BI-OBJECTIVE FAULT TOLERANT MODEL FOR WORKFLOW TASK SCHEDULING ON GRIDS

Sridevi S [1], Golda Jeyasheeli P [2]

[1]PG student, Dept of CSE, Mepco Schlenk Engineering College, Sivakasi, India.
[2]Assistant Professor, Dept of CSE, Mepco Schlenk Engineering College, Sivakasi, India,
Email:L sridevi5983 @ gmail.com

Abstract

The spur of Grid computing is to aggregate the power of widely dispersed resources, and provide non-trivial services to users. In attempts to utilize a diverse set of resources in grids proficiently, scheduling has been made. The primary intention of scheduling is the minimization of application completion time; however, they may lead to the usage of excess and redundant resources. Our algorithm performs the scheduling by accounting for both completion time and resource usage. Since the performance of grid resources changes dynamically and the accurate estimation of their performance is very difficult, our algorithm incorporates rescheduling to deal with unforeseen performance fluctuations effectively. Also, fault tolerance is an essential part of the grid. In Grid environments, execution failures can occur for various reasons such as network breakdown, failure or non-availability of required resources. Fault tolerance can be achieved in grids by Over provisioning and Check pointing techniques. Since, over provisioning violates the resource usage control, check pointing strategy is implemented in our proposed method.

Key Words: — check pointing, fault tolerance, grid scheduling, make span, resource optimization

I. INTRODUCTION

COMPUTATIONAL grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities. Recent developments in grid infrastructure technologies make it possible to execute large and distributed applications on it. Many of these applications fall in the category of interdependent task model. These classes of applications are generally referred as workflow applications, which are often represented as DAGs (Directed Acyclic Graph) with nodes representing tasks and edges representing dependencies.

Scheduling algorithms in grid platforms generally focus on the minimization of application completion time. However resource usage also plays an essential role in the performance of grid. Scheduling on minimum make span with minimal use of resources yields optimal solution. To achieve non-trivial services from grids, an efficient fault tolerant strategy becomes vital. Hence our paper focuses on all these three aspects: Make span, resource usage, fault tolerance effectively. In this paper, we address the problem of scheduling workflow applications in grids and propose by a novel semi-dynamic scheduling heuristic, referred as *adaptive bi-intentional fault tolerant scheduling* (ABIFTS) algorithm.

ABIFTS algorithm statically generates the initial schedule using an evolutionary technique. It adapts dynamically as the performance of resources changes. Dynamic rescheduling also occurs where there is any fault in the scheduling process. ABIFTS can generate an optimized schedule even in presence of resource failure.

The main strengths of the proposed system are: 1) The good quality of output schedules with minimum resource usage. 2) The adaptability to resource performance fluctuations. 3) The ability to generate efficient schedule in presence of resource failure. The first strength is achieved by iteratively improving an initial random schedule using a branch-and-bound technique and mutation. The second strength is made possible by using a rescheduling strategy. Specifically, ABIFTS initiates a rescheduling event if a job finishes later than expected and its late completion results in an increase in the overall application completion time. Rescheduling occurs with all of the remaining jobs that are not running. The third strength is achieved by means of check pointing strategy. Check pointing is a technique for inserting fault tolerance into computing systems. It consists of storing a snapshot of the current application state and uses it for restarting the execution in case of failures.

II. RELATED WORK

Since workflow scheduling in grids is similar to the conventional task scheduling problem in tightly coupled heterogeneous computing systems, some well-known task scheduling algorithms like HEFT [2] have been adopted and modified for grid scheduling. Most modifications deal with the dynamic nature of grids. Two approaches adapted from traditional task scheduling algorithms fall into the look-ahead category and just-in-time category. The major difference between these categories is whether scheduling decisions are made before the actual job dispatch or at the time any ready jobs are identified. For look-ahead approaches, the acquisition of accurate performance information on resources plays a critical role in their decision making. The drawback of just-in-time approaches is the loss of timely data transfers. For example, provided that a job has three predecessors and they complete at different times, the data transfers from these predecessors to the job start at the time the last predecessor completes its execution. Here, the time between the times the first two predecessors are completed and the time the last predecessor is completed is wasted.

III. PROPOSED WORK

A. System Model

The grid in the current study consists of a number of sites each of which has m computational hosts. Each site is an autonomous administrative domain that has its own local users who access the resources provided by it. These sites are connected with each other through a wide area network. Hosts are composed of both space shared and time-shared machines with various processing speeds. These resources are not entirely dedicated to the grid. They are used for both local and grid jobs. We assume that hosts in the same site are able to access each other's data repositories as if they were accessing their own, i.e., a set of data repositories in a site can be represented as a single data repository. This assumption is made because in general, a site connects its hosts through a high-bandwidth local area network.

The availability and capacity of resources, e.g., hosts and network links, fluctuates. Therefore, the accurate completion time of a job on a particular host is difficult, if not impossible, to determine a priori. Moreover, the job may fail to complete due to a failure of the resource on which it is running. At that time, fault tolerant mechanism works. i.e., rescheduling occurs from the check point.

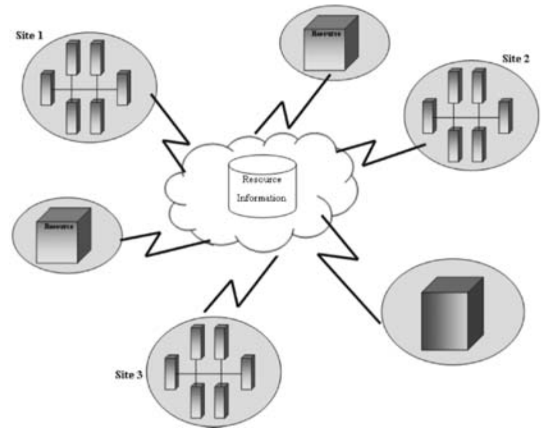


Fig. 1. Grid System Model

B. Application Model

Workflow applications are essentially the same as typical parallel programs, with one exception: a workflow application consists of a set of interdependent applications (not partitioned tasks of a parallel program). Like conventional parallel programs, workflow applications can be represented by a DAG. A DAG consists of a set V of v nodes and a set E of e edges. A DAG is also known as a task graph. The nodes usually represent jobs of a workflow application, and the edges usually represent precedence constraints. An edge between job *nsubi* and job n_j represents the interjob communication. Specifically, the output of job n_i must be transmitted to job n_j for job n_j to start its execution. A job with no predecessors is called an entry job, n_{entry} an exit job, n_{exit}, is one that has no successors. Among the predecessors of a job n_i, the predecessor that completes the communication at the latest time is the most influential parent (MIP) of the job denoted as MIP(n_i). A job is called a *ready job* if all of its predecessors have been completed. The longest path of a task graph is the critical path (CP).

The weight on a job n_i, denoted as w_i, represents the computation time of the job. The computation time of a job ni on a host h_j is w_{ij}. The weight on an edge, denoted as c_{ij} represents the communication time between two jobs n_i and n_j. However, the communication time is only required when two jobs are assigned to different hosts. In other words, the communication time when they are assigned to the same host can be ignored, i.e., it will be zero.

The average computation and communication times of a job n_i are $\overline{w}$ and $\overline{c}$, respectively. The former is the average computation time of job n_i over all of the hosts in a given system. The latter is the average communication time between job n_i and its successor jobs.

C. Prioritization of jobs

Each job n_i is associated with its scheduling priority based primarily on inter job dependencies. The priority of job n_i can be computed using different job prioritization methods. One of the common methods is b-level computation.

The b-level of a job is computed by adding the computation and communication times along the longest path from an exit job in the task graph (including the job).

The b-level value of a job n_i is defined by

$$b\,(n_{i)} = \overline{w_i} + \max_{n_j \in \text{imed} - \text{succ}(n_i)} \left\{ \overline{C_{i+j}} + b(n_{j)} \right\} \qquad \text{... (1)}$$

where *imed_succ(n_i) is the set of immediate successor jobs of job n_i. For an exit job, its average computation time is its b-level value.*

IV. SCHEDULING METHODOLOGY

A. Workflow Scheduling Problem

The scheduling problem addressed in this paper is the scheduling of a set of interdependent jobs, comprising a workflow application, onto a set of heterogeneous hosts dispersed across multiple sites in a grid. The primary goal of this scheduling is to make as many appropriate job-host matches as possible so that the makespan, also called the schedule length, of a workflow application can be minimized with as little resource usage as possible. The makespan is defined as the amount of time taken from the time the first job starts running to the time the last job completes its execution. The resource usage is defined as the total amount of resource (both computing and network resources) times used for running a given workflow application. We assume that the cost value between one unit of computation time and one unit of communication time is equal; that is, the cost ratio is one.

B. Calculation of Start and Finish times

The earliest start and finish times of a job n_i on a host h_j are defined as

$$\text{EST}\,(n_i, h_j) = \begin{cases} 0 & \text{if } n_i = n_{entry} \\ \text{EFT}\,(\text{MIP}\,(n_{i)}, h_{k)}, h_k \in H + C_{\text{MIP}(n_{i)},j} & \text{otherwise} \end{cases} \qquad \text{... (2)}$$

$$\text{EFT}\,(n_i, h_j) = \text{EST}\,(n_i, h_j) + W_{i+j} \qquad \text{... (3)}$$

where $h_k \in H$ is the host executing MIP(n_i). The latest start and finish times of a job n_i on a host h_j are defined as

$$\text{LST}\,(n_i, h_j) = \text{LET}\,(n_i, h_j) - W_{i+j} \qquad \text{... (4)}$$

$$\text{LFT}\,(n_i, h_j) = \begin{cases} \text{EFT}\,(n_i) & \text{if } n_i = n_{exit} \\ \min\limits_{n_i \in \text{succ}(n_{i)}} \left\{ \text{LST}\,(n_k, h_m) - C_{i,k} \right\} & \text{otherwise} \end{cases} \qquad \text{... (5)}$$

where succ (n_i is the set of successor jobs of job n_i, and h_m is the host executing job n_k.

C. Check Pointing Technique

Check pointing technique in scheduling can be explained from the given diagram as:

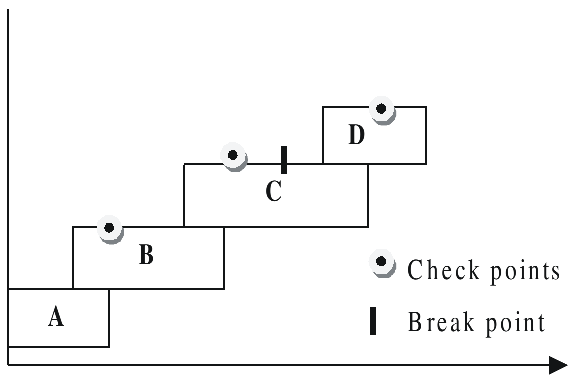


Fig. 2. Use of Check points in the scheduling process

The various check points in the scheduling process are used to save the current status of the process at various points. Assume that the system fails at the Break point. Then what will happen if no fault tolerant mechanism is used? The entire system fails & hence degrades the performance of grid. But if check points are used, the system recovers from the last check point before the occurrence of fault (break point) and then rescheduling dynamically with the available resources except the failed resource.

V. ABIFTS ALGORITHM

Unlike many other workflow scheduling schemes, we consider both makespan and resource usage to be equally important and take this into account in our scheduling model. Efficient resource usage is crucial in grid scheduling because 1) a grid consists of multiple sites administered by different entities that use their own resources for other tasks beside the grid jobs and 2) due to the fluctuations and uncertainty surrounding sites in a grid system, lower resource usage—not necessarily the minimization of the number of resources used, rather the minimization of resource time— means lower overall variance in the expected completion time (makespan) of an application. Rescheduling is another technique adopted to increase the practicality of ABIFTS.

To start with, ABIFTS generates a random schedule; this initial solution undergoes the manipulation process of ABIFTS repeatedly for further improvement in makespan and/or resource usage. This schedule manipulation involves a branch-and-bound-style technique and two types of mutation (point and swap). Each job in the initial random schedule is tried on each available host to check whether any of these new matches shortens the current makespan. If one or more matches better than the original match are identified, the host on which the makespan is reduced the most is selected.

```
Algorithm ABIFTS
Input: : A workflow application G(V,E), a set H of h hosts in grid, #iterations g
Output: A fault tolerant schedule of G onto H
1. Compute b-level of each job $n_i \in V$
2. Sort $V$ in decreasing order of b-level value
3. Generate a random schedule $S$
4. while $g^{th}$ iteration is not reached do
5. Place check points at optimal intervals
6. for each $n_i \in V$ do
7. Try $n_i$ on each $h_j \in H$
8. Select the best host based on make span
9. end for
10. if no improvements on the schedule then
11. Mutate the schedule with 0.5 probability
12. Continue iteration with the mutated one
13. end if
14. Reschedule from check points in case of failure
15. end while
16. Compute the ALFT of each $n_i \in V$
17. while $n_i$ is not dispatched do
18. Dispatch all ready jobs
19. Remove the dispatched jobs from $V$
20. Wait until any job n to finish
21. if AFT(n) > ALFT(n) then
22. goto step 4
23. end if
24. end while
```

Fig. 3. ABIFTS Algorithm

At the end of each iteration, mutation is considered if no improvement is made during the current iteration. ABIFTS randomly chooses a mutation method between point and swap mutations and mutates each job in the schedule with a probability of 0.5—sufficient to generate substantially different schedules. Mutation occurs only with unscheduled jobs when rescheduling. The mutated schedule is then used as the current schedule and becomes the strict best schedule in the next iteration. If there have been some improvements on the schedule in the current iteration, iterate for further improvements. This is because changes made to jobs with low b-level values (low-priority jobs) may enable better job-host matches, in the next iteration, for unchanged high-priority jobs leading to improvement in the quality of the current schedule as well as the best schedules. This schedule manipulation process repeats for a predefined number of iterations.

The status of the process is recorded at regular intervals using check points and can be used in case of failure. If failure occurs, the scheduling process can recover from the nearest check point using the rescheduling strategy. If no failure occurs, the schedule proceeds in the normal fashion.

Now, jobs in the best schedule are dispatched to their assigned hosts as they become ready, i.e., their predecessor jobs have finished. During this actual job dispatch process, there might be cases in which some jobs are delayed—for unacceptably long—to complete their execution. These will trigger rescheduling events.

The actual latest start and finish times of a job n_i on a host h_j are defined as

$$\text{ALST}(n_i, h_j) = \text{ALFT}(n_i, h_j) - W_{i+j} \quad \text{... (6)}$$

ALFT $(n_i, h_j) =$

$$\begin{cases} \text{AFT}(n_i) & \text{if } n_i = n_{exit} \\ \text{Min}\left\{ \min_{n_k \in \text{succ}(n_i)} \text{ALST}(n_k, h_m) - c_{i+k}),\ \text{ALST}(n_{next}, h_j) \right\} & \text{otherwise} \end{cases}$$

... (7)

where ALST(n_{nex} ,h_j) is the actual latest start time of the next job scheduled after ni on the same host hj. The ALFT of a job is an indicator of whether the delay in the completion of the job is acceptable. In other words, the late completion of a job does not affect the makespan of a given workflow application as long as the time of the completion is no later than the actual latest finish time of the job; hence, the delay is acceptable.

VI. SIMULATION RESULTS

In my work, I have considered a set of 26 tasks scheduled in three hosts with varying processing power. Simgrid simulator is used for the simulation of the scheduling process. The task graph is given below:

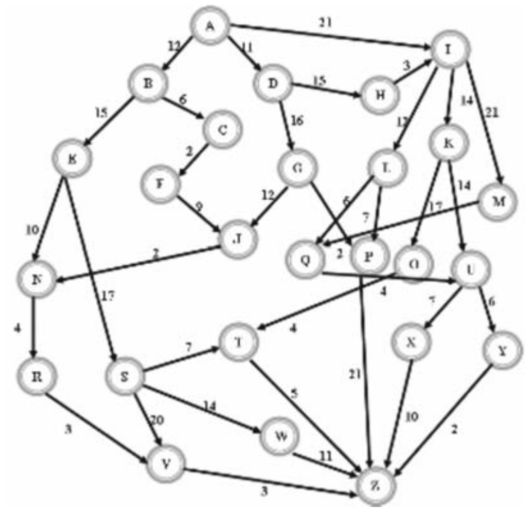


Fig. 4. Sample task graph

Table 1. Computation time of each task

A – 16	B – 20	C – 18	D – 15
E – 14	F – 26	G – 42	H – 22
I – 12	J – 40	K – 12	L – 12
M – 14	N – 19	O – 17	P – 23
Q – 12	R – 25	S – 6	T – 4
U – 10	V – 20	W – 18	X – 16
Y – 14	Z – 12		

The tasks are represented as circles. The interdependencies between jobs are represented by the edges. For example, in the above graph, I is dependent on A. The numbers in edges denote the communication time between the jobs. The computation time of each tasks is tabulated separately.

Chart 1. Graphical representation of scheduling outputs

The results are analyzed and represented as time-line chart as given below. Here time and workstations are taken in x and y axis respectively.

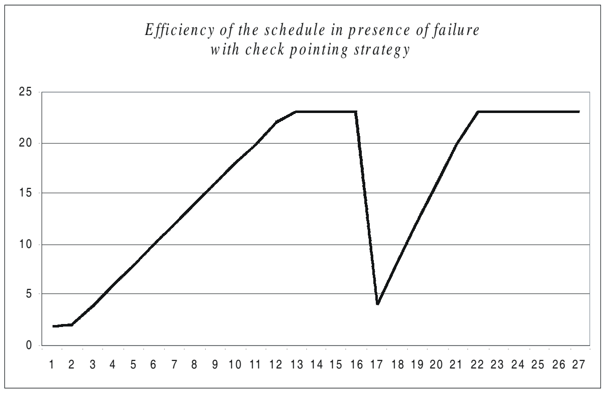
VII. RESULT ANALYSIS

The same algorithm is applied to various task graphs and corresponding optimal schedules are obtained. The collected results are tabulated as follows:

Table 2. Comparative results for various schedules

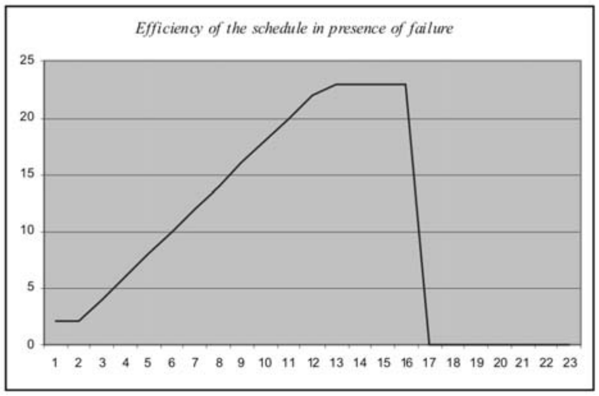
Number of Tasks taken	Number of resources in the grid	Schedule length	+	Efficiency improvem ent in %
		Random Schedule	ABIFTS schedule	
26	3	27.725	23.25	45.6%
19	4	18.32	11.892	39.3%
10	3	17.625	12.124	40.7%
35	4	29.567	24.127	44.9%

The graphs representing the effect of using fault tolerant mechanism (check point) is charted below:



Graph 1. Efficiency of the schedule in presence of failure

From graph1 we infer that the efficiency of the entire scheduling process degrades completely in case of failure of at least one resource.



Graph 2: Efficiency of the schedule in presence of failure by using check points

From graph2 we infer that the efficiency of the process decreases to some extent in case of resource failure. However, it improves gradually since rescheduling occurs from the check point. The gap between two peaks is referred as the rescheduling time.

VIII. CONCLUSION

From the above results, we can conclude that the ABIFTS algorithm can be used to provide an optimized schedule both by make span (completion time) and resource usage. And also, it can provide high efficiency in case of resource failures. Hence, we are able to get an optimized fault tolerable schedule which yields greater efficiency compared to other existing grid scheduling approaches.

REFERENCES

[1} Lee Y.C., Subrata, R., Zomaya, A.Y., Sep. 2009"On the Performance of a Dual-Objective Optimization Model for Workflow Applications on Grid Platforms," IEEE Trans. Parallel and Distributed Systems, vol. 20, no. 9, pp. 1273 - 1284.

[2] Topcuoglu H., Hariri S., and M. Wu, Mar. 2002. "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," IEEE Trans. Parallel and Distributed Systems, vol. 13, no. 3, pp. 260-274.

[3] Bozdag D., Catalyurek U., and Ozguner F., Apr. 2005 "A Task Duplication Based Bottom-Up Scheduling Algorithm for Heterogeneous Environments," Proc. 19th Int'l Parallel and Distributed Processing Symp. (IPDPS '05).

[4] Legrand A., Marchal L., and Casanova H., 2003 "Scheduling Distributed Applications: The SimGrid Simulation Framework,"Proc. Third IEEE/ACM Int'l Symp. Cluster Computing and the Grid (CCGrid '03), pp. 138-145.

[5] Sanguthevar Rajasekaran and John Reif "Handbook of Parallel Computing Models, Algorithms and Applications"

[6] Baghavathi Priya S., Prakash M, Dhawan Dr.K.K, 2007 "Fault Tolerance-Genetic Algorithm for Grid Task Scheduling Using Check point" IEEE Proc. 6th International conference on Grid and Cooperative Computing.

[7] http://simgrid.gforge.inria.fr

Sridevi.S received her B.E. degree in Information Technology from Madurai Kamaraj University, Madurai, Tamil Nadu, India in 2004. She has worked as a lecturer in the Department of Information Technology with Sri Kaliswari College, Sivakasi, India during the year 2005 – 2006. Now she is pursuing her M.E. degree in Computer Science and Engineering with Mepco Schlenk Engineering College, Sivakasi, Tamil Nadu, India. She has attended 2 National Conferences and 1 International Conference. Her field of research interest includes Grid computing and scheduling.