A Dynamic Critical Path Algorithm for Scheduling Scientific Workflow Applications on Global Grids

Mustafizur Rahman, Srikumar Venugopal and Rajkumar Buyya
Grid Computing and Distributed Systems (GRIDS) Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
{mmrahman, srikumar, raj}@csse.unimelb.edu.au

Abstract

Effective scheduling is a key concern for the execution of performance driven Grid applications. In this paper, we propose a Dynamic Critical Path (DCP) based workflow scheduling algorithm that determines efficient mapping of tasks by calculating the critical path in the workflow task graph at every step. It assigns priority to a task in the critical path which is estimated to complete earlier. Using simulation, we have compared the performance of our proposed approach with other existing heuristic and meta-heuristic based scheduling strategies for different type and size of workflows. Our results demonstrate that DCP based approach can generate better schedule for most of the type of workflows irrespective of their size particularly when resource availability changes frequently.

1. Introduction

Many of the large-scale scientific applications executed on present-day Grids are expressed as complex e-Science workflows [1][2]. A workflow is a set of ordered tasks that are linked by data dependencies [3]. A Workflow Management System (WMS) [3] is generally employed to define, manage and execute these workflow applications on Grid resources. A WMS may use a specific scheduling strategy for mapping the tasks in a workflow to suitable Grid resources in order to satisfy user requirements. Numerous workflow scheduling strategies have been proposed in literature for different objective functions [4]. However, the majority of these are static scheduling algorithms that produce a good schedule given the current state of Grid resources and do not take into account changes in resource availability.

Critical path heuristics [5] have been used extensively for scheduling interdependent tasks in multiprocessor systems. These aim to determine the longest of all execution paths from the beginning to the end (or the *critical path*) in a task graph, and schedule them earliest so as to minimize the execution time for the entire graph. Kwok and Ahmad [6] introduced the Dynamic Critical Path (DCP) algorithm in which the critical path is dynamically determined after each task is scheduled. However, this algorithm is designed for mapping tasks on to homogeneous processors, and is static, in the sense that the schedule is only computed once for a task graph. In this paper, we extend the DCP algorithm to map and schedule tasks in a workflow on to heterogeneous resources in a dynamic Grid environment. We have extensively compared the performance of our algorithm, called DCP-G (Dynamic Critical Path for Grids), against well-known Grid workflow algorithms.

The rest of the paper is organized as follows. In the next section, we describe existing heuristics and meta-heuristics based workflow scheduling techniques on distributed systems such as Grid. The proposed DCP-G workflow scheduling algorithm is presented in Section 3. Experiment details and simulation results are presented in Section 4. Finally, we conclude the paper with the direction for future work in Section 5.

2. Related Work

Generally, a workflow application is represented as a Directed Acyclic Graph (DAG) in which graph nodes represent tasks and graph edges represent data dependencies among the tasks with weights on the nodes representing computation and weights on the edges representing communication volume. Therefore, workflow scheduling problem is usually considered as a special case of the DAG scheduling problem. As the DAG scheduling problem is NP-complete, we rely on heuristics and meta-heuristics based scheduling strate-

gies to achieve the most efficient possible solution. In the following, we present some of the well-known heuristics and meta-heuristics for workflow scheduling on Grid systems.

Myopic [7]: schedules an unmapped ready task, in arbitrary order, to a resource which is expected to complete that task earliest, until all tasks have been scheduled. It is considered as the simplest method for scheduling workflow applications.

Min-Min [8]: is a list scheduling heuristic that assigns priority to the task based on its Expected Completion Time (ECT) on a resource. In every step of iteration, it discovers the task that has Minimum Expected Completion Time (MCT) among all the available tasks and assigns it to the resource that provides the MCT. This is repeated until all tasks are assigned. The intuition behind Min-Min is to consider all unmapped independent tasks during each mapping decision, whereas Myopic only considers one task at a time.

Max-Min [8]: is similar to Min-Min except that in each iterative step, a task having the maximum ECT is chosen to be scheduled on the resource which is expected to complete the task at the earliest time. Intuitively, Max-Min attempts to minimize the total workflow execution time by assigning longer tasks to comparatively better resources. Both Min-Min and Max-Min have been used for scheduling workflow tasks in Pegasus [9].

Heterogeneous Earliest Finish Time (HEFT) [10]: used in the ASKALON workflow manager [7][11], is a well-established list scheduling algorithm which assigns higher priority to the workflow task having higher rank value. It calculates rank value based on the average execution time for each task and average communication time between resources of two successive tasks, where the tasks in the 'critical path' get comparatively higher rank values. In the resource selection phase, tasks are scheduled in the order of their priorities and each task is assigned to the resource that can complete the task at the earliest time. The advantage of using this technique over Min-Min or Max-Min is that while assigning priorities to the tasks it considers the entire workflow rather than focusing on only unmapped independent tasks at each step.

Greedy Randomized Adaptive Search Procedure (GRASP) [12]: is an iterative randomized search technique. In GRASP, a number of iterations are conducted to search a possible optimal solution for mapping tasks on resources. A solution is generated at each iterative step and the best solution is kept as the final schedule. This searching procedure terminates when the specified termination criterion, such as the completion of a certain number of iterations, is satisfied. GRASP can generate better schedules than the other scheduling techniques stated previously as it searches the whole solution space considering entire workflow and available resources.

Genetic Algorithm [13]: is also meta-heuristic based scheduling technique such as GRASP. It allows a high quality solution to be derived from a large search space in polynomial time by applying the principles of evolution. Instead of creating a new solution by randomized search as in GRASP, GA generates new solutions at each step by randomly modifying the good solutions generated in previous steps which results a better schedule within less time.

3. The Proposed DCP-G Algorithm

For a task graph, the lower and upper bounds of starting time for a task are denoted as the Absolute Earliest Start Time (AEST) and the Absolute Latest Start Time (ALST) respectively. In the DCP algorithm [6], the tasks on the critical path have equal AEST and ALST values as delaying these tasks affects the overall execution time for the task graph. The first task on the critical path is mapped to the processor identified for it. This process is repeated until all the tasks in the graph are mapped.

However, this algorithm is designed for scheduling all the tasks in a task graph with arbitrary computation and communication times to a multiprocessor system with unlimited number of fully connected identical processors. But, Grids [14] are heterogeneous and dynamic environments consisting of computing, storage and network resources with different capabilities and availability. Therefore, to work on Grids, the DCP algorithm needs to be extended in the following manner:

- For a task, the initial AEST and ALST values are calculated for the resource which provides the minimum execution time for the task. The overall objective is to reduce the length of the critical path at every pass. We follow the intuition of the Min-Min heuristic in which a task is assigned to the resource that executes it fastest.
- For mapping a task on the critical path, all available resources are considered by DCP-G, as opposed to the DCP algorithm, which considers only the resources (processors) occupied by the parent and child tasks. This is because, in the latter case, the execution time is not varied for different processors, and only the communication time between the tasks could be reduced by assigning tasks to the same resource. However, in Grids, the communication and computation times are both liable to change because of resource heterogeneity.

- When a task is mapped to a resource, its execution time and data transfer time from the parent node are updated accordingly. This changes the AEST and ALST of succeeding tasks.

3.1. Calculation of AEST and ALST in DCP-G

In DCP-G, the start time of a task is not finalized until it is mapped to a resource. Here, we also introduce two more attributes: the Absolute Execution Time (AET) of a task which is the minimum execution time of the task, and Absolute Data Transfer Time (ADTT) which is the minimum time required to transfer the output of the task given its current placement. Initially, AET and ADTT are calculated as,

$$AET(t) = \frac{Task\_size(t)}{\max_{k \in ResourceList} \{PC(R_k)\}}$$

$$ADTT(t) = \frac{Task\_output\_size(t)}{\max_{k \in ResourceList} \{BW(R_k)\}}$$

Where, $PC(R_k)$ and $BW(R_k)$ are processing capability and transfer capacity i.e. Bandwidth of resource R_k respectively.

Whenever a task t is scheduled to a resource, the values of $AET(t)$ and $ADTT(t)$ are updated accordingly. Therefore, the AEST of a task t on resource R, denoted by $AEST(t,R)$, is recursively defined as,

$$AEST(t,R) = \max_{1 \le k \le p} \{AEST(t_k, R_{t_k}) + AET(t_k) + C_{t,t_k}(R_t, R_{t_k})\}$$

where, t has p parent tasks, t_k is the k^{th} parent task and,
$AEST(t,R)=0$; if t is an entry task.
$C_{t,t_k}(R_t, R_{t_k})=0$; if $R_t = R_{t_k}$
$C_{t,t_k}(R_t, R_{t_k}) = ADTT(t_k)$; if t and t_k are not scheduled.

Here, the communication time between two tasks is considered to be zero if they are mapped to the same resource, and equal to the ADTT of parent task, if the child is not mapped yet. Using this definition, the AEST values can be computed by traversing the task graph in a breadth-first manner beginning from the entry tasks.

Once AESTs of all the tasks are computed, it is possible to calculate Dynamic Critical Path Length (DCPL) which is the schedule length of the partially mapped workflow. DCPL can be defined as,

$$DCPL = \max_{1 \le i \le n} \{AEST(t_i, R_{t_i}) + AET(t_i)\}$$

where, n is the total number of tasks in the workflow.

After computing the DCPL, the values of ALST can be calculated by traversing the task graph in a breadth-first manner but in the reverse direction. Thus, the ALST of a task t in resource R, denoted as $ALST(t,R)$, can be recursively defined as,

$$ALST(t,R) = \min_{1 \le k \le c} \{ALST(t_k, R_{t_k}) - AET(t) - C_{t,t_k}(R_t, R_{t_k})\}$$

where, t has c child tasks, t_k is the k^{th} child task and,
$ALST(t,R)=DCPL - AET(t)$; if t is an exit task.
$C_{t,t_k}(R_t, R_{t_k})=0$; if $R_t = R_{t_k}$
$C_{t,t_k}(R_t, R_{t_k}) = ADTT(t_k)$; if t and t_k are not mapped.

As in DCP, a task in DCP-G is considered to be on the critical path if its AEST and ALST values are equal. In order to reduce the value of DCPL at every step, the task selected for scheduling is the one that is on the critical path and has no unmapped parent tasks.

3.2. Resource Selection

After identifying a critical task, we need to select an appropriate resource for that task. We select the resource that provides the minimum execution time for that task. This is discovered by checking all the available resources for one that minimizes the potential start time of the critical child task on the same resource, where the critical child task is the one with the least difference of AEST and ALST among all the child tasks of the critical task. Finally, the critical task is mapped to the resource that provides earliest combined start time.

3.3. DCP-G Example

Figure 1 illustrates the DCP-G algorithm with a step-by-step explanation of the mapping of tasks in a sample workflow. The sample workflow consists of five tasks denoted as T_0, T_1, T_2, T_3 and T_4 with different execution and data transfer requirements. The length and size of the output of each task shown in Figure 1(a) are measured in Million Instructions (MI) and GigaBytes (GB) respectively. The tasks are to be mapped to two Grid resources R_1 and R_2 with processing capability (PC) and transfer capacity i.e. Bandwidth (BW) as indicated at the bottom of Figure 1.

First, the AET and ADTT values for each task are calculated as shown in Figure 1(a). Then using these values, AEST and ALST of all the tasks are calculated according to Section 3.1 (Figure 1(b)). Since T_0, T_2, T_3 and T_4 have equal AEST and ALST, they are on critical path with T_0 as the highest task. Hence, T_0 is selected as the critical task and mapped to resource R_1 which gives T_0 the minimum combined start time. At the end of this step, the schedule length of the workflow, i.e. DCPL is 890. Similarly, in Figure 1(c), T_2 is selected as critical task and mapped to R_1. As both T_0 and T_1 are mapped to R_1 and the data transfer time of T_0 is now zero, the AEST and ALST of all the tasks are changed and the schedule length becomes 850 (Figure 1(d)). In the next step, T_3 is mapped to R_1 as well and

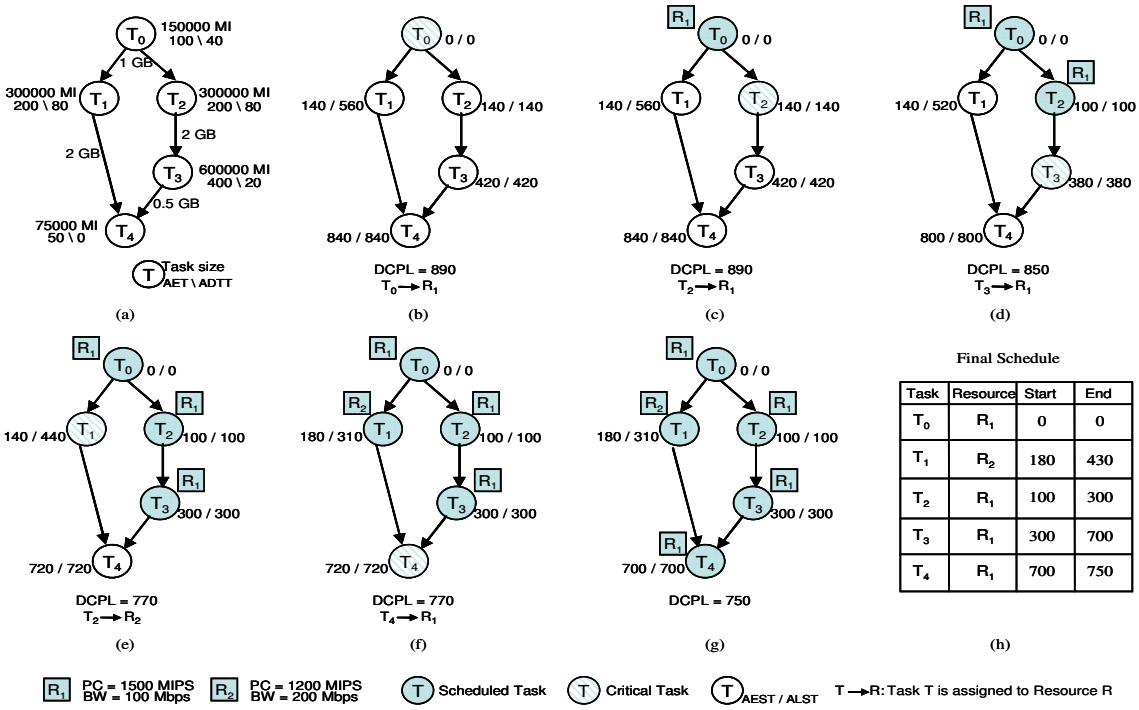


Figure 1. Example of workflow scheduling using DCP-G algorithm

the DCPL is reduced to 770 as the data transfer time for T_2 is zero.

Now T_4 is the only task remaining on the critical path (Figure 1(e)). However, one of its parent tasks, T_1, is not mapped yet and therefore T_1 is selected as critical task. As T_2 and T_3 are already mapped to R1, the start time of T_1 on R_1 is 700. Therefore, T_1 is mapped to R_2 as its start and end times on R_2 are 180 and 430 respectively. Finally, when T_4 is mapped to R_1 (Figure 1(g)), all the tasks have been mapped and the schedule length can not be improved any further and a schedule length of 750 is obtained. The final schedule generated by DCP-G is shown in a table in Figure 1(h).

4. Performance Evaluation

We evaluate DCP-G by comparing the schedules produced by it against those produced by the other algorithms described previously for a variety of workflows in a simulated Grid environment. In this section, first we describe our simulation methodology and setup, and then present the results of experiments.

4.1. Simulation methodology

We use GridSim [15] to simulate the application and Grid environment for our simulation. We model different entities in GridSim in the following manner.

Workflow model. We implement a workflow generator that can generate various formats of weighted pseudo-application workflows. The following input parameters are used to create a workflow.

• *N*, the total number of tasks in the workflow.

• α, the shape parameter represents the ratio of the total number of tasks to the width (i.e. maximum number of nodes in a level). So width W = $\left\lceil \frac{N}{\alpha} \right\rceil$

• Type of workflow: Our workflow generator can generate three types of workflow namely parallel workflow, fork-join workflow and random workflow.

Parallel workflow: In parallel workflow [16], a group of tasks creates a chain of tasks with one entry and one exit task and there can be several such chains in one workflow. Here, one task is dependent on only one task, but the tasks at the head of chains are dependant on entry task and the exit task is dependant on the tasks at the tail of chains. Number of levels in a parallel workflow can be specified as,

Number of levels = $\left\lfloor \frac{N-2}{W} \right\rfloor$

Fork-join workflow: In fork-join workflow [2], forks of tasks are created and then joined. So, there can be only one entry task and one exit task in this kind of workflow but the number of tasks in each level depends on total number of tasks and the width in that level, W. Number of levels in fork-join workflow can be specified as, Number of levels = $\left\lfloor \frac{N}{W+1} \right\rfloor$

Random workflow: In random workflow, dependency and number of parent tasks of a task which equals to the indegree of a node in DAG representation of the workflow, is generated randomly. Here, task dependency and the indegree are calculated as,

Maximum Indegree (T_i) = $\left\lfloor \frac{W}{2} \right\rfloor$

Minimum Indegree (T_i) = 1

Parent (T_i)= $\{T_x | T_x \in [T_0 \ldots T_{i-1}]\}$;if T_i is not a root task; where, x is a random number and 0 <= x <= $\left\lfloor \frac{W}{2} \right\rfloor$

Parent (T_i) = $\{\phi\}$; if T_i is a root task

In Figure 2, a sample of each type of workflow is illustrated where N=10, and α=5.

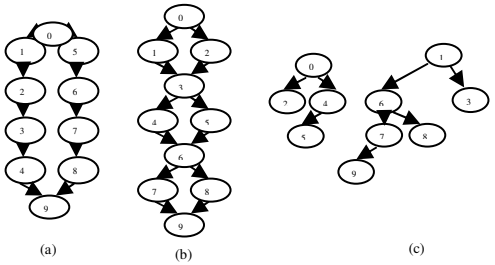


Figure 2. Three sample workflows : (a) Parallel workflow; (b) Fork-join workflow; (c) Random workflow

In simulation, we use MI (Million Instructions) to denote the length of tasks and MB (Mega Bytes) to denote the output data size of each task.

Resource model. As the execution environment for tasks in scientific workflows is heterogeneous, we use heterogeneous resources with different processing capabilities. Here, we choose 8 resources (refer to Table 1) from the European Data Grid (EDG) 1 test bed [17] used for simulation in [18]. The processing capability of the resources is measured in MIPS (Million Instructions per Second) and the bandwidth in Mbps (Megabits per second).

Table 1. Resources used for evaluation

Resource Name/Site (Location)	No. of Nodes	Single PE Rating (MIPS)	Mean Load
RAL (UK)	41	1140	0.9
NorduGrid (Norway)	17	1176	0.9
NIKHEF (Netherlands)	18	1166	0.9
Milano (Italy)	7	1000	0.5
Torino (Italy)	4	1330	0.5
Catania (Italy)	5	1200	0.6
Padova (Italy)	13	1000	0.4
Bologna (Italy)	20	1140	0.8

4.2. Simulation setup

The workflows for evaluation are generated using the following parameters:

Type = {parallel, fork-join, random}

N = {50, 100, 200, 300}

α = {10}

Here, the size of each task in the workflow is generated from a uniform distribution between 100,000 MI to 500000 MI while the output data size of each task is also generated from a uniform distribution between 1 GB and 5 GB.

For GRASP, we run 600 iterations to map tasks to the resources and then select the best schedule out of those are generated. For GA, the parameters for various genetic operators such as selection, crossover and mutation are set using those applied in previous studies [19]. Table 2 shows the values of different parameters used for simulating GA.

Table 2. Parameters of Genetic Algorithm

Parameter	Value/type
Population size	60
Crossover probability	0.7
Swapping mutation probability	0.5
Replacing mutation probability	0.8
Fitness function	Makespan of workflow
Selection scheme	Elitism-Roulette Wheel
Stopping condition	300 iterations
Initial individuals	Randomly generated

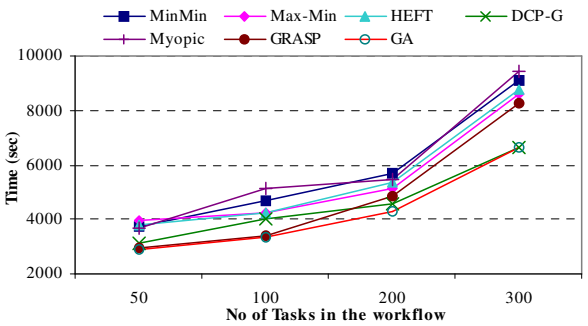
4.3. Results and Observations

We evaluate the scheduling heuristics on the basis of the total makespan produced and the time required for scheduling the workflows. Makespan is the total time required for executing an entire workflow.

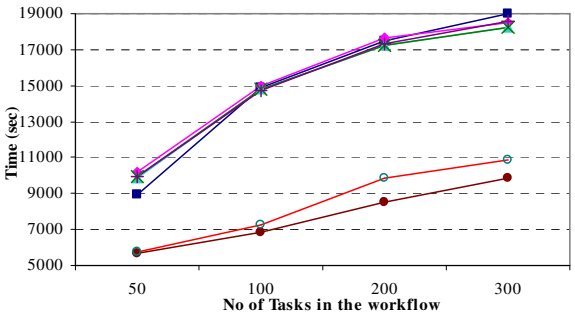
Two sets of experiments were carried out. In the first set, we consider an ideal case where availability and load of Grid resources remain static over time. For this environment, we statically map tasks to resources according to different strategies and execute tasks accordingly. In the next set, we evaluate the strategies on a more realistic scenario where the availability and load of Grid resources vary over time. In this case, the instantaneous load (i.e. number of PEs occupied) for each resource during the simulation was derived from a Gaussian distribution, as performed in [18].

4.3.1 Execution time in static environment. The graphs in Figure 3 plot the execution time of parallel, fork-join and random workflows of 50, 100, 200 and 300 tasks for seven workflow scheduling strategies namely, Myopic, Min-Min, Max-Min, HEFT, DCP-G, GRASP and GA in static environment.

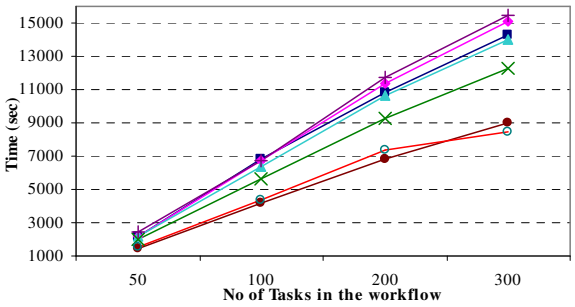
For random workflow (refer to Figure 3(c)), DCP-G can generate schedules with up to 13% less makespan than HEFT which generates better schedule than Myopic, Min-min and Max-min. Since from any task in the random workflow, there can be multiple paths to an exit node, assigning priority to tasks dynamically helps DCP-G to generate better schedules. As GRASP and



(a) Parallel workflow

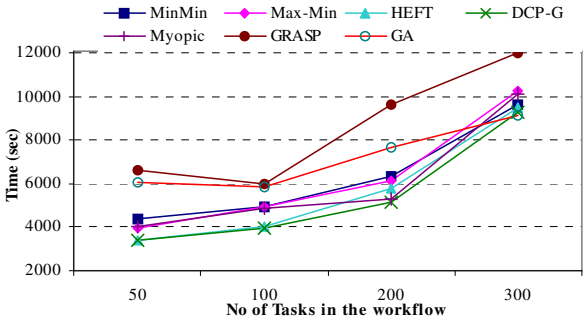


(b) Fork-join workflow

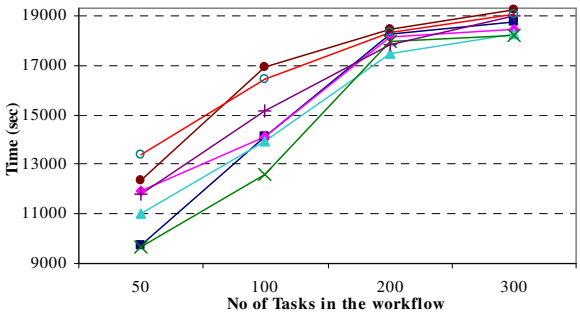


(c) Random workflow

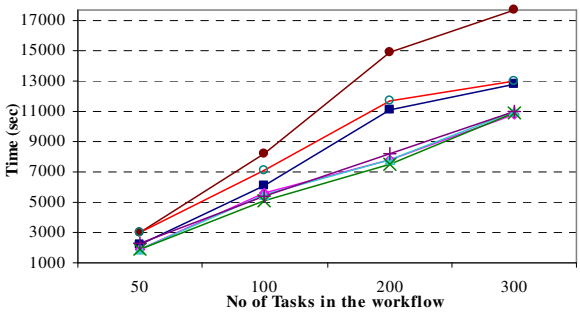
Figure 3. Execution time of different type of workflows for static environment



(a) Parallel workflow



(b) Fork-join workflow



(c) Random workflow

Figure 4. Execution time of different types of workflows for dynamic environment

GA search the entire solution space for the best schedule, they generate 20-30% better schedule than DCP-G.

However, execution time of fork-join workflows (refer to Figure 3(b)) shows a significant difference between heuristics and meta-heuristics based approaches. During the process of task selection for mapping, heuristics-based approaches do not consider the impact of mapping child tasks. Thus, all the heuristic based techniques generate similar schedule with DCP-G being marginally better. However, in a fork-join workflow, a join task depends on the output of all the forked independent tasks that precede it. If this join task is assigned to a resource with low bandwidths to other resources, increase in data transfer time impacts the makespan adversely. However, meta-heuristics (GA, GRASP) consider the impact of mapping not only the parent fork tasks but also the child join tasks, and are therefore, able to generate 40-50% better schedule than DCP-G which is the best among all the heuristics-based methods.

According to Figure 3(a), execution time of parallel workflow exhibits slow exponential growth with the increase in workflow size. The reason is that, unlike fork-join workflows, the number of unmapped ready tasks at every step of scheduling in a parallel workflow always equal to W and a task becomes ready as soon as its parent finishes. Thus, when available resources are less than unmapped ready tasks, the time spent by some of these tasks in waiting to be scheduled results in an increase in the total execution time. In case of parallel workflows, DCP-G and GA generate better schedule than others and the makespan is reduced by least 20%.

Here, the execution time of GRASP rises beyond that of DCP-G as the number of candidate solutions for task mapping increases exponentially with workflow size. This will be explained further in Section 4.3.3.

4.3.2. Execution time in dynamic environment. In dynamic environment, as the resource availability changes over time, resource availability information needs to be continuously updated after a certain period of time and the tasks have to be re-mapped, if necessary, depending on the updated availability of resources. Here, we compare the performance of re-scheduling using DCP-G and other heuristics-based approaches against the static schedules generated by the meta-heuristics.

Figure 4 shows the execution time of different scheduling techniques in dynamic environment, where resource information is updated every 50 seconds. The number of available processing elements, and hence the number of tasks that can start execution in a resource varies with the load on the resource. However, for GA and GRASP, if a resource is heavily-loaded and unavailable, the tasks mapped to that resource have to wait to be executed. This waiting time consequently impacts start time of other dependent tasks and increases the makespan. This is reflected in the poor performance of GA and GRASP in the graphs in Figure 4. Thus, heuristics-based approaches are able to generate up to 30% better schedule than these two meta-heuristics based approaches. Among the heuristics, DCP-G is able to achieve up to 6% better makespan than the others. This is because, in DCP-G, tasks on the critical path waiting to be executed on a heavily-loaded resource are rescheduled on to resources with available processing elements. This reduces the critical path length and therefore, the makespan for the execution.

It can also be seen that heuristic-based approaches perform better in dynamic than static environments for the same workflows and experimental setup. This can be attributed to the fact that not only the load but the resource availability in dynamic environments is updated regularly. This means that the heuristics are able to adapt to better resources being more frequently available and therefore, produce better schedules.

4.3.3 Scheduling time. Table 3 shows the average scheduling time (in milliseconds) for one task of parallel, fork-join and random workflows to generate a single schedule for different scheduling techniques. To generate a single schedule, Myopic, Min-Min, Max-Min and HEFT require nearly 1 millisecond for each task irrespective of workflow size and type whereas the average scheduling time of one task for DCP-G is 16 to 17 milliseconds, and does not vary with the type of workflow as the task selection procedure is independent of workflow structure.

Scheduling time using GRASP increases exponentially not only with the increase of tasks in a workflow but also with change in workflow structure. In each iteration, GRASP creates Restricted Candidate List (RCL) for each unmapped ready task and then selects resource for the task in random. When number of tasks increases, RCL increases exponentially resulting in increased scheduling time. But the size of the RCL is also dependent on workflow structure. For example, when a workflow consists of 300 tasks, parallel and fork-join structures contain 30 tasks in each level, whereas the random structure contains random number of levels as well as random number of tasks in each level. Thus, at every step a parallel workflow has 30 ready tasks, fork-join workflow has maximum 30 ready tasks and average number of ready tasks in each level of random workflow is less than 30. Therefore, scheduling time for random workflow is the lowest and parallel workflow is the highest in this case.

Table 3. Average scheduling time per task

Scheduling Strategy	Random workflow(ms)	Fork-join workflow(ms)	Parallel workflow(ms)
Myopic	1	1	1
Min-Min	1	1	1
Max-Min	1	1	1
HEFT	1	1	1
DCP-G	17	16	16
GRASP	1180	2840	5720
GA	1940	1780	1750

However, scheduling time for GA does not change much with the type of workflow because it executes same number of genetic operations irrespective of workflow structure. But the size of each individual in the solution space is equal to number of tasks in workflow. So, scheduling time increases with the increase in the size of the workflow.

While it is possible to reschedule at regular intervals in GA and GRASP, Table 3 shows that the scheduling times for these are at least 100 times that of DCP-G, and increases with the size of the workflow as well. Hence, we did not incorporate rescheduling for GA and GRASP in the experiments for dynamic environment.

4.3. Discussion

From Figure 3, it is evident that among the heuristics-based scheduling techniques, DCP-G can generate better schedule by up to 20% in static environment, especially for random and parallel workflow, irrespective or workflow size. GA and GRASP can generate more effective schedule than DCP-G for random and fork-join workflow, but they suffer from the problem of higher scheduling time. In our simulation, for parallel workflow of 300 tasks, DCP-G takes 6 seconds to map the tasks to resources, whereas GA and GRASP take 580 and 2076 seconds respectively.

In dynamic environment, heuristics based techniques adapt to the dynamic nature of resources and can avoid performance degradation. But meta-heuristics based techniques perform worse in this situation due to the unavailability of mapped resources at certain intervals. However, in dynamic environment, DCP-G can generate better schedule than other approaches irrespective of workflow type and size.

5. Conclusion and Future Work

In this paper, we have presented DCP-G algorithm for scheduling Grid workflows. Using simulation, we have compared the performance of our proposed approach with other existing heuristic and meta-heuristic based scheduling strategies for different type and size of workflows. The results show that our approach can generate better schedule for most of the type of workflows irrespective of their size particularly when resource availability changes frequently.

In future work, we will endeavor to combine the advantages of DCP-G and GA or GRASP to develop a failure-aware scheduling strategy for a volatile Grid environment. We will develop methods to generate the first schedule using meta-heuristics and then use the concept of DCP whenever rescheduling is necessary due to failure of resources which will enhance robust and efficient execution of e-Science workflow applications on dynamic Grids.

Acknowledgement

This work is partially supported by Australian Research Council (ARC) Discovery Project grant.

References

[1] C. Laity, N. Anagnostou, B. Berriman, J. Good, J. C. Jacob, and D. S. Katz, "Montage: An Astronomical Image Mosaic Service for the NVO", Astronomical Data Analysis Software & Systems (ADASS) XIV, Pasadena, California, October 2004.

[2] P. Blaha, K. Schwarz, G.K.H. Madsen, D. Kvasnicka and J. Luitz, "WIEN2k, An Augmented Plane Wave + Local Orbitals Program for Calculating Crystal Properties", Karlheinz Schwarz, Techn. Universität Wien, Austria, 2001. ISBN 3-9501031-1-2.

[3] J. Yu and R. Buyya, "Taxonomy of Workflow Management Systems for Grid Computing", Journal of Grid Computing, 3(3-4): 171-200, Springer, New York, USA, Sept. 2005.

[4] J. Yu and R. Buyya, "Workflow Scheduling Algorithms for Grid Computing", Tech. Rep., GRIDS-TR-2007-10, University of Melbourne, Australia.

[5] S. Kim, and J. Browne, "A General Approach to Mapping of Parallel Computation upon Multiprocessor Architectures", Proceedings of IEEE International Conference on Parallel Processing, 1988, IEEE press.

[6] Y. Kwok and I. Ahmad, "Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors", IEEE Trans on Par and Dist Systems, 5(7): 506-521, May 1996.

[7] M. Wieczorek, R. Prodan, and T. Fahringer. "Scheduling of Scientific Workflows in the ASKALON Grid Enviornment", ACM SIGMOD Record, 34(3): 56-62, Sept. 2005.

[8] M. Maheswaran, S. Ali, H.J.Siegel, D. Hensgen, and R. Freund, "Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computng Systems", 8th Heterogeneous Computing Workshop (HCW'99), Apr. 1999.

[9] A. Mandal et al., "Scheduling Strategies for Mapping Application Workflows onto the Grid", Proceedings of IEEE International Symposium on High Performance Distributed Computing (HPDC 2005), 2005.

[10] H. Topcuoglu, S. Hariri, and M. Y. Wu. "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing", IEEE Trans on Par and Dist Systems, 13(3): 260-274, March 2002.

[11] T. Fahringer et al., "ASKALON: a tool set for cluster and Grid computing", Concurrency and Computation: Practice and Experience, 17:143-169, Wiley Inter-Science, 2005.

[12] J. Blythe et al., "Task Scheduling Strategies for Workflow-based Applications in Grids", Proceedings of IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005).

[13] D. E. Goldberg, "Genetic Algorithms in Search, Optimization, and Machine Learning", Addison-Wesley, 1989.

[14] I. Foster and C. Kesselman, "The Grid: Blueprint for a New Computing Infrastructure", Morgan Kauffmann Publishers, Inc., 1999.

[15] R. Buyya, and M. Murshed, "GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing", Concurrency and Computation: Practice and Experience, 14(13-15): 1175-1220, Wiley Press, USA, 2002.

[16] P. Rutschmann and D. Theiner, "An Inverse Modeling Approach for the Estimation of Hydrological Model Parameters", Journal of Hydroinformatics, 2005.

[17] W. H. Bell, D. G. Cameron, L. Capozza, A. P. Millar, K. Stockinger, F. Zini, Simulation of Dynamic Grid Replication Strategies in OptorSim, in: Proceedings of the 3rd International Workshop on Grid Computing(GRID '02), Springer-Verlag, Berlin, Germany, Baltimore, MD ,USA, 2002, pp. 46–57.

[18] S. Venugopal, and R. Buyya, "A Set Coverage-based Mapping Heuristic for Scheduling Distributed Data-Intensive Applications on Global Grids", Proceedings of 7th IEEE/ACM International Conference on Grid Computing, Barcelona, Sept. 2006, IEEE CS press.

[19] J. Yu and R. Buyya, "A Budget Constrained Scheduling of Workflow Applications on Utility Grids using Genetic Algorithms", Workshop on Workflows in Support of Large-Scale Science, Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing (HPDC), 2006.