

Jorda Polo et al (2009) proposed a new scheduler for a MapReduce framework that facilitated the performance-driven management of MapReduce tasks. They presented an application centric task scheduling for Hadoop. They proposed a scheduler with a capability to calculate the estimated completion time for each MapReduce job in the system.

Jackson et al (2008) presented an implementation of the MapReduce library supporting parallel Field Programmable Gate Arrays (FPGA) and Graphics Processing Units (GPU). Here parallelization due to pipelining, multiple data paths and concurrent execution of FPGA/GPU hardware were automatically achieved.

Jeffrey Dean and Sanjay Ghemawat (2008) designed a new abstraction that enables simplified data processing that achieves high performance on large clusters of commodity PCs. They suggested the use of a functional model with user-specified map and reduce operations which allow us to parallelize large computations easily and to use reexecution as the primary mechanism for fault tolerance. The present research work implements the same abstraction so that the user can specify the map and reduce operations to be performed in the created cluster computing environment.

Jonathan Cohen (2009) implemented a Graph twiddling in a MapReduce environment. The MapReduce approaches did not implement existing graph algorithms in MapReduce rather they discarded the usual algorithms and found procedures that produce the same outcome. The common patterns followed for the Graph Algorithm are the map operation, a
reduce operation for each vertex bin and another reduce operation for each edge bin.

Chao Tian et al (2009) discussed the MapReduce performance under heterogeneous workloads. They analyzed a typical MapReduce system and classified it into three categories based on the CPU and I/O utilization. They proposed a Triple-Queue scheduler that dynamically determines the category of one job. It contains a waiting queue to test-run the new joined job and to predict the workload type. They found out that the Triple Queue Scheduler could increase the map tasks throughput of the system by 30% and save makespan by 20%.

The schedule problem in MapReduce has also attracted a lot of attention. Matei Zaharia et al (2009) addressing the problem of how to robustly perform speculative execution mechanism under heterogeneous hardware environment. Parag Agrawal (2008) derives a new family of scheduling policies specially targeted to sharable workloads.

Pravanjan Choudhury et al (2008) proposed a methodology for task graphs that were dynamic in nature due to the presence of conditional tasks and tasks whose execution times were unpredictable but bounded. They presented the methodology as a three-phase strategy, in which task nodes were mapped to the processors in the first (static mapping) phase. In the second (selective duplication) phase, some critical nodes were identified and duplicated for possible rescheduling at runtime, depending on the code memory constraints of the processors. The third (online) phase was a runtime scheduling algorithm that performs list scheduling based on the actual dynamics of the schedule up to the current time. They showed that this technique provided better schedule length (up to 20 percent) compared to all the other previous techniques, which were predominantly static in nature, with low overhead and a complexity comparable with existing online techniques.
They investigated the effects of model parameters like the number of processors, memory, and various task graph parameters on performance in this paper.

Due to the NP-Completeness of the task scheduling problem, there are a huge number of heuristics for finding an optimal makespan. From the literature discussed in this thesis, it is seen that it is necessary to propose metaheuristic approaches for the task scheduling problem. Metaheuristics avoid local minima and guide the search towards global optima. Hence, in this work, various metaheuristics have been applied to the task scheduling problem and the work tries to find an optimal makespan.

1.6 CONTRIBUTION OF THESIS IN COMPARISON WITH THE RELATED WORKS

Compared to all the works in the literature reviewed, the heuristic algorithms discussed in this thesis are different in many ways. First, the SLTS proposed in this thesis provides a scalable low complexity task scheduling heuristic for a homogeneous cluster of workstations. Second, this thesis provides a Hybrid PSO technique with Tabu Search as hybrid search strategy introducing a probability to avoid getting trapped in a local minimum. Although PSO is very robust and has a good global exploration capability, it has the tendency to be trapped in local minima and slow convergence. Then the contention for communication resources is achieved in order to improve the scheduling accuracy through a new scheduling model where the network is modeled as a topology graph and processors are assigned to them. Guided random search techniques like Simulated Annealing and Tabu Search are applied to enrich the searching ability to improve the makespan. But the CPU load and the data locality of the tasks are not aware by the classical scheduling model which can be achieved through a new programming model namely, MapReduce. A Resource Aware Scheduler developed in this research
work based on MapReduce model can adapt to any distributed environment. Thus the proposed Resource Aware Scheduler could effectively use both the CPU and I/O resources of the cluster.

1.7 CLUSTER OF WORKSTATIONS ENVIRONMENT

High performance Cluster Of Workstations (COW) are created for the research work using MPICH2-1.0.2 which is a programming library for inter processor communication in the Message Passing Interface (MPI) and it consists of 30 HP P-IV, 3 GHz processors connected through 100 Mbps Ethernet connectivity. Previously, the PVM Library was used to form clusters. MPI gives the illusion of a single system image for the clustering environment.

BENCHMARKED two famous combinatorial problems such as Matrix Multiplication and the Knapsack problem which are known to be NP-Hard problem are deployed in the available cluster computing environment. Their scalability and performances are analyzed by varying the number nodes in the cluster. The entire programs were written using C in MPI under the Master / Slave Paradigm model.

1.7.1 Matrix Multiplication

Matrix Multiplication is a fundamental operation in many numerical linear algebra applications. Its efficient implementation on parallel computers is an issue of prime importance when providing such systems with scientific software libraries. Many parallel algorithms have been designed, implemented, and tested on different parallel computers or cluster of workstations for matrix multiplication. With the emergence of portable message-passing interfaces (PVM and MPI), the parallel matrix multiplication algorithms were implemented in a portable manner, distributed widely and
used in applications. There exist some high-performance library routines for this operation, scalAPACK, PBLAS, ATLAS. They are often optimized based on the target architecture.

Tests were conducted on $N \times N$ matrices where $N$ was 100, 200, 500, 1000, 2000 and 10,000. The proposed algorithm gives better performance for different matrices size and it can be executed in any number of processors. The master has to allocate the process among the processors dynamically and it has to wait until it receives the partial computed result from every other process that is running in the other processors.

### 1.7.2 Knapsack Problem

The Knapsack problem is one of the most powerful problems in combinatorial optimization, and is known to be NP-Hard. Using Dynamic Programming the problem can be solved serially in pseudo-polynomial time. For many combinatorial problems, exhaustive search is unfeasible. In using Dynamic Programming (DP), the computational cost is so high and the memory requirement is also high. Parallel computers greatly reduce the computation time for solving large scale problems since there are a number of parallel operations that occur in the evaluation of the DP recursive formulae. The parallel implementations of the Knapsack problem in hypercube and ring topology are based on precedence graph. The proposed methodology known as summing the subset from an original input set checks to see whether it satisfies the objective function is implemented in the cluster of workstation environment.

### 1.8 CONTRIBUTION OF THESIS

In this work, the Scalable Low complexity Task Scheduling heuristics is proposed which gives a better schedule when compared to the
existing list scheduling heuristics in terms of the makespan, efficiency and scalability. The Hybrid PSO approach proposed in this thesis performs better when compared to the GA and PSO. It combines the excellence of both PSO and TS. The proposed TS and SA based Contention aware scheduling approach outperforms the List Scheduling, DLS and BSA. It can adapt to any target computing system architecture and network topology and is able to capture both end point and network contentions thereby improving the quality of the schedules. The Memetic Algorithm based approach for the task scheduling problem is introduced and enhanced with Local Search techniques like Hill Climbing and TS. The Resource Aware Scheduler developed in this work is scalable in distributed environment and considers the pre-emptive tasks. Profiling and Scheduling is done based on the data-locality of the MapReduce tasks in the cluster computing environment. By applying the suitable heuristic algorithms, the tasks are effectively scheduled in the cluster computing environment thereby producing the optimal makespan.

1.9 THESIS ORGANIZATION

Chapter 1 presents a survey of the related works which are compared with the proposed work and the relevant background of the task scheduling problem.

Chapter 2 proposes the simple list scheduling heuristic called the Scalable Low complexity Task Scheduling Algorithm (SLTS). The proposed approach is scalable and has less complexity when compared to other list scheduling heuristics.

Chapter 3 explains the proposed Hybrid PSO based technique that leads to better result and converges very fast when compared to the conventional Genetic algorithm.
Chapter 4 introduces contention awareness through edge scheduling into the SLTS. This work proposes a Tabu Search and Simulated Annealing based list scheduling heuristics which extend link contention.

Chapter 5 introduces a Memetic Algorithm based on the Tabu Search and the Hill Climbing concept into the task scheduling problem. This chapter explains how task scheduling is made contention aware by integrating edge scheduling and the topology graph into the scheduling process.

Chapter 6 presents the real time implementation of the task scheduling concept using the MapReduce Framework. The developed Resource Aware Scheduler considers preemptive tasks with the awareness of the computing resources available for execution using profile information.

Chapter 7 sums up the necessary conclusions that are arrived at in this work and indicating the further directions in applying the computational intelligent techniques for scheduling tasks in cloud environment.