CHAPTER 2
AN ADAPTIVE JOB SCHEDULING WITH EFFICIENT FAULT TOLERANCE STRATEGY IN COMPUTATIONAL GRID

2.1 INTRODUCTION

In this work, an efficient scheduling strategy is proposed which poses a key challenge for scheduling light weight jobs using Adaptive Job Scheduling (AJS) Strategy. The system also makes an attempt to tolerate faults with an effective fault tolerance strategy. The performance of the system is analyzed and it is found that the algorithm proposed is best suited for light weight jobs in the grid environment.

The initial discussion starts with the grid architecture, computational grids, scheduling grid resources, fault tolerance and the grid system environment. A brief overview of the existing Serial Job Scheduling Strategy is also discussed followed by a detail discussion on the proposed Adaptive Job Scheduling algorithm.

2.1.1 Grid Architecture

The architecture shows the grid computing resources scattered globally and aggregated, thus, providing diverse resources to the users. The grid environment might hold devices located at common places but possesses the facility to work independently from diverse locations. The group of
servers and other computing resources aggregated to solve the regular problems form a grid. Such grids consist of diverse hardware and computational systems along with the protocols and various services to perform certain grid functionalities at different layers of grid.

The grid is a layered architecture for providing protocols and services in five various layers as represented in Figure 2.1.

![Grid Architecture Diagram]

**Figure 2.1 Grid architecture**

The fabric layer is the bottom most layer of this architecture, supplying distributed resources like bandwidth for the network, time for processing the job in Central Processing Unit (CPU) and scientific devices like sensors, memory management and so forth. The received data at this layer are communicated directly to other communicating devices or get accumulated inside the grid database. Standard protocols are available within the grid for controlling resources, and resources are shared for measuring QoS requirements. Additionally, kernels, operating systems and queuing systems are also included as a part of this layer.
The connectivity layer offers safe and easy access to services and is responsible for providing communication and authentication needed for connections. The protocols responsible for communication allow the exchanging of information between the resource layer and the fabric layer. The protocols responsible for authentication provide safety mechanisms for detecting the users and resources.

The resource layer is responsible for protocols that work well with distributed resources, and is constructed upon the communication and authentication protocols of connectivity layer for describing Application Program Interfaces (API’s) and System Development Kit (SDK) for protected cooperation, organization, commencement and management of distributed resources.

The collective layer is responsible for general services. Mutual operations within the distributed resources are placed in this layer which synchronizes resource distribution like indexing services, scheduling, brokering services, organization, information analysis and data duplication services.

The application layer is the top most in the grid architecture, containing user applications and their implementations, offering a gateway to the users and supervisors for communicating with the grids.

The grid architecture having different layers performs their functionalities associating different activities and protocols, highly contributing towards resource allocation for scheduling the job in the computational grids. The major role and contribution of the computational grids are also discussed extensively.
2.1.2 Computational Grids

The computational grid is a scattered architecture that resembles the user as a single huge computing resource, and transforms the view on computations and resources. Each and every element of a grid is under the influence of the user and appears scattered over a network of increasing size. There are several CPU’s and databases scattered over a Local Area Network (LAN) to include diverse resources over a Metropolitan Area Network (MAN) and Wide Area Network (WAN). Due to the technological advancements, more consistent, minimal cost services and worldwide access are also possible.

The infrastructure of the computational grids is a collection of huge computing resources including data, sensors and people. These huge computational resources require hardware components for attaining necessary interconnections and software components for examining and controlling the results. The computational grid provides various services like Consistent Service, Reliability, Universal Access and Low Cost.

Consistent Service is the second fundamentally required service for accessing the standard connections and operations with standard parameters. Without these standards the policies and universal access are not feasible. The great challenge occurs while aggregating diverse resources without compromising the performance.

Reliability is a fundamentally required service and guarantees should be provided to the users for providing expected, constant and improved performance from a variety of elements within grids. The performance vary from one application to another, based on bandwidth utilized by the networks, latency, jitter, execution time, security, services provided by software and reliability.
Universal Access represents the services available within the computing environments. The universal access is focused not only on global resources, but also on local resources for achieving accessibility and control.

In computation grid environments, the Low Cost Services provide low cost services for acceptance and large scale utilization. These services are aggregated to define the type of computations achieved for representing the outcomes and usage based on computations performed. These services can be broadened according to the technological developments for wide deployments.

2.1.3 Scheduling Grid Resources

The diverse grid resources are scattered over the globe to several users concurrently. For the proper handling of the user jobs and best utilization of these resources, the jobs must be scheduled to one among those resources for satisfying the demands accordingly. The grids are autonomous which changes in accordance with time due to which various factors like access to newer resources, failure of resources, newer requests and job deadline are focused. These factors require reallocation of jobs for executing them in the resources.

The traditional techniques faced many problems in job allocation and resource scheduling in autonomous and scattered environments, but the problem in autonomous system is more complex and remains a challenge, and so, the traditional techniques cannot perform well. The solution can be formed by using suitable techniques for handling the grid resources and for executing the jobs by considering the demands of the users. The following points are very important during the scheduling of the jobs to the resources in grid computing environment.
A proper knowledge about the computational load for executing each job based on resource allocation.

The computational capacities for performing computation using the allotted resource.

A prior calculation of load for every resource is required.

Calculating the maximum time for job completion.

The following are the most important terminologies used, while scheduling the jobs to grid resources.

- The primary time to begin processing a job.
- The total time required for processing to finish a job.
- The exact time to begin processing a job.
- The amount of time required to finish the processing of job.
- The time taken by a job to complete its execution.

The resource allocation describes the architecture for managing the resources and expandability of the system. The allocation can be classified as centralized, hierarchical or decentralized.

Centralized Allocation technique allows the submission of all jobs to a single resource scheduler liable to allocate the job based on the resources available. Here all the information for job allotment is obtainable on a single position in such a way that the allocation decisions are the finest, but the technique does not offer scalability over grids.

Decentralized Allocation technique does not require a centralized scheduler, because the allocation is achieved by the requestor of the resources and the owners uniquely. This technique, offering scalability and best suited
over grids, the focus should be made upon every single scheduler, since the coordination is required for making decisions related to allocation and the obtained results are not the finest.

Hierarchical Allocation technique allows the scheduler to appear in a hierarchy. The resources at higher level are allocated at higher levels and lower sub-levels are allocated at lower levels of the allocation hierarchy, combining the other two techniques.

### 2.1.4 Fault Tolerance

The computational grids focus on fault tolerance, because the grids are a collection of autonomous nodes, and so there is no guarantee for reliability. It is necessary that the grid must work even during the failure of one or more resources, and the techniques for identifying and correcting faults are needed to avoid the occurrence of the faults within the grid. The fault tolerance is conserving the liberty of estimated services despite the errors caused because of the faults within the system. The identification and correction of errors along with stable faults are listed and eradicated, allowing the system to deliver the requested services.

The grid is a collection of scattered and variable systems involving diverse resources positioned in a global domain providing the fault tolerant resource services. The diverse nature of the grid computing literally means that the applications within this environment will continue to perform even in the situation of failures. Without a technique for tolerating faults the grid, is not capable enough to continue its work during the failure of one or more resources or during any collapse of the entire system. During the occurrence of a failure, the jobs must be reassigned rapidly and dynamically from the user's perspective.
The technique for tolerating the faults is required for the grids to prolong the execution while there are faults. During the occurrence of faults, it is important to perform the below stated tasks.

- Quickly deciding the location of faults precisely.
- Segregating the rest of the network from failures to prolong working without intrusion.
- Adjusting the networks in a way to reduce the operational collisions without failed components.
- Replacing the failed components for returning the networks to its preliminary state.

The architecture of the basic grid fault tolerance is depicted in Figure 2.2. The system is categorized into five different components as discussed below.

![Figure 2.2 Fault tolerant architecture](image-url)
Grid Interface allows the users to present their jobs to the grid system for execution.

Allocator is responsible for choosing the best possible resource for executing a job and the decision of the allocator solely depends on the QoS requirements of the users.

Information Server holds the entire information about all the resources in the grid, including the speed of computations, available memory, workload and so on. This component provides all the information necessarily required for the scheduler to schedule the jobs by suitably allocating the resources.

Fault Handler is responsible for identifying the failure resources and approximating the necessary information for tolerating faults.

Grid Resources is a collection of scattered and variable systems involving diverse resources positioned in global domain providing fault tolerant resource services.

As the faults are inevitable, precautions are to be taken so that the performance of the system may not be degraded because of resource faults. The fault tolerance system plays a vital role in addressing the above issue in such a way that on receiving the jobs from the user, the Allocator consults the Information Server each time, before allocating the resources. Since the Fault Handler notifies and frequently updates the faulty resources in the Information Server, it in turn restricts the allocating resources to those having more history of faults, thus tolerating the faults. However, if the job size does not match the capacity of the resource, there may be possibilities of fault occurrence.
2.1.5 The Grid System

The technology synchronizes the huge distributed resources and utilizes them to solve the problems in several dynamic groups. This technology differs from any other technological development, like internet and scattered networks among the organizations and peer-to-peer computations. The major issue to be addressed is to improve the QoS requirements, managing information, job assignments, resource access, safety and overall system performance. The grid infrastructure supports the diverse user applications in solving some specific problems which pose a major challenge to the system.

Various types of resources are owned by the grid systems, belonging to diverse independent organizations with diverse resources and policies and due to this services offered by grid, their applications differ more in resource behaviour than estimated. As the technology offers different services to different users, the ultimate goal of the grid system is to provide the required services to the users to gain access to the services through suitable resources, without degrading the overall performance of the system.

Apart from the policies and the services offered, the ultimate objective of the grid system is to aggregate the conventional resources with their hardware components, operating environments, limited resource organization and safety to resources, which is achieved through the creation of the Virtual Organization (VO), allowing the identification within the scattered resources distributed for recognizing the location of the resources. The grid environment delivers a set of protocols, middleware, toolkits and services that are built on top of these protocols. The resources are diverse in grid architecture, making the security and resource accessibility a major challenge. The utilization of the global and local resources differs in
accessibility and ability along with their policies, diverse hardware and software components and its underlying platforms.

2.2 AN OVERVIEW OF SERIAL JOB SCHEDULING STRATEGY

The grid technology provides services to the users as per the demands without compromising the QoS requirements. So, in one of the existing Serial Job Scheduling Strategies, the clients submits their jobs to the resource for processing the demands. Here the resources resembles a collection which holds computational elements capable enough to accept the arrived jobs and for scheduling them to the resources as shown in Figure 2.3.

![Figure 2.3 Allocation of incoming jobs to resources](image)

Initially the jobs arrive from the clients, containing a mixture of all kinds of jobs which demands processing, allowing the consumption of resources as per the job length, and moreover, there may be a situation where the resources are required by some other jobs too, during which that particular job has to wait in the line for accessing those resources. This takes additional time for the waiting of jobs for the resources and thus increasing the execution time and proportionally degrading the system performance.
The resources resembles a collection which should be classified accordingly for processing the incoming jobs in order to overcome the errors caused due to resource failure or unavailability of resources. Only a classified set of resources can handle the jobs based on its resource classification. So, these issues can considerably degrade the overall performance of the system, and maximize the execution time of the jobs and problems with jobs waiting in the line for further processing. The evaluation is carried out by taking the below mentioned criteria into consideration.

- The total number of arriving jobs
- The total number of available resources

The grid accepts the incoming jobs for processing, considering different metrics during the execution and this determines the overall performance of the system. Scheduling serial jobs to the resources determines the order of executing the jobs and time consumed by them to execute.

The concentration is made not only towards organizing jobs, allocating tasks and execution time for them, but also on the process that can create faults due to resource unavailability. Moreover, neither all the resources are capable of performing all the jobs, nor is it not possible to pre-estimate the time taken for a resource to execute a job and so, suitable alternative techniques are to be designed to address all the above challenges.

2.3 SYSTEM MODEL

Figure 2.4 depicts an effective and potent grid environment holding a variety of resources in a large scale, and resource capacity varies from one another based on the tasks. The main goal of the proposed system is to make a proper usage of all the inactive resources to attain load equalization, using an effective job allocation technique for improving the performance of the grid.
systems, highlighting the importance of the fault tolerance mechanisms to be included within the system.

![Diagram of Grid System Model](image)

**Figure 2.4 Grid system model**

The resource selection and combination should be focused upon because of the distribution of the equal sized tasks (i.e.) the light weight jobs to the available resources enhances the utilization of resources whereas the jobs with high potential termed as heavy weight jobs tends to degrade the performance of the system. The main objective here is to minimize the communication time and maximize the CPU usage time to enhance the overall performance of the system. The grid jobs are executed by the grid systems as stated below:

- The users of the grid present their jobs to the grid scheduler by considering the QoS requirements like job execution time, operating system type and so on.
• The grid scheduler accepts the jobs and immediately allocates the tasks to the optimal grid resources, based on their availability.

• The results obtained after execution are given back to the users on the successful completion of the tasks.

2.3.1 Job Scheduling Mechanism

The scheduling allows rapid job assignment to the global, diverse and autonomous resources, and should be a rapid process without compromising the metrics like time taken for response, accessibility and delay. The job completion is dependent on an independent resource and no cooperation exists between the jobs. The job arrival rate decides the environment and the processing capacity of every individual resource in the system, because they are dissimilar in nature, possessing varied execution time for every job on diverse machines.

In the grid environment the following parameters are given importance:

• The number of unique tasks for allocation: Once the jobs are received from the user, it is necessary to measure the size of the jobs in terms of Million Instructions Per Second (MIPS) for identifying whether the job is fine grained job or a normal job. Unique tasks represent the fine grained job category that can be allocated to any of the free resources in the pool, for all the resources are capable of executing all such tasks.

• The number of diverse resources (machines): Once the number of fine grained jobs are identified, then, it is
important to know the number of resources that can execute those tasks to help the system improve the performance by assigning tasks simultaneously.

- The time taken by the job for execution on each machines: This gives the Maxspan and Flow Time of the tasks assigned, as the efficiency of the system and the overall performance are evaluated.

  MIPS basically refer the size of the incoming job with total number of instruction count. The threshold is set to decide whether the incoming job is fine grained job (light weight) or a normal job. Algorithm for finding Maxt has also been included. In the simulation environment, capacity of each resource is identified through its initial simulation test, which is always maintained in the scheduler and the average computational power of the grid resources also calculated. Subsequently, the resource with less execution capacity is identified and set as Maxt. When MIPS is less than the threshold Maxt, then the job can be considered as fine grained. This indicates that any grid resource can execute this job and assigned to any of the available grid resource which can successfully be executed. The work assumes that the computation time for each and every job is based on a fixed threshold value.

  \[
  \text{MIPS} < \text{Max}_t \\
  \tag{2.1}
  \]

  In case the MIPS of an executing job is less than the fixed threshold as in Equation (2.1), then the job is a fine grained one otherwise it is a normal job. The fine grained jobs generally termed as light weight jobs, consist of MIPS less than the fixed threshold Max, and are fixed based on the lowest capacity resource among all the available grid resources with the capacity to process the fine grained jobs. The decisions are made on the type of the job which requires an efficient scheduling algorithm. This means that any job referred to as fine grained can be assigned to any available resource meaning
that all the resources in the pool have the capacity to process the job immediately.

The normal jobs are the jobs with greater MIPS than the fixed threshold which means that the job requires a higher capacity resource to process it. With a highly autonomous and scattered nature of the grid environment, the resource scheduling techniques become a challenging task and the resources are grouped based on the following characteristics:

- Communicative services without having a resource manager.
- The grid resource observing the component containing the information about the current accessibility to free resources within the collection of resources and the resource capacities are depicted as in Figure 2.5.

![Figure 2.5 Scheduling mechanism using monitoring system](image-url)
The information, present inside the grid resource monitoring component is capable enough to update its own contained information periodically and during the entry of heavy job loads into the resource collection as per the allocation by the resource scheduler. This mechanism is purely dependent on the Grid Resource Information Management (GRIM) prototype and Grid Resource Information Representation (GRIR) protocol.

The job allocation is most important in a grid resource management system. The request for the job is received by the scheduler based on which the suitable resources are selected for executing that requested job. The work focuses on job scheduling based on Maxspan and Flow Time and during the job scheduling of the fine grained jobs, the so called the light weight jobs requires less time for the execution compared to that of normal jobs.

The estimated time for completing a job ‘i’ on resource ‘j’ is given by Equation (2.2).

\[
\sum C_{i,j} + W_i \tag{2.2}
\]

The independent jobs are aggregated together, forming a meta-job for deriving static jobs where the number of jobs ‘i’ and the number of resources ‘j’ are already known. \( \sum C_{i,j} \) describes the expected time for executing a job ‘i’ on a resource ‘j’ and \( W_i \) gives the time taken for completing all the previous jobs assigned to that resource. The main goal is to reduce the maxspan which is shown in Equation (2.3).

\[
\text{Maxspan} = \text{Max} \{ \sum C_{i,j} + W_i \} \text{ where } i=\{1,2,\ldots,n\} \tag{2.3}
\]

where, \( \text{Max} \{ \sum C_{i,j} + W_i \} \) represents the resource accessibility time i.e. the time taken by the resource ‘j’ to complete the earlier jobs.
It is also necessary to reduce the Flow Time by reducing the average job completion time and the Flow Time is defined as the aggregation of all the time required for the resources for completing all the jobs and is calculated as in Equation (2.4).

\[
\text{Flow Time} = \sum_{i=1}^{\sigma} \sum C_{ij} \tag{2.4}
\]

where \( \sum_i \) denotes the time for completing a job ‘i’ and ‘i’ represents the expected time for executing a job using a resource ‘j’ and \( \sum C_{ij} \) represents the expected time for executing a job ‘i’ on a resource ‘j’. As per the Equations (2.3) and (2.4) the Maxspan and Flow Time of the allotted jobs are executed in accordance with the execution time. The main objective of the resource scheduler is attained by obtaining the minimized Maxspan and Flow Time.

In due course of assigning the jobs, there are possibilities for the computational grid environment to face certain challenges as discussed below:

- During the occurrence of the faults within a grid resource the job is re-allocated to the next best available resources that fails to satisfy the user’s QoS requirements and the job consumes more time to re-execute the tasks.

- The computational grid provides assured resources which allow the job to be executed within the time, but some faults within the resources may lead the job to failure.

The grid scheduler selects that particular resource, because the grid environment agrees to meet the prerequisites of the user for executing the tasks over grid environment without compromising the QoS requirements for the jobs.
The grid system is experimented with the adaptive job scheduling strategy for a given set of jobs and set of resources to evaluate the system performance. The algorithm for adaptive job scheduling specifically developed for scheduling the light weight job is given below:

**Algorithm 1: An Adaptive Job Scheduling for scheduling fine grained jobs.**

Let \( R = \{ R_1, R_2, \ldots, R_n \} \) for all \( n \in N \) // Setting Threshold \( \text{Max}_t \)

\( J = \{ j_1, j_2, \ldots, j_n \} \) for all \( n \in N \) and \( j_i = 2^i \) for all \( i \in N \)

IF \( R=1 \) job are executed by all resources

ELSE \( R=0 \) all resource have not executed job

IF \(( R_i \text{ executes } j_i )\) THEN \( R=1; \) where \( i, j \in N \), \( R_i \in R \) and \( J_i \in J \)

\( R_i \) do not executes \( J_{i+1} \) then \( R=0; \) Again \( R_i \) do not execute \( j_{i+2} \) ; \( R=0; \)

Then set Max\(_t\)=\( j_i \);

All jobs ‘i’ for allocation to all resources ‘j’ are noted

\[ \sum C_{ij} = \sum C_{ij} + w_i \text{ is calculated} \]

End for

\( \sum C_{ij} \) is collected for selecting the best possible resource for execution

\( \text{Max} C_{ij} \) is calculated

Let task ‘i’ on ‘j’ be stored

Let \( \text{Max} C_{ij} = \sum C_{ij} \) be calculated //\( \text{No. of Resources} \)

For all resource ‘j’ if \( (C_{ij} \leq \sum C_{ij}) \)

Let the stored jobs be chosen as fine grained
End for

For all selected jobs ‘i’ within the resource ‘j’

if \( C_{i,j} \geq \sum C_{i,j} \) 

Let new \( \sum C_{i,j} = C_{i,j} \) be calculated

if new \( (C_{i,j} \leq \sum C_{i,j}) \)

\( \sum C_{i,j} \) is collected for selecting the best possible resource for execution

End for

\section*{Description}

The job ‘i’ are scheduled to the resource ‘j’ where the execution time \( C_{i,j} \) for job ‘i’ on the resource ‘j’ should be the minimum. The proposed system considers only the light weight jobs as they consume only less time for execution than the normal jobs. In case the resource ‘j’ encounters a normal job, it does not consider it since it can degrade the performance of the system. All the jobs are collected and grouped for identifying the best possible resource for execution in order to minimize the execution time of the jobs and to improve the overall system performance.

Once the scheduler receives the incoming jobs from the user, one of the functions within scheduler named ‘instruction count’ which is a function of GSL library is invoked to find the total instruction count of that job. Based on the number of instruction count, the job is either classified as fine grained (light weight) or a normal job. If the job is fine grained, it can be allocated to any of the available resources and this algorithm is perfectly suitable for fine grained jobs and hence the algorithm is adaptive to fine grained job and normal jobs.
2.3.2 Fault Tolerance

The fault tolerance is to predict the correct behaviour of resources. The failure happens, when a system on execution diverges from its specific behaviour, which literally means that the faults are the origin for failures, which can be diagnosed using errors occurring within the system because they serve as symptoms for faults. All the errors may not result in faults, but in some cases it may result in multiple faults occurring within the system. It is also to be noted that a single error is also capable of causing several faults.

The proposed technique addresses the fault tolerance within the computational grid environment because, the main goal is to obtain better results for user’s demands even during the faults. For obtaining better results, parameters like Overall Time to Reply (OTR), job running time and the amount of jobs completed within time are measured during the failures also. It is noted that, when the faults happen within the grid environment, while allocating resources to normal tasks, it tends to consume a large amount of time to execute. After encountering such faults, the information relating to such grid resources are updated. The information about the fault occurrences are used during the job test summit, before scheduling jobs to the resources over grid.

The fault directory of all the prevailing resources are preserved and updated. The fault directory gets increased each and every time the resource is scheduled to normal tasks consuming a higher execution time and may also result in resource failures. Consequently, the fault directory of a resource does not get increased during the allocation of job with minimum execution time i.e. to equal sized segments. Hence, the algorithm is best suited for fine
grained jobs or adaptive jobs which exactly suits the resources mapped and executed for improving the system performance.

2.3.3 Components of Fault Tolerance Strategy

Figure 2.6 depicts the communication among various components in a computational grid environment in the proposed scheduling and the fault tolerant mechanism. The proposed technique adds the scheduler for minimizing the job response time, and the fault tolerant technique minimizes the time consumed for normal jobs. But, actually it resumes the normal tasks from the last hoard test summit for completion.

![Figure 2.6 Components of fault tolerant strategy](image)
The components and the working mechanism of the proposed technique are discussed below:

The scheduler is one of the major components in the grid environment which initially receives the job from the grid users, and chooses the available resources in accordance with the information obtained from the GIS. The GIS maintains all the information about the grid resources and the scheduler seeks the help of GIS to know about the available free or less loaded resources for every job allocation and execution. The resources are matched to the job requirements after which the time for execution is estimated for every matched resource. The resource selector chooses the resource with a reduced response time and finally, every individual job is delivered to the test summit manager on one by one basis.

The test summit manager receives the scheduled jobs from the scheduler, and assigns test summit actively, based on the rate of failed resources upon which it is scheduled. The final stage being the submission of these jobs to the resources, the test summit manager collects the task completion message or job failure message from the grid resources and acts in accordance with it. In case of job failure during the execution, the job is reallocated from the last test summit, instead of continuing the execution from thereon.

The test summit manager creates a test summit during every individual failure, which reports the status of the job to the test summit server. This information is saved and utilized during the failure of the job or resource. For every independent job, the test summit server removes the results obtained from earlier test summits for hoarding the newly obtained test summit results.
Fault Directory component retains the fault directory value for every resource, which represents the resource failure rates. The fault directory gets incremented every time, when the resource is scheduled to a normal job and also due to the failure in resource. The fault directory is decreased upon the successful completion of every scheduled job within the time limits.

Finally the job dispatcher receives the successfully executed jobs from the test submit manager and dispatches it back to the grid users through the job dispatcher. The time elapsed between the job submission by the user and the time which the job is received back by the user in a grid system is defined as the total processing time of the job and is represented as an overall time to reply which is discussed below.

2.3.4 Calculating Overall Time to Reply (OTR)

The OTR comprises communication time of the input and output information to and fro from the resources, job waiting time in the resource waiting line, and the job service time allocated to the resource. The OTR is defined in Equation (2.5).

\[
\text{OTR} = \text{Communication Time} + \text{Job Waiting Time} + \text{Service Time} \tag{2.5}
\]

The expected communication time for a task ‘i’ from the scheduler to the resource ‘j’ is represented in Equation (2.6).

\[
\text{Communication Time}_{ij} = \frac{s_i}{b_{s\rightarrow j}} \tag{2.6}
\]

Here ‘\(s_i\)’ represents the job size ‘i’ and \(b_{s\rightarrow j}\) represents the bandwidth utilized by the network between the scheduler denoted as ‘s’ and the resource as ‘j’.

\[
\text{Job Waiting Time}_{ij} = \sum_{j=1}^{\text{Waiting Line}} \text{Service Time}_j \tag{2.7}
\]
The job waiting time for ‘i’ in the waiting line of the resource ‘j’ is calculated by adding all the service time of jobs in the waiting line of ‘j’ assigned to that particular resource before the incoming job ‘i’ as in Equation (2.7).

\[
\text{Service Time } _{ij} = \frac{\text{Instruction Count}_i}{\text{Speed}_j}
\]  

Service Time \(_{ij} = \text{Instruction Count}_i / \text{Speed}_j\) \hspace{1cm} (2.8)

The service time for job ‘i’ within the resource ‘j’ is defined as the instruction count in the job ‘i’ divided by the ‘j’ processing speed of the resource ‘j’ as in the Equation (2.8).

Upon the submission of each grid job, the successfully completed jobs are returned to the users and the total time taken for executing each job is also computed. It is also necessary to update the flaw directory or fault directory immediately, so that it further enables to allocate resources to the next incoming jobs by knowing the status of the resources, including the fault status. The fault directory is updated based on the test summits, and the procedure for the same is given below.

**Algorithm 2: Updating Fault Directory based on Test Summits**

**Input:** \(C_{i,j} \geq \sum C_{i,j}\)

**Output:** Test Summit are created at intervals

Fd : Fault directory of the chosen grid resource

Fd (i) where, \(i = 0, 1, 2 \ldots n\) and \(F_d (0) < F_d (1) < \ldots < F_d (n)\) where \(F_d (n)\) are allowable fault directories based on the fixed threshold value.

1.  **If** \((F_d < F_d(n))\)

   The test summit manager receives the job fault message from the resource then,
a) An increment message is sent to the fault directory of the resource that fault has occurred during the job processing.

b) The test summit server receives a message for tracking the results of test summit for a task.

c) In case of the presence of a test summit in the test summit server, the remaining part of the task received after the previous test summits is given to the scheduler for re-scheduling.

End If

GOTO STEP 3.

2. If (Fd == 0) Then,

a) A message is sent to decrement the fault directory of the resource that the scheduled job is completed.

b) The details of the finished task are sent to the scheduler.

END if

GOTO STEP 3.

3. END

Description

The larger job ‘i’ executing on a resource ‘j’ is given as input because the normal job has the tendency to consume large amount of time for execution, that creates errors causing faults since the resource are incapable of handling the normal tasks, which makes the resources unavailable to the newer jobs. The information is updated to the fault directory, a test summit is created on that point and the successor jobs are discarded and again sent to the scheduler for re-scheduling. In case, the job does not consume more time for
execution, it best suits and the scheduled job is completed without any hindrance and the entire flow of the activity is shown in Figure 2.7.

![Control Flow Diagram](image-url)

**Figure 2.7 Control flow diagram**
2.4 EXPERIMENTAL STUDY

The implementation is performed within a grid environment using GridSim tool and the results are obtained.

For implementation, the number of grid jobs scheduled varies between a minimum of ‘16’ jobs to a maximum of ‘512’ jobs. The obtained results solely dependent upon the types of jobs allocated to the grid system which is a collection of autonomous and diverse resources and there are some features to be considered. The resource types are not static and they tend to change in accordance with the user’s demands. The simulation considers ‘16’ resources for executing grid jobs and it tend to change according to the user’s demands. The execution time and overall job finishing time are purely based on the jobs scheduled to the grid system.

The varying jobs from ‘16’ to ‘512’ were executed and the amount of resource remained unchangeable. The number of jobs and the corresponding time taken for executing those jobs are tabulated in Table 2.1.

**Table 2.1 Execution time of adaptive job scheduling**

<table>
<thead>
<tr>
<th>No. of Tasks</th>
<th>Execution Time (Milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>1717</td>
</tr>
<tr>
<td>32</td>
<td>3433</td>
</tr>
<tr>
<td>64</td>
<td>3850</td>
</tr>
<tr>
<td>128</td>
<td>7209</td>
</tr>
<tr>
<td>256</td>
<td>13944</td>
</tr>
<tr>
<td>512</td>
<td>21889</td>
</tr>
</tbody>
</table>

The obtained results clearly states that the equal sized jobs are executed faster than the uneven sized jobs and hence it nullifies the idle
waiting time for the resources between the maximum and minimum job execution time in all the cases. It is clear that, when there is a decrease in the number of jobs then, the total job execution time is minimized and there remains a huge deviation between minimum and maximum job execution time.

Figure 2.8 shows the execution time of the tasks that are carried out in the simulation. The simulation test is conducted for different number of tasks ranging from ‘16’ tasks to ‘512’ tasks and the corresponding execution times of jobs are also plotted. The results obtained from the analysis explain that the normal tasks consumes higher job execution time and the equal sized tasks consumes lower job execution time as it nullifies the idle waiting time of the jobs for the resource. The nullified idle waiting time shows a great variance in the total execution time of all the tasks which brings a significant improvement in the performance of the system.
The proposed system is simulated using GridSim tool considering a sample with a maximum of ‘512’ jobs and ‘16’ resources and the system performance is evaluated. The proposed technique is compared with the conventional Serial Job Scheduling technique on the basis of the total execution time, processing time, as well as the resource accessibility. The simulation is performed for three different cases for evaluating the resource accessibility:

Case 1 represents the system with high workload, Case 2 represents the system with low workload and Case 3, the system with medium workload.

Table 2.2 Comparison based on resource accessibility

<table>
<thead>
<tr>
<th>Cases</th>
<th>Resource Accessibility (in %)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Serial Job Scheduling</td>
</tr>
<tr>
<td>1</td>
<td>75</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td>3</td>
<td>85</td>
</tr>
</tbody>
</table>

Figure 2.9 Comparison based on resource accessibility
Figure 2.9 tabulated using Table 2.2 clearly depicts that the proposed technique achieves better resource utilization than the conventional technique.

The variation in the total execution time of the jobs varying between ‘16’ jobs to ‘512’ jobs using the Serial Job Scheduling and Adaptive Job Scheduling is shown in Table 2.3.

**Table 2.3 Comparison of serial job scheduling and adaptive job scheduling**

<table>
<thead>
<tr>
<th>No. of Tasks</th>
<th>Execution Time (Milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Serial Job Scheduling</td>
</tr>
<tr>
<td>16</td>
<td>6040</td>
</tr>
<tr>
<td>32</td>
<td>12078</td>
</tr>
<tr>
<td>64</td>
<td>16963</td>
</tr>
<tr>
<td>128</td>
<td>32282</td>
</tr>
<tr>
<td>256</td>
<td>61129</td>
</tr>
<tr>
<td>512</td>
<td>122937</td>
</tr>
</tbody>
</table>

In Figure 2.10, the simulation results of both the existing and proposed techniques are compared for varying loads ranging between 64 tasks and 512 tasks. As the number of tasks increases gradually, the performance of the Adaptive Job scheduling also increases with its execution time.
Based on the observations from the obtained results evaluated in terms of overall execution time and resource accessibility, it is clear that the time required for the execution is much reduced thus subsequently reducing the makespan and the processing time accordingly.

### 2.5 CONCLUSION

The chapter focuses on reducing the overall time taken for executing the jobs by proposing an Adaptive Job Scheduling algorithm specially meant for scheduling fine grained jobs or light weight jobs within the grid environment. The incoming user jobs are scheduled to the appropriate resources based on their processing speed, operating system and so on. This scheduling allows the job to be executed in its suitable environment by assigning a suitable resource among the available list of resources.
The technique focuses on reducing the idle waiting time of the jobs to the resources by considering only the light weight jobs which require only a minimum time for the execution and, in case, a normal job is encountered it consumes higher execution time and the access to resource is not possible during which fault tolerance techniques are employed for addressing the problem and providing the best possible solution. The simulation result of the proposed system is compared with that of the existing scheduling techniques and it is clear that it accomplishes a significant performance improvement.