A SURVEY ON THE PERFORMANCE OF JOB SCHEDULING IN WORKFLOW APPLICATION ON GRID PLATFORMS

D.Daniel  
PG Scholar  
Department of Computer Science and Engineering  
Karunya University, Coimbatore  
E-Mail: daniel_joen@yahoo.com

Mrs. Margret Johnson  
Asst.Professor  
Department of Computer Science and Engineering  
Karunya University, Coimbatore  
E-Mail: margret_cse@karunya.edu

D.Asir  
PG Scholar  
Department of Computer Science and Engineering  
Karunya University, Coimbatore  
E-Mail: asir.info87@gmail.com

A.Catherine Esther Karunya  
PG Scholar  
Department of Computer Science and Engineering  
Karunya University, Coimbatore  
E-Mail: catherineesther@karunya.du.in

ABSTRACT

Performance is the main concern of scheduling the performance driven grid applications. The job scheduling algorithms provides the best throughput in completion of the task or it mainly concentrates on the resource utilization. The performance is calculated based on any one of the above mentioned objective, both for static and dynamic scheduling. The Adaptive Dual Objective Optimization scheduling (ADOS) algorithms takes two objectives, resource usage and completion time of a task for measuring the performance of the schedule in workflow applications. The study shows
ADOS algorithm delivers promising performance in three aspects: completion time, resource usage, and robustness to resource-performance fluctuations.

**Index Terms** Tasks, Makespan, Resource utilization, Resource usage.

I. INTRODUCTION

To produce an effective scheduling by accounting completion time and resource usage for workflow applications on grid platform. The proposed algorithm performs the scheduling by accounting for both completion time and resource usage—dual objectives. This algorithm incorporates rescheduling to deal with unforeseen performance fluctuations effectively. The proposed algorithm delivers performance and robustness to resource performance fluctuations.[1]

To reduce the makespan, the resource are made redundant, hence the performance of the resource management decreases. Since the performance of the grid resource changes dynamically, its accurate estimation of their performance is very difficult.

The majority of the grid applications fall into the interdependent task model. These classes of applications are generally known as workflow applications. Each composed of a number of interrelated jobs. Like traditional parallel programs, a workflow application can be represented by a directed acyclic graph (DAG), where the nodes represent individual jobs, and edges represent dependencies between these jobs.

Workflow applications can take advantage of a grid computing platform; however, inter job dependencies in such applications, besides resource heterogeneity and Dynamism, impose a great burden on scheduling. In some systems, this workflow scheduling is left for manual dispatch by users, while other systems employ automated workflow management platforms (WMPs) such as, Grid Flow [2]. These WMPs tend to focus on the minimization of the application’s completion time. However, there are other important performance considerations of WMPs, such as resource usage, load balancing, and fault tolerance. Although some WMPs have facilities to deal with these considerations, they often lack the capability of explicit resource usage control. Rather, for the sake of fault tolerance, resources are overly used (task duplication).
II. AN ADAPTIVE RESCHEDULING STRATEGY FOR GRID WORKFLOW APPLICATIONS.

Adaptive Rescheduling approach that can both continuously improve performance by considering the new resources and minimize the impact caused by unexpected resource downgrade or unavailability.

System Architecture: This system design which adapts the Planner to dynamic grid environment via collaboration with the Executor. The system consists of two main components Planner and Executor. The GRID Services on top of which the Executor is built, is a collection of essential Services for any grid system. The capabilities of each system components are defined as below:

Planner: The Planner has a collective set of subcomponents, including Scheduler, Performance History Repository and Predictor. For each workflow application represented as a DAG, the Planner instantiates a Scheduler instance. Based on the performance history and resource availability, the Scheduler inquires the Predictor to estimate the communication and computation cost with the given resource set. It then decides on resource mapping, with the goal of achieving optimal performance for entire workflow, and submits the schedule to the Executor. During the execution, the Scheduler instance listens to the pre-defined events of interest, for example, addition of new resources and significant variance of job performance evaluates the event and reschedules the application if necessary. In the mean time, the Scheduler updates the Performance History Repository with the latest job performance information to improve the estimation accuracy subsequently. As each instance of Scheduler manages an individual DAG, it will focus on the events specifically associated with the DAG it manages to continuously improve the makespan. By collectively working with other instances of Scheduler, this multiple-instance scheme can further improve the overall system performance [7].

Executor: The Executor is an enactment environment for workflow applications, can be further decomposed into Execution Manager, Resource Manager and Performance Monitor according to their respective roles in the run time environment. The Executor supports advance reservation of resources. The Execution Manager receives the DAG and executes it as scheduled. It is also responsible for getting job input file ready
and executing the job on mapped resource. Upon arrival of a schedule, the Resource Manager will reserve the resource as per the schedule. If the arriving schedule is a result of rescheduling, it revokes resource reservation for replaced schedule before making new reservations. As part of the collaboration, the Resource Manager and Performance Monitor update the Planner with information and event subscribed by the Planner.

**A. HEFT – Based Adaptive Rescheduling: AHEFT**

Adaptive scheduling strategy, which is an HEFT-based adaptive rescheduling algorithm, referred to as AHEFT hereafter. Specifically, we use HEFT heuristic to implement the schedule \((S_0, P, H)\) method in the generic algorithm. For consistence purposes, A workflow application is represented by a direct acyclic graph, \(G = (V, E)\), where \(V\) is the set of \(v\) jobs (nodes) and \(E\) is the set of edges between jobs. Each edge \((i, j) \in E\) represents the precedence constraint such that job \(n_i\) should complete its execution before job \(n_j\) starts. Data is a \(v \times v\) matrix of communication data, where \(data_{i,k}\) is the amount of data required to be transmitted from job \(n_i\) to job \(n_k\). \(R\) is a set of resources which represent computation units.

![Figure 2.1 The diagram of the system design.](image)
The variable clock is used as logical clock to measure the time span of DAG execution, it is initially set as 0 when the DAG starts to execute. When the DAG finishes successfully, the clock reads as the Makespan of the DAG. A job cannot start without all required inputs available on the resource on which the job is to execute. Such inputs are in turn the outputs from immediate predecessor jobs. The upward rank is calculated for each job and sorted in non-increasing order, which corresponds to significance order how the individual job affects the final Makespan [7][9]. The basic concept of this algorithm is to select the “best” resource which minimizes the earliest finish time of the job currently with highest upward rank and remove the job from unscheduled job list once it is assigned with resource. The resource selection process repeats until the list is empty.

III. GRIDFLOW: WORKFLOW MANAGEMENT FOR GRID COMPUTING.

Grid computing is becoming a mainstream technology for large-scale distributed resource sharing and system integration. Workflow management is emerging as one of the most important grid services. A workflow management system for grid computing called Grid Flow. The main functionalities of grid workflow management include workflow construction, simulation, scheduling, execution, monitoring, conflict solving, and so on. A grid environment presents a number of new challenges for the workflow management system:

**CROSS – DOMAIN:**

The process of a grid workflow encompasses multiple administrative domains (organizations). The lack of central ownership and control results in incomplete information and many other uncertain factors.

**Dynamism:** Since grid resources are not entirely dedicated to the environment, computational and networking capabilities can vary significantly over time. Application performance prediction becomes difficult and real-time resource information update within a large-scale global grid becomes impossible [2].

**A. GRID WORKFLOW MANAGEMENT.**

This work focuses more on service-level support such as grid workflow management and scheduling, a Grid Flow user portal is also developed that provides a
graphical user interface (GUI) to facilitate the composition of grid workflow elements and the access to additional grid services. The system is designed so that workflow management follows the same layered framework as that of resource management, including global grid workflow management and local grid sub workflow scheduling. The implementation of grid workflow management is carried out at multiple layers:

**Task:** Tasks are the smallest elements in a grid workflow. Task scheduling is implemented using Titan, and as stated, this work focuses more on the sub-workflow and workflow levels of management and scheduling.

**Sub-workflow:** A sub-workflow is a flow of closely related tasks that is to be executed in a predefined sequence on grid resources of a local grid (within one organization). Conflicts occur when tasks from different sub-workflows require the same resource simultaneously.

**Workflow:** A grid application can be represented as a flow of several different activities, each activity represented by a sub-workflow. These activities are loosely coupled and may require multi-sited grid resources. Simulation, execution and monitoring services can be provided [2].

1) **GRIDFLOW USER PORTAL**

The Grid Flow portal is an integrated environment that enables users to construct a grid workflow and access grid services. To construct a grid workflow, a user needs to define properties of each sub-workflow and task and their execution sequences. In general, a sub-workflow or a task can have several pre- and post- activities the portal also provides direct user interfaces to the information and performance services. However, if the user does not know anything about the available grid services and resources, he can submit the workflow to the global workflow management system, which will provide the services automatically.

2) **GLOBAL GRID WORKFLOW MANAGEMENT**

The global grid workflow management system receives requests from the Grid Flow portal with XML specifications of grid workflows, and provides three main functionalities:
Simulation: Simulation takes place before a grid workflow is actually executed, during which time a workflow schedule is achieved. The simulation results can be returned to the Grid Flow portal for user agreement or passed directly to the execution engine.

Execution: A grid workflow is executed according to the simulated schedule. Due to the dynamic nature of the grid environment, the schedule may not be executed accordingly. When large delays of some sub workflows occur, the rest or whole of the workflow may be sent back to the simulation engine and rescheduled.

Monitoring: Global grid workflow management also provides interfaces that provide access to real-time status reports of task or sub-workflow execution. GGWM algorithm is used [11].

3) LOCAL GRID SUB-WORKFLOW SCHEDULING

Scheduling a flow of tasks onto grid resources within a local grid is very similar to the process that schedules a workflow onto different local grids introduced above. One Important difference is that the local grid sub workflow scheduling has to deal with multiple tasks that may belong to different sub-workflows. The execution time has to be estimated with the extra consideration of conflicts, which may occur when multiple tasks require the same grid resource at the same time. LGSS algorithm is used.

IV. PERFORMANCE-EFFECTIVE AND LOW-COMPLEXITY TASK SCHEDULING FOR HETEROGENEOUS COMPUTING.

The application scheduling problem has been studied and various algorithms have been proposed mainly for the homogeneous processors. There are only few algorithms for heterogeneous processors, but they are high scheduling cost and they do not deliver quality schedule with lower cost. Two new algorithms have been proposed for heterogeneous processors which provide fast scheduling time and high performance, HEFT and CPOP algorithm.

A. HETEROGENEOUS – EARLIEST – FIRST – TIME (HEFT) ALGORITHM

The HEFT algorithm is an application scheduling algorithm for bounded number of heterogeneous processors, which has two major phases a task prioritizing phase for
computing the priorities of all tasks and a processor selection phase for selecting the tasks in order of their priorities and scheduling each selected task on its “best” processor, which minimizes the task’s finish time.

**Task prioritizing phase:** This phase requires the priority of each task to be set with upward rank value, which is based on the mean computing and mean communication cost. The task list is generated by sorting the task by decreasing order of rank value. Tie breaking is done randomly. There can be alternative policies for tie breaking, such as selecting the task whose immediate successor tasks has higher upward ranks. Since these alternate policies increase the time complexity, random selection strategy is preferred.

**Processor selection phase:** For most of the task scheduling algorithm, the earliest available time of a processor pj for a task execution is the time when pj completes the execution of its last assigned task. However, the HEFT algorithm has an insertion-based policy which considers the possible insertion of a task in an earliest idle time slot between two already – scheduled tasks on a processor. The length of an idle timeslot, i.e. the difference between execution start time and finish time of two tasks that were consecutively scheduled on the same processor, should be at least capable of computation cost of the task to be scheduled.

**B. CRITICAL PATH-ON-A PROCESSOR (CPOP) ALGORITHM**

CPOP algorithm has two phase, task prioritizing and processor selection phase. It uses a different attribute for setting the task priorities and a different strategy for determining the best processor for the selected task.

**Task prioritizing phase:** Upward rank and downward rank for all tasks are computed using mean computation and mean communication cost. CPOP uses critical path for the given application graph. The length of the critical path is the sum of computation cost of tasks on the path and intertask communication cost along the path, critical path length is equal to the entry task’s priority. Initially the entry task is selected and marked as critical path task; an immediate successor of the selected task that has the highest priority value is selected and marked as critical path task. This process is repeated
until the exit node is reached. For tie break, successor which has highest priority is selected.

**Processor selection phase:** Critical path processor is the one that minimize the computation cost of the tasks on critical path. If the selected task is on critical path, then it is scheduled on the critical path processor. Otherwise, assigned to processor which minimize the earliest execution finish time of task. In both cases insertion policy is consider.

V. A TASK DUPLICATION BASED BOTTOM-UP ALGORITHM FOR HETEROGENEOUS ENVIRONMENTS

DBUS algorithm traverses the DAG in a bottom-up fashion while taking advantage of task duplication and task insertion. The makespans generated by the DBUS algorithm are much better than those generated by the existing algorithms, HEFT, HCPFD and HCNF.

A. DBUS: Duplication-based Bottom-Up Scheduling Algorithm

The DBUS algorithm consists of a critical path-based listing phase followed by duplication based scheduling phase. In the traditional approach of top-down DAG traversal, duplication is carried out when a task's finish time can be improved with duplication of its critical parent on the same processor with the task itself. This means that a task with multiple children can be duplicated on multiple processors if its duplication helps improving its children's finish time on those processors [6]. The drawback in this approach is that, the duplication of the parent task on each of such processors is done independently. Therefore the positions of the parent task on relevant processors are decided without any optimization with respect to each other. By traversing the DAG in a bottom up fashion, the proposed algorithm first schedules all children of a task before scheduling the task on as many processors as necessary. Consequently, it is more likely to make good duplication decisions since all copies of the parent task are considered at the same time.

In addition, the stop criterion for duplication is not determined by the number of duplications already carried out, but by the quality of the current schedule. Since the number of beneficial duplications may differ significantly across different problems,
DBUS offers superior performance compared to algorithms that limit duplication by the number of duplicates. Furthermore, the DBUS algorithm is an insertion based algorithm that allows tasks to be scheduled at the first available time slot that can accommodate themselves. Task insertion based algorithms have better chances of finding shorter schedule lengths compared to non-insertion based ones. There are two phases: listing phase and scheduling phase [5].

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Task insertion</th>
<th>Restrictions on duplication</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEFT</td>
<td>Yes</td>
<td>No duplication</td>
</tr>
<tr>
<td>HCPF D</td>
<td>No</td>
<td>Only critical parent of a task</td>
</tr>
<tr>
<td>HCNF</td>
<td>No</td>
<td>Only critical parent of a task</td>
</tr>
<tr>
<td>DBUS</td>
<td>Yes</td>
<td>No restriction</td>
</tr>
</tbody>
</table>

Table 5.1 Comparison of Scheduling Algorithms

VI. ON THE PERFORMANCE OF A DUAL OBJECTIVE OPTIMIZATION MODEL FOR WORKFLOW APPLICATIONS ON GRID PLATFORMS

The primary objective of scheduling the workflow application is to minimize the completion time of the application. However, they tend to achieve this objective at the expense of redundant resource usage. The proposed algorithm performs the scheduling by accounting for both completion time and resource usage - dual objectives. Since the performance of grid resources changes dynamically and the accurate estimation of their performance is very difficult, this algorithm incorporates rescheduling to deal with unforeseen performance fluctuations effectively.

WORKFLOW APPLICATION

The majority of applications in the grid environment fall into interdependent task model known as Workflow applications. Each workflow application is composed of number of interrelated jobs. Like traditional parallel programs, a workflow application can be represented by a directed acyclic graph (DAG), where the nodes represent individual jobs, and edges represent dependencies between these jobs.

Workflow applications can take advantage of a grid computing platform; however, interjob dependencies in such applications, besides resource heterogeneity and dynamism, impose a great burden on scheduling. In some systems, this workflow scheduling is left for manual dispatch by users, while other systems employ automated
workflow management platforms (WMPs) [1]. These WMPs tend to focus on the minimization of the application’s completion time. However, there are other important performance considerations of WMPs, such as resource usage, load balancing, and fault tolerance. Although some WMPs have facilities to deal with these considerations, they often lack the capability of explicit resource usage control. Rather, for the sake of fault tolerance, resources are overly used (task duplication)[8].

**ADAPTIVE DUAL OBJECTIVE SCHEDULING (ADOS) ALGORITHM**

This paper address the problems of scheduling workflow applications in grid and propose an Adaptive Dual Objective Scheduling (ADOS) Algorithm as a novel semi dynamic scheduling heuristic, which statically generates the initial schedule using an evolutionary technique and adapts it dynamically as the performance of resources changes; hence, the algorithm is semi dynamic. The two main strengths of ADOS are 1) the good quality of the output schedules with small resource usage and 2) the adaptability to performance fluctuations on resources. The first strength is achieved by iteratively improving an initial random schedule using a branch-and-bound technique and genetic operator (mutation) improvements are made not only in completion time but also in resource usage. Resource usage is equally seriously taken into account compared to completion time, because the overuse of grid resources shared by many users, so at any given time, there might be a number of jobs running on those resources degrades the overall performance of the grid. The second strength is made possible by incorporating a systematic rescheduling strategy. Specifically, ADOS 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 yet running [1][10].

While the task scheduling problem in heterogeneous computing systems with perfectly accurate performance information on resources and applications still remains very difficult (NP-hard), uncertainties on resource performance and the lack of control over grid resources make workflow scheduling even more challenging. This algorithm for
workflow scheduling with the detailed description of our specific approaches to those issues in such a scheduling problem. 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 ADOS.

A. WORKING OF ADOS

To start with, ADOS generates a random schedule, this initial solution undergoes the manipulation process of ADOS repeatedly for further improvement in makespan and 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. ADOS keep track of the two best schedules (relaxed and strict) possibly the same schedule based on makespan and on makespan and resource usage, respectively. At first glance, the reason for maintaining the best schedule based only on makespan, which is not clear, ADOS determines the quality of schedule based on both makespan and resource usage. We have observed that the application of such strict criteria throughout the entire scheduling process tends to produce very good schedules in terms of resource usage; however, this strict schedule selection scheme raises the issue that schedules are less attractive, as far as makespan is concerned. This observation has led to the incorporation of a loosened schedule selection scheme as an effective solution to the issue. This scheme helps shorten the makespan noticeably, as the result of a compromise between makespan and resource usage. The scheme implicitly takes into account resource usage. Specifically, the resource
usage of the best schedule is guarded to remain at a reasonable level by , which can be seen as the quality guarantor of ADOS for resource usage. Note that $S^*$ is a slight modification of $S'$. Thus, the degree of sacrifice made to the resource usage of $S^*$ is minimal.

At the end of each iteration, mutation is considered if no improvement is made during the current iteration. ADOS randomly choose a mutation method between point and swap mutations and mutates each job to generate substantially different schedules. Note that mutation only occurs with unscheduled jobs when rescheduling. The mutated schedule is then used as the current schedule and the strict best schedule in the next iteration. If there have been some improvements in the current iteration, $S$ is passed onto the next iteration 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.

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 trigger rescheduling events. Here, it is very crucial how to define “unacceptably long.” For each job, ADOS computes the actual latest finish time, which is a modification of LFT. The modification is made because ADOS uses job insertion. 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 [1].
VII. EVALUATION AND COMPARISON

Adaptive scheduling strategy based HEFT-based adaptive rescheduling algorithm (AHEFT) has the advantage of continuously improves the performance, considering the new resource and minimizes the impact caused by unexpected resource down grade or unavailability [7]. Drawbacks of adaptive rescheduling technique are it takes time in rescheduling the jobs and to implement the collaboration model, rescheduling has to be integrated with advance resource reservation and resource availability prediction model.

The grid flow gives the Advantage of grid performance service comprises performance prediction capability with a new application response measurement technique, which can be used to enable prediction-based scheduling as well as response-based scheduling. But the disadvantage such as the process of a grid workflow encompasses multiple administrative domains (organisations)[7]. The lack of central ownership and control results in incomplete information and Computational and networking capabilities can vary significantly over time in the grid environment. Application performance prediction
becomes difficult and real-time resource information update within a large-scale global grid becomes impossible, which lower its performance.

Critical Path-on-a processor (CPOP) Algorithm and Heterogeneous Earliest First Time (HEFT) Algorithm, both gives more or less same performance measures as high performance and fast schedule time. But has slight disadvantage of high schedule cost. Duplication-based Bottom-Up Scheduling Algorithm (DBUS) gives uses both task insertion and task duplication. It also gives the facility to minimize the schedule length [6]. It does not impose any restriction on number of task duplication and Task duplication mainly causes an increase of resource usage which causes disadvantage to the algorithm. The ADOS algorithm has the highest possibility for reducing the makespan of the task, which is the total amount of time required from the start of the first job to the end of the last job. It also reduces the resource usage .the another advantage of this algorithm is semidynamic, which could adapt to the dynamic changes that happens in the grid environment.

VIII. CONCLUSION

In this paper we surveyed four scheduling algorithms. Adaptive scheduling strategy based HEFT-based adaptive rescheduling algorithm (AHEFT) which is follows dynamic scheduling. Critical Path-on-a processor (CPOP) Algorithm and Heterogeneous Earliest First Time (HEFT) Algorithm, Duplication-based Bottom-Up Scheduling Algorithm (DBUS) which follows static scheduling. ADOS algorithm follows Semidynamic scheduling strategy, which schedules the jobs statically and adapts its schedule as per the dynamic changes occur in the grid environment using rescheduling. This algorithm also provides dual object and improves the job scheduling.

REFERENCE


