CHAPTER 2

LITERATURE SURVEY

During the past years, a significant amount of interest has been shown in the development of various scheduling algorithms for grid environment. Many algorithms have been proposed by researchers for scheduling jobs in the grid system. This chapter explores a comprehensive review of different job scheduling methods with their merits and demerits.

2.1 INTRODUCTION

Efficient scheduling of jobs to appropriate resources is very much essential to reduce waiting time and increase throughput of the grid system. A simple scheduling algorithm cannot cover all the aspects of jobs and resources such as resource utilization, throughput and makespan. Load balancing is another vital feature that should be built with the scheduler to take care of balanced load in the resources. Fault tolerant techniques are essential to reduce failures in job execution.

2.2 SCHEDULING ALGORITHMS IN GRID

Researchers have proposed static heuristic algorithms for mapping of independent tasks to the appropriate resources in order to improve resource utilization and reduce makespan. Minimum Execution Time (MET) algorithm (Armstrong et al 1998) allocates jobs to the machine with minimum expected execution time. In this method, minimum execution time is considered for assigning the job without considering resource availability. It does not
consider the expected completion time of jobs on the machine. It assigns each job to its best machine but produces severe load imbalance since job allocation is done without considering resource availability which results in less resource utilization.

Opportunistic Load Balancing (OLB) algorithm (Freund & Gherrity 1998) allocates jobs in random order to the machines which are idle. The main goal of this algorithm is to keep all the resources as busy as possible. It does not consider the expected execution time of a job on a machine which has poor makespan. Its implementation is very simple because the complexity of this algorithm is $O(m)$ where $m$ is the number of resources.

Min-min heuristic algorithm has been studied by Maheswaran et al (1999). It schedules a set of independent tasks iteratively by considering the set of unmapped jobs and calculates the expected completion time for all the jobs in the set in all the machines. For each job, the machine with minimum expected completion time is identified. Finally, the job with minimum expected completion time from among all other jobs in the set is allocated to the machine which has the minimum completion time for that job. Subsequently, the scheduled task is removed from the unassigned list of tasks and load of the resource is updated. This process is repeated for the remaining unmapped tasks. In this algorithm, the makespan is comparatively improved but the problem of idleness of the machine remains unsolved.

Backfilling algorithm (Mu`alem & Feitelson 2001) is an improved first come first serve algorithm which is mainly designed to increase the resource utilization. Here the expected execution time is calculated, so that the scheduler can predict when the job will be completed. In case when the oldest job in the queue cannot be executed at that time, due to unavailability of required resources, remaining jobs in the queue are exploited by the scheduler whether they can be executed without delaying the oldest queued job. If such
jobs are found, they are scheduled immediately. It improves resource utilization and throughput because idle resources are assigned with suitable jobs.

Subramani et al (2002) proposed a distributed scheduling algorithm based on multiple simultaneous requests at different sites. In this approach, information about other sites is collected and decision is made based on the collected information. It is divided into two models such as K-Distributed Model and K-Dual Queue Model. In K-Distributed Model, each job is scheduled to K sites which are least loaded by the local meta-scheduler. The jobs in each of these K sites are scheduled locally. When a job is started in any of these sites, the site informs the meta-scheduler at the job-originating site, which in turn contacts other K-1 meta-schedulers to remove the jobs from their queues. The value of K is varied based on the scalability required. When K is increased, the amount of communication and synchronization are increased. In K-Dual Queue Model, dual queues are used at each site i.e. one for local jobs and the other for remote jobs. When a job is submitted to the meta-scheduler at a site, it schedules it to the K least loaded sites. It is suitable only for homogeneous resources and does not take data requirements into account.

Xiaoshan et al (2003) presented a QoS guided min-min heuristic algorithm for scheduling tasks in grid environment. It chooses the best match host for an application and the selection strategy is based on prediction of the computing power of the host. The expected Completion Time \( CT_{ij} \) of task \( t_i \) on machine \( m_j \) is calculated using the following equation.

\[
CT_{ij} = b_i + ET_{ij}
\]  \hspace{1cm} (2.1)

where \( b_i \) is beginning time of \( t_i \) and \( ET_{ij} \) is the expected Execution Time of task \( t_i \) on machine \( m_j \). It schedules jobs by considering the network
bandwidth that is required for the job. The task with high QoS request is first scheduled. User demand is not considered in this algorithm to improve user satisfaction.

Dong et al (2006) proposed an algorithm called QoS priority grouping scheduling. It considers deadline and acceptation rate of the tasks and the makespan of the whole system as major factors for task scheduling. It classifies the jobs into two categories such as high QoS requiring jobs and low QoS requiring jobs. It considers QoS requirement of the jobs such as deadline and schedules the jobs.

Lee et al (2006) proposed an adaptive task scheduling algorithm which is developed for high-throughput applications. In this method, the length of a task represents its workload which is based on the number of instructions in the task. Based on the utilization and capacity of a resource, the capability of a processor can be calculated. It indicates the workload completed by a processor per unit time. Scheduler selects a suitable resource for a job based on the capacity of the resource and length of the job which is represented in the number of instructions. It uses waiting and execution queues to manage all the tasks. The waiting queue is used to manage the tasks that are not yet allocated to processors. When a task is assigned to a resource, then the task is removed from the waiting queue and moved to the execution queue. Each task in the execution queue has a specific order and the tasks are scheduled based on round robin methodology.

Sumathi & Gopalan (2006) proposed a priority based scheduling algorithm in which priority of the jobs is taken into account. It classifies the jobs into high, medium and low categories based on their priority. The priority is assigned to a job based on their computational power and level of parallelism. The value for level of parallelism is assigned based on the amount of parallelism exhibited by the job and the available resources. In this
algorithm, a job is given high priority when it needs high computational power and low parallelism. A job which exhibits high parallelism and needs low computational power is given low priority. A job which exhibits a medium level of parallelism and needs normal computational power is given medium priority. The job which has the highest priority is scheduled to the fast free resource available in the grid.

Lin et al (2007) proposed an application demand aware approach which concentrates on user satisfaction by taking application requests into account. It is both system centric and application centric. The expected completion time for each job in all the resources is calculated. The given expected completion time is compared with the expected completion time of each job and the job with smaller value is allocated to the resource which has minimum expected completion time. User satisfaction is improved by allocating most suitable resources to jobs without missing their expected completion time.

McClatchey et al (2007) proposed a data intensive and network aware meta-scheduling approach which takes data requirements, processing power and network characteristics into account when making scheduling decisions across multiple sites. It considers network characteristics as a first criterion in the scheduling decision matrix along with computation and data. The scheduler makes decisions by taking into account the changing state of the network, locality and size of the data and the pool of available processing cycles. The disadvantage of this approach is that jobs might fail due to configuration errors at the destination sites.

Minimum Completion Time (MCT) algorithm (Baghban & Rahmani 2008) allocates jobs randomly to the machine with minimum expected completion time for that job. It uses Completion Time (CT) which is defined as sum of Ready Time (RT) of the resource and job Execution Time
(ET) on the selected resource. It results in less resource utilization. The performance of this algorithm is better compared to OLB and MET but still the load imbalance problem occurs.

Max-min algorithm (Wenzheng & Wenyue 2009) calculates the expected completion time for all the jobs in the unassigned list of jobs on the list of machines. The job with maximum expected completion time is scheduled to the machine which has minimum completion time for that job. Subsequently, the scheduled task is removed from the unassigned list of tasks and workload of the resource is updated. The above procedure is repeated until no jobs remain in unassigned list. This improves the makespan and balances the load to some extent and performs better for jobs with longer execution time.

Abbaa et al (2012) proposed a modified prioritized deadline based scheduling algorithm using project management techniques for efficient job execution with deadline constraints of jobs. It executes jobs with closest deadline time delay in a cyclic manner using dynamic time quantum. It assigns time quantum by computing Least Common Multiple (LCM) of all burst time and sorts the jobs based on time delay in ascending order. Then the job with minimum time delay is selected for execution. If multiple jobs have the same time delay, then a job is selected based on arrival time. Jobs are preempted based on time quantum and if a job completes its execution before time quantum then the job is deleted from queue. After each cycle, a new time quantum will be assigned by computing LCM of all remaining burst time of jobs.

Bhatia et al (2013) proposed a secure requirement prioritized task scheduling algorithm which schedules the jobs based on the resource requirement of the jobs. It considers the memory requirement as the resource requirement of the jobs. In this algorithm, memory requirement of the job is
passed to scheduler in encrypted form. It sorts and makes a list of tasks based on their memory requirement as from complete list of tasks. Then it encrypts the memory requirement of the tasks in the list using Rivest Cipher 5 (RC5) algorithm and makes a list of tasks with their encrypted memory requirement. The scheduler collects the list, decrypts the memory requirement of the jobs and finds the list of resources which satisfy the memory requirement of the task. Then the scheduler selects a resource with minimum completion time and assigns the job to it.

2.3 GROUPING BASED SCHEDULING ALGORITHMS

Ranganathan & Foster (2002) proposed a set of job-scheduling and data-movement algorithms for data intensive applications. This algorithm considers computation scheduling and data scheduling are independent and does not incorporate them to get suitable resources for jobs. This approach considers the scheduling framework in which scheduling logic is divided into three modules. They are external scheduler, local scheduler and dataset scheduler. External scheduler collects the jobs from users and decides on the remote site to which the job has to be sent based on external information like load at a remote site or the location of a dataset to make its decisions. Local scheduler of a site decides how to schedule all jobs allocated to it on its local resources. Dataset scheduler at each site keeps track of the popularity of each dataset locally available.

Keat et al (2006) developed an algorithm which groups independent jobs with less processing requirements into suitable jobs with large processing requirements. It considers network bandwidth for scheduling but it does not consider the dynamic characteristics of resources, utilization of the resources and user demands of the tasks. In this method, job grouping and job selection are done. In job grouping, jobs submitted by the user to the scheduler are collected and grouped together based on the information of resources. The
size of a grouped job depends on the processing requirement length. When a resource is selected from the resource list, its processing capability is used as the maximum limit for a grouped job. At the same time, job selection is also conducted where a grouped job corresponds to the resource in question. The process is performed iteratively until all jobs are grouped according to their respective resources.

Muthuvelu et al (2005) presented a dynamic job grouping strategy which concentrates on maximizing the utilization of grid resource processing capabilities and reducing the overhead time and cost to execute jobs through a batch mode dynamic scheduling. But this algorithm does not consider user demand of each task and does not improve user satisfaction. In this method, job grouping is done based on the processing requirements for each job, the processing capabilities of available resources and granularity size which is defined as the time within which a job is processed at the resources.

Ang et al (2009) proposed a bandwidth aware job grouping based scheduling approach which focuses on grouping independent jobs with small processing requirements into large jobs. Grouping of jobs is done based on the resource capability and size of the jobs. Then, these grouped jobs are sent to the corresponding resources based on largest job first strategy and results of the processed jobs are sent back to the users. It focuses on both computational and communicational capabilities of the resources when scheduling the user’s jobs.

Liu & Liao (2009) proposed an Adaptive Fine-grained Job Scheduling (AFJS) algorithm which focuses on scheduling lightweight jobs in grid computing. It uses the resource monitoring mechanism to collect latest details about the grid resources. Based on the status of the resources, lightweight jobs are grouped as coarse-grained jobs according to processing capabilities and bandwidth of the resources. The algorithm has two phases. In the first phase, the scheduler receives the status of the resources and sorts the
jobs in descending order and allocates a new ID for the jobs. In the second phase, the lightweight jobs are grouped into clusters that still meet the constraints. This method is mainly used to improve utilization of the resources.

Selvarani & Sudhasadhasivam (2010) developed a heuristic approach based on particle swarm optimization algorithm for scheduling tasks in grid environment. It groups the tasks in non uniform manner and scheduling is done based on the processing capability of the resources. By grouping the jobs, this approach optimizes computation/communication ratio and the utilization of resources. But user demands of the tasks are not considered while scheduling the tasks.

Kaur & Kaur (2013) proposed an efficient load balancing dynamic grouping based job scheduling approach for grouping the fine-grained jobs. It consists of two phases: (1) Creating available resource and job list (2) Job grouping and scheduling. Initially, jobs are received and stored in a job list. Then resources are sorted in descending order based on their processing capabilities in MIPS (Million Instructions Per Second) and jobs according to their length in MI (Million Instructions). Then, the number of jobs in each group is specified based on total number of jobs and resources. Jobs are added into groups based on group size by alternatively taking jobs from front end i.e. job with highest length and rear end of the job list i.e. job with smallest length in the job list. Then, jobs are scheduled based on the number of jobs at a particular time and the resource capability.

2.4 LOAD BALANCING ALGORITHMS

Cao et al (2005) proposed a grid load balancing technique which uses artificial intelligence techniques to achieve effective workload and resource management. A combination of intelligent agents and multi-agent approaches is applied to both local grid resource scheduling and global grid
load balancing. Each agent is a representative of a local grid resource and utilizes predictive application performance data with iterative heuristic algorithms to ensure local load balancing across multiple hosts. At a higher level, agents cooperate with each other to balance workload using a peer-to-peer service advertisement and discovery mechanism.

Payli et al (2006) proposed a dynamic load balancing scheme which provides application level load balancing for individual jobs. It ensures that jobs submitted through dynamic load balancing environment are distributed in such a way that the overall load in the system is balanced and jobs get maximum benefit from available resources.

Yagoubi & Slimani (2006) proposed layered dynamic load balancing algorithm based on tree model representation. This model transforms any grid architecture into a unique tree with four levels. For each site, it first creates a two-level sub tree. The leaves of this tree represent the computing elements of a site and the root represents a virtual node associated to the site. These sub trees, that correspond to sites of a cluster are grouped together to form a three-level sub-tree. Finally, these sub-trees are grouped together which form a four-level tree called load balancing generic model. The features of this algorithm include a layered model and support heterogeneity and scalability, totally independent of any physical architecture of grid.

Load balancing mechanism for optimal load distribution in a non-dedicated cluster or grid computing system with heterogeneous servers processing both generic and dedicated applications was proposed by Li (2008). It uses an optimal load distribution strategy for generic tasks on heterogeneous servers preloaded with different amount of dedicated tasks such that the overall average response time of generic applications is minimized.
Li et al (2009) proposed a hybrid load balancing algorithm for sequential tasks in grid environment. The main objective of this algorithm is to achieve minimum execution time, maximum node utilization and balanced load across all the nodes in the grid. The instantaneous scheme, first-come-first-served of the hybrid scheduler function is used to find the earliest completion time of each task individually. If the system workload grows heavy (i.e. if more tasks are waiting in the queue), then the scheduler performs load balancing. The tasks are then shifted to other systems so that the overloaded condition is avoided.

Yan et al (2009) proposed a hybrid load balancing policy which integrates static and dynamic load balancing technologies. The static load balancing policy is applied to select effective and suitable set of nodes. When a node reveals possible inability to continue providing resources, the dynamic load balancing policy will determine whether the node in question is ineffective for providing load assignment or not. The system will then obtain a new replacement node within a short time to maintain system execution performance.

Yagoubi & Meddeber (2010) proposed a load balancing model which has two folds: First, a distributed load balancing model transforms any grid topology into a forest structure. Second, a two level strategy is used to balance the load among resources of computational grid.

- Level 0: Each cluster manager is associated with a physical cluster of the grid. In this load balancing strategy, cluster manager is responsible for:
  - Maintaining the workload information to all worker nodes
  - Estimating the workload of associated cluster and diffusing this information to other cluster managers
- Deciding to start intra-cluster load balancing
- Sending the load balancing decisions to the worker nodes which they have for execution
- Initiating inter-cluster load balancing
  - Level 1: At this level, the worker nodes of a grid that are linked to their respective clusters are determined. Each node at this level is responsible for
    - Maintaining its workload information
    - Sending workload information to its cluster manager
    - Performing load balancing decided by its cluster manager

Balasangameshwara & Raju (2011) proposed a fault tolerant hybrid load balancing algorithm which consists of two phases: static load balancing and dynamic load balancing policy. In the first phase, a static load balancing policy selects the desired effective sites for executing the submitted jobs. If any of the sites is unable to complete the assigned job, a new site will be identified using the dynamic load balancing method. The assignment of jobs must be adjusted dynamically in accordance with the variation of site status. The variation in site status can be identified by following three conditions.

- When the grid scheduler receives the message that a certain site can no longer provide resources
- When job execution on a certain site exceeds the expected execution time
- When the site is overloaded.
Kokilavani & Amalarethinam (2011) proposed a load balanced Min-min algorithm which is mainly implemented for reducing the makespan and increasing the utilization of resources in grid environment. It is implemented in two phases. In the first phase, the Min-min algorithm is applied to schedule the tasks. In the second phase, tasks in the overloaded resources are rescheduled to effectively use the underutilized resources.

Kumar (2011) proposed a dynamic load balancing algorithm based on resource type policy. In this algorithm, jobs with higher order are completed first which means that these tasks are given higher priority than others which leads to starvation. It increases the completion time of tasks and the load balance is also not guaranteed. This algorithm makes changes to the distribution of works among workstations at run-time. It uses current or recent load information when making distribution decisions. Multi-computers with dynamic load balancing allocate/reallocate resources at runtime based on priori task information, which may determine when and whose tasks can be migrated. It schedules the tasks by using fair completion time and reschedules by using mean waiting time of each task to obtain load balance. It provides optimal solution so that it reduces the execution time and expected price for the execution of all the jobs.

Payli et al (2011) proposed a dynamic load balancing protocol for grid environment. Here, the grid is partitioned into a number of clusters. Each cluster has a coordinator which performs local load balancing decisions and communicates with other cluster coordinators across the grid to provide inter cluster load transfer and balancing. The distributed protocol uses the coordinator of the cluster to perform local load balancing within the clusters and if this is not possible, load balancing is performed among the clusters under the control of cluster coordinators.
Qureshi et al (2011) proposed a dynamic and distributed load balancing methodology for grid environment. The proposed scheme uses distributed methodology which reduces the communication overhead at resource site and idle time of the resources during load balancing. The proposed load balancing approach is based on enhanced gridsim architecture. It works at three levels: Broker, Resource and Machine. When a new job arrives at a machine, it is submitted to a processing element which is lightly loaded.

Bardsiri & Rafsanjani (2012) proposed a load balancing mechanism which works in two phases. In the first phase, minimum completion time is found for each job in each resource like Min-min heuristic algorithm. In the second phase, jobs are allocated to the machines based on workload of the machines.

Alharbi (2012) presented a Simple Scheduling Algorithm with Load Balancing (SSALB). First, the completion time for each task on each machine is calculated. Then, the task with maximum average completion time is selected and the selected task is assigned to the machine with minimum completion time.

Hao et al (2012) proposed a dynamic, distributed load balancing methodology which provides deadline control for jobs. First, states of the resources are checked and then the grid broker assigns the jobs according to the change of load state based on the deadline request.

Maheshwari & Bansal (2012) proposed a load balancing algorithm which schedules the tasks based on minimum completion time and reschedules by waiting time of each task to obtain load balance. In this method, higher priority is assigned to the resource which has less loading factor and lower priority is assigned to the resource which has high loading
factor. Then, it calculates total cost for the execution of the jobs on the resources based on priority, execution time and communication time. Finally, it schedules jobs to the resources based on total cost.

Raj et al (2012) proposed an augmenting hierarchal load balancing algorithm. To evaluate the load of the cluster, probability of deviation of average system load from average load of cluster is calculated and checked for the confinement within a defined range of 0 to 1. The fittest resources are assigned with the jobs by comparing the expected computing power of the jobs with average computing power of the clusters.

Ezzat et al (2013) proposed a grouping based scheduling with load balancing for fine grained jobs. It groups fine grained jobs to form coarse grained jobs based on the processing capability of resources and the processing requirements of grouped jobs. The scheduler selects a resource and computes the product of MIPS and granularity time which is used to measure the total number of jobs that can be completed within a specified period of time. Then, it selects the resource that has minimum number of waiting jobs.

2.5 FAULT TOLERANCE BASED SCHEDULING ALGORITHMS

Hwang & Kesselman (2003) proposed a failure detection service and flexible failure handling mechanism. It is used to detect both task crashes and user defined exceptions. It uses a notification mechanism which is based on the interpretation of notification messages being delivered from the underlying grid resources. This method allows users to achieve failure recovery in a variety of ways depending on the requirements and constraints of their applications. It describes how to achieve flexibility by the use of workflow structure as a high-level recovery policy specification, which enables support for multiple failure recovery techniques, the separation of
failure handling strategies from the application code and user-defined exception handlings.

Dabrowski (2009) proposed various fault recovery mechanisms which include check pointing, replication and rescheduling.

- **Check pointing**: Taking checkpoints is the process of periodically saving the state of a running process to durable storage. This allows a process that fails to be restarted from the point its state was last saved or its checkpoint on a different resource.

- **Replication**: Replication means maintaining a sufficient number of replicas or copies of processes executed in parallel on different resources so that at least one replica succeeds.

- **Rescheduling**: Rescheduling finds different resources to reschedule the failed tasks.

Nazir et al (2009) proposed a fault tolerant based job scheduling strategy which is used to tolerate faults for an economy based grid environment. In this approach, fault index of grid resources is maintained. Then the fault index value is updated based on successful or unsuccessful completion of an assigned task. The method considers fault index of grid resources to schedule jobs.

Zheng & Veeravalli (2009) developed a fault tolerant scheduling approach for computational grid to schedule backups and minimize the response time of jobs. In this approach, the jobs are modeled by directed acyclic graphs. It schedules the jobs with communication delays so that execution failures can be avoided even in the presence of processor faults. First, a communication model is developed which determines when
communication between a backup and backups of its successors is necessary. Then the backup can start upon any processor failure. This minimizes response time and replication cost.

Nandagopal & Uthariaraj (2010) proposed a fault tolerant scheduling strategy which uses checkpoint replication based fault tolerance mechanism with Minimum Total Time to Release (MTTR) job scheduling algorithm. In this method, Total Time to Release (TTR) includes execution time of the job, waiting time in the queue, transfer time of input and output data to and from the resource. This algorithm minimizes TTR by selecting a computational resource based on job requirements, job characteristics and hardware features of the resources. It uses job checkpoints based on the resource failure rate. In case of resource failures, the job is restarted from its last successful state using a checkpoint file from another grid resource. Replica Resource Selection Algorithm (RRSA) is proposed to provide Checkpoint Replication Service which increases the availability of checkpoint files.

Priya & Ravichandran (2011) proposed a checkpointing mechanism to achieve fault tolerance. The checkpointing process periodically saves the state of a process running on a computing resource so that in the event of resource failure, it can resume on a different resource. If any resource failure happens, it invokes the necessary replicas in order to meet the user application reliability requirements.

Amoon (2012) proposed a Fault Tolerant Scheduling Algorithm (FTSA) for computational grids. It calculates scheduling indicator for selecting resources. Scheduling indicator is calculated based on response time and failure rate of grid resources. The grid scheduler schedules the job to the grid resource which has the minimum scheduling indicator.
Keerthika & Kasthuri (2012) proposed a Fault Tolerant Min-Min (FTMM) scheduling approach for computational grid. It calculates expected completion time (CT\textsubscript{ij}) for all tasks in all the resources which is the sum of Expected Time to Compute (ETC\textsubscript{ij}) of task i in resource j and Ready Time of resource j (RT\textsubscript{j}) by using the following formula.

\[ CT_{ij} = ETC_{ij} + RT_j \]  

(2.2)

It then calculates Total Completion Time (TCT\textsubscript{ij}) of each task Ti on each resource Rj and it is given by

\[ TCT_{ij} = CT_{ij} + CMT_{ij} \]  

(2.3)

where CMT\textsubscript{ij} is the communication time of each task Ti to the resource Rj. Then it calculates fitness value based on total completion time and failure rate of the resources. Finally, jobs are scheduled based on the fitness value.

Latchoumy & Khader (2012) proposed an Improved Fault Tolerant Job Scheduler (IFTJS) for optimal resource utilization in computational grid. The algorithm maintains the history of fault occurrence of resources with respect to memory, processor and bandwidth. This information is used to reduce chance of selecting a resource which has more probability of failure and thus improves the resource utilization. It assigns highest priority to resources with the lowest failure rate. The job manager monitors the execution of the job and returns the results to the user after successful completion. If failure occurs, it re-executes the job with the same resource using the last saved state when the failure rate of the resource is lesser than the optimal value or with the backup resources when it exceeds an optimal
value with the last saved state using Reduced Recovery Time (RRT) strategy. Otherwise it reschedules the failed job with the next available optimal resource using the last saved state. Hence the recovery time of job execution gets reduced.

Keerthika & Kasthuri (2013) proposed bi-criteria scheduling algorithm that considers user satisfaction along with fault tolerance. It focuses on a pro-active fault tolerant mechanism which considers failure history of the resources while scheduling jobs. It considers user deadline, total completion time of the jobs at all the resources and calculates fitness value. Then, jobs are scheduled based on the fitness value.

A Multi Criteria Scheduling Algorithm (MCSA) is proposed by Keerthika & Kasthuri (2013). It considers resource heterogeneity, resource availability and job characteristics such as user deadline and designs scheduling model with load balancing and fault tolerance. It schedules the jobs based on load factor, expected completion time and user deadline.

2.6 CONCLUSION

All these heuristic scheduling algorithms discussed in this chapter have advantages and also some disadvantages. The Opportunistic load balancing algorithm does not consider the expected execution time and henceforth its makespan is poor. Minimum Execution Time heuristic algorithm does not consider completion time of jobs which results in severe load imbalance. MCT also results in poor makespan. Max-min heuristic performs better when compared to all these algorithms but only for the shortest jobs. Among all these heuristic algorithms discussed, Min-min
heuristic is simple, fast and performs better by considering the system performance by reduced makespan without considering user satisfaction.

Application demand aware approach performs better when user satisfaction is taken into account. Many scheduling algorithms focus on user deadline and load balancing separately but no scheduling algorithm considers both user deadline and the load of the resources. Hence there is a large scope for scheduling algorithms that focus on both these factors.