Dynamic Adaptation of DAGs with Uncertainties in Execution Times for Heterogeneous Computing Systems

Qin Zheng

*Advanced Computing Programme, Institute of High Performance Computing, Agency for Science, Technology and Research (A*STAR), Singapore 138632.*

Abstract

In this paper, we consider the problem of schedule DAGs with uncertainties in task execution times. Given an off-line planned schedule based on the estimated task execution times, we first consider when the schedule should be adapted based on the current information about the status of the system. The objective is to limit the number of runtime adaptations and optimize the response time of the DAG. We also consider the case without off-line planned schedules and discuss dynamic planning and adaptation of tasks. We conduct extensive simulation experiments to quantify the performance of the proposed scheduling algorithms.

Keywords: Directed acyclic graphs, scheduling, adaptation, uncertainty, execution time, response time

1 Introduction

Efficient scheduling of application tasks is critical to achieving high performance in parallel and distributed systems [3]. We consider scheduling of workflow applications, which can be represented as a direct acyclic graph (DAG). It is a NP-complete problem as given an application consisting of a number of jobs, either independent jobs or DAGs, and a set of heterogeneous resources, the problem of scheduling the jobs into the resources subject to some criteria is a well-known NP-complete problem [14]. Various static and dynamic heuristics have been developed to solve this problem and in [15] they are classified into list scheduling, clustering, duplication-based algorithms and guided random search methods. Dynamic scheduling assigns each ready task according to the current status of the system. Each time, a dynamic scheduling algorithm is invoked and if a schedule is not available, the whole DAG will not be able to complete. On the other hand, static heuristics plan all tasks based on their expected execution times. However, in many cases, the actual execution time of a task is different from the expected one [3] and this may greatly affect the performance of static algorithms [4]. In [5], the authors point out that considering their average case behavior is not safe, because execution overruns may cause other tasks to miss their deadlines. Usually, in hard real-time feasibility analysis it is assumed the worst case execution time as the task computation time. They comment that this would be too pessimistic, and would cause a waste in processing power.

In [6], approaches to dealing with uncertainties in a scheduling environment are classified into *proactive scheduling* and *reactive scheduling*. The goal of proactive scheduling [7, 8, 9, 3, 1] is to build (static) schedules that are likely to remain valid under different disturbances. In other words, by considering the uncertainty information when producing a schedule, it aims to generate a schedule that is more robust. In [3], a genetic algorithm is developed to schedule DAGs with uncertain (no-deterministic) task execution times to optimize the response time and the robustness. In [1], for independent tasks with uncertainties in processing times, the uncertainties are modeled stochastically and the authors develop both greedy heuristics and global search heuristics.

In reactive scheduling [2, 4, 11, 12], the scheduler decides when and where to execute a task based on the current information about the status of the system and perhaps an existing preliminary schedule. Therefore, a schedule is revised or modified as necessary when the status of the system changes. All the above approaches have an off-line phase where an initial schedule based on the estimated task execution times is generated. In [2], a DAG is partitioned into a number of vertical blocks and tasks within a block are independent. In the runtime phase, the remapper modifies schedules of tasks in a block k using run-time values

for task completion times while tasks in block $k-1$ or before are executing. [4] takes a similar approach for scheduling independent task on the Grid [13]. Runtime information, such as the variation in computation and communication costs and the early release of resources, are taken into account in order to improve the performance of the initial schedule. In [11], at run-time before each task starts execution, the starting time of each task against its estimated start time is evaluated to make a rescheduling decision. In [12], an initial contract that specifies the expected performance of tasks on the assigned resources is created. Due to the uncertainty of the Grid environment, this contract will be probably violated, which triggers the task to be migrated to other resources. Another cause of migration is that a better resource for the task becomes available.

Our work belongs to reactive scheduling. Different from existing approaches in this category, during runtime a task is considered for adaptation during the period from the completion of its first parent to the completion of all its parents. A schedule may become invalid if its parents overrun and the schedule is adapted timely when the timing information from its parents becomes available. Further, on the other hand, when a parent completes earlier than expected, its children are considered for adaptation in order to optimize the response time of the DAG. The algorithms are designed to limit the number of adaptations and will only do so when a schedule becomes invalid or the response time is likely to be improved. We also discuss the inefficiency of off-line planned schedules under uncertainties and present dynamic planning and adaptation of children.

2 Task and System Model

An application is represented by a directed acyclic graph (DAG) $G = (V, E)$ where V is the set of v tasks and E is the set of e directed edges that define precedence relationships among the tasks. A $v \times v$ matrix denotes the amount of communication data to be transmitted between any two tasks. For a task, we use predecessors to refer to tasks that it depends on and descendants to refer to tasks that depend on it. We use parents to refer to its immediate predecessors and children to refer to its immediate descendants.

The system has a finite set of n heterogeneous processors and these processors are assumed to be fully connected. Further, inter-processor communications are performed without contention and computation can be overlapped with communications. These assumptions have been used in [15, 3]. The data transfer rates between processors are represented by a matrix of size $n \times n$. Intra-processor communication cost is assumed to be zero. A $v \times n$ matrix is used to give the estimated execution time to process each task on each processor,

as assumed in [2, 15].

3 Off-line Planning and Effective Dynamic Adaptation Under Task Execution Time Uncertainties

This approach has two phases. In the off-line planning phase, all tasks in the DAG are scheduled, for example using *HEFT* [15]. Note that expected (mean) execution times are needed for off-line algorithms but we do not rely on them as adaptations will happen during runtime if the expected value is inaccurate. As schedules are reserved beforehand, during the runtime phase, the schedule for a task is adapted only when necessary. A schedule is adapted in the following two cases:

1. The task overruns.
2. The schedule for the task becomes invalid when all its parents complete.

If Case 1 happens, the task is rescheduled as in [16]. This is called isolation so as not to affect existing schedules on the same processor for other tasks. Note that it could be adapted on the same processor (i.e., allowed to continue running as in some existing works) if it will not affect other existing schedules. Note that in [16] only the remaining portion of the task is rescheduled. In this paper, we can follow this assumption if it is feasible; otherwise, the task is adapted with its worst-case execution time. Recall that usually, in hard real-time feasibility analysis it is assumed the worst case execution time is known [5].

Case 2 is because some of its parents overrun and the task is not able to receive inputs from them at its scheduled start time. Adaptation for these two cases may fail if a new schedule is not available, which stops this DAG from completion. Also, a scheduling algorithm needs to be invoked to find the new schedule, which takes time. Further, as the adapted schedule is mostly likely to complete later than the initially planned schedule, it results in longer response time for the DAG. To address these issues, we will not wait for all the parents to complete (Case 2). Instead, we propose four cases (Cases 2A, 2B, 3A and 3B) when a schedule should be adapted. The objective is to increase the chance of finding a new schedule and optimize the response time.

3.1 Adaptation of children considering the timings of its parents

Instead of adapting a schedule when all its parents complete (Case 2), the schedule is adapted in the following two cases:

2A. Any of its parents overruns and makes the schedule invalid.

2B. The schedule is invalid at the time its last parent starts running.

Case 2A allows an invalid schedule to be adapted timely upon the completion of an overrun parent. It increases the chance of finding a new schedule and gives time for the scheduling algorithm to do so. The new schedule should be able to receive input from that parent before it starts. Case 2B complements Case 2A and it is beneficial when i) the task only has one parent, ii) only the last parent overruns and makes the schedule invalid, iii) the last parent overruns and completes the latest (which determines the time that the task can start), or iv) the complete times of its parents are very close. That is, the execution of this last parent can be used as a "buffer", which gives time for the scheduling algorithm to find a new schedule if necessary. Further, it is a very good time to make adaptation decision given that the completion times of some parents are known while for the other parents, their actual start times are known. Note that these actual start times may be later than their planned start times, which may cause them to complete later than expected (based on their expected execution times). As the schedule under consideration is planned based on the expected completion times of its parents, it is likely to become invalid. In this case, the schedule is adapted and its start time is determined by the completion times and actual start times of its parents.

Besides increasing the chance of finding a new schedule, by making timely adaptations (once actual timing information is available) in the two cases, the schedule found could complete earlier, which may also reduce the number of adaptation needed for its children. Note that when a task overruns (Case 2A) and before it completes, schedules for some of its children may become invalid. However, we will only adapt them when the overrunning parent completes (Case 2B) as only then its actual completion time is known and the affected child can be adapted with a new valid schedule with respect to this parent. Although children could be adapted proactively according to the worst-case execution time of the overrunning parent (as it is adapted), doing so may lead to "false alarms", i.e., schedules for children appear to be invalid may actually be valid as the overrunning parent is likely to complete before its worst case. Note that we do not need to consider grandparents as even when they overrun, the schedule of a task may still be valid if its parents complete earlier than expected. The number of adaptations by these two cases for a task is at most the number of its parents (which overrun and make the schedule invalid) plus one. In order to limit the number of adaptations, we could allow adaptation for Case 2B only. Therefore, the schedule for each task only needs to be adapted once, which is the same as in Case 2.

3.2 Adaptation of children to optimize the response time of the DAG

In the previous section, a schedule is adapted only when it becomes (or is likely to be) invalid. Therefore, the final response time of the DAG is mostly likely to be longer than the initial one. In order to maintain or optimize the response time, a schedule should also be considered for adaptation when its parents complete earlier than expected. This is because in this case, the task may be able to complete earlier if adapted. However, recall that as all schedules have been reserved beforehand, the objective at the runtime phase is to adapt as less schedules as possible. Also, note that adapting a schedule when its parents complete earlier does not necessarily reduce the response time. Therefore, in this section, a schedule is adapted in the following two cases:

3A. Any of its parents completes earlier than expected and the task is on the critical path.

3B. Any of its parents completes earlier than expected and for that parent the task is the child scheduled to complete the latest.

The objective of Case 3A is to make the task on the critical path complete earlier so as to minimize the response time. By completing earlier, it also minimizes the number of adaptation needed for its children. As a task may not have a child on the critical path, when it completes earlier, Case 3B allows its *critical* child to be adapted. This child could complete earlier, which may lead to reduced response time. Note that at most one child is adapted when any task completes earlier. Therefore, the number of adaptations by these two cases for the whole DAG is at most the number of its tasks that complete earlier.

Note that even if a parent completes earlier, its child may not be able to start earlier. The start time of the child is determined by the completion times of its completed parents and the expected completion times of other parents. Also, when a task completes earlier, we do not need to adapt its grandchild even if it is on the critical path. Instead, the grandchild is better to be adapted later as its parents may overrun.

3.3 Scheduling algorithm

A schedule is adapted when Cases 1, 2A, 2B, 3A or 3B happens. The objective of the scheduling algorithm is to find a new schedule that completes the earliest. It benefits the overrunning cases as the new schedule has better chance to complete timely so as to minimize the impact on its children and the number of adaptations for them. It also benefits the completing earlier case as the task on the critical path or the critical child could complete

the earliest leading to reduced response time. Note that for Cases 1, 3A and 3B, only one task at a time needs to be adapted. However, in Cases 2A and 2B, multiple children may need to be adapted. These children are adapted in the increasing order of the earliest (scheduled) start time of their own children with the objective to minimize the number of adaptations needed for their own children.

4 Efficient Dynamic Planning and Adaptation Under Task Execution Time Uncertainties

This approach does not require off-line planned schedules. During the runtime phase, schedules are planned for certain tasks, which will run in the near future. The rationales and detailed explanation on this dynamic planning approach are given next. This approach can reduce the number of adaptations due to task overrun (Case 1). We propose a generalized Case of Case 3A and a new case for Case 3B to optimize the response time.

4.1 Inefficiency of off-line planned schedules

When task execution times are uncertain, off-line planned schedules may not be necessary as they are likely to be violated often. In Figure 1, suppose that tasks 3, 4, 5, and 7 are scheduled to run in 3, 4, 5, and 7 hours, respectively. When task 3 overruns by 2 hours, the planned schedules for tasks 4 and 5 become invalid and the planned schedule for task 7 is also likely to be invalid. Therefore, a task overrun could make the planned schedules for all its descendants become invalid. Note that it leads to resource inefficiency as these scheduled are reserved.

On the other hand, when a task completes earlier, in order to reduce the response time, its descendants need to be adapted. For example, if task 1 completes earlier, in order to optimize the response time, its dependents (tasks 2, 6 and 7) on the critical path need to be rescheduled and hence their planned schedules are not utilized. Finally, note that during runtime, either scenario (task 3 overruns or task 1 completes earlier) may cause the critical path (for the remaining DAG) to change, as shown in the figure.

4.2 Dynamic planning of children

Instead of off-line planned schedules, tasks are dynamically planned for the near future. Specifically, a task is planned when any of its parents completes or its last parent starts, whichever is earlier. That is, a task is planned only after its parents start running. After

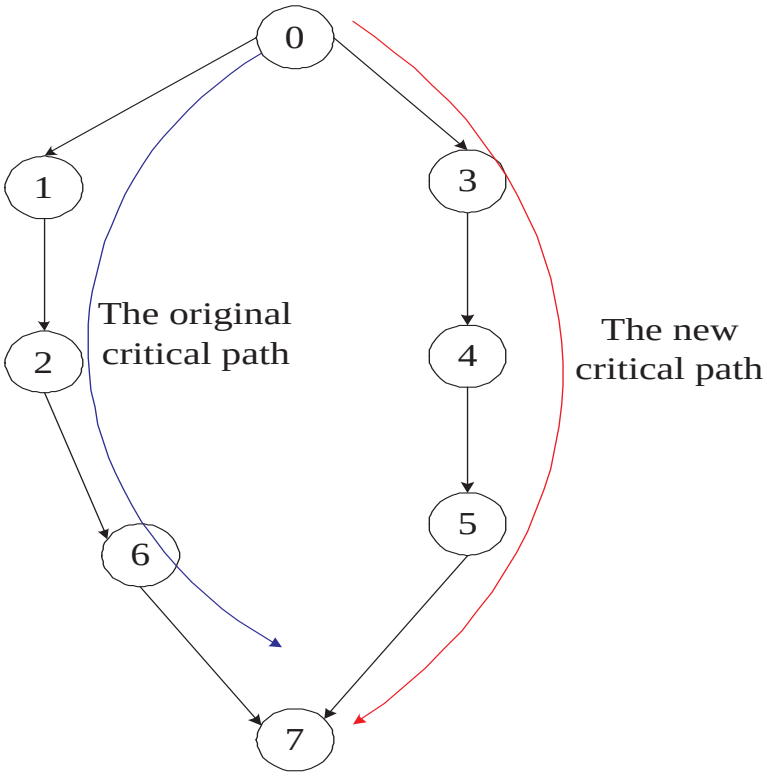

Figure 1: Inefficiency of off-line planned schedules for a DAG and dynamics of the critical path.

all its parents complete, this task will become one of the next set of tasks to run. When a parent completes or the last parent starts, there may be a number of unplanned children. These children are scheduled one by one according to their potential impact on the response time. This impact can be determined based on the expected start time of the child (based on available information on the actual completion time of the parent or the actual start times of all its parents) and its upward rank [15]. The former is similar to the downward rank [15], which is the longest distance from the entry task to this child (exclusive), but is updated with actual (computing and communication) time values of completed tasks and hence is accurate. The upward rank is the length of the critical path from this child (inclusive) to the exit task. Therefore, the child is scheduled according to the expected completion time of the critical path from it to the exit task. Note that the upward ranks only need to be calculated once in off-line and they remain fixed in the runtime phase.

A task is planned based on available timing information of its parents and its schedule may adapted in the next section when more such timing information becomes available (i.e., its parents overrun or complete earlier). It is scheduled according to its maximum (worst-case) running time while the resource will be released/reclaimed if (in most cases) it

completes earlier than the worst case. The objective is to reduce the Case 1 adaptation as when a task overruns, it is less affordable in time to find a new schedule. It is different from Case 2A and Case 2B adaptations where a task may have chances to be adapted a few times before it actually runs. Therefore, by planning according to the worst-case running time, we could minimize Case 1 adaptations which may fail or delay the execution of the DAG. We will demonstrate its effectiveness in simulations. Finally, note that schedules are reserved to the maximum execution times for tasks whose parents are running and these schedules will be executed soon and released upon their completions. In contrary, in the off-line planning, schedules are reserved (to the mean running times) for all tasks and these schedules will be released depending on when they will complete. Further, if robustness against uncertainties is provided in the off-line planning, schedules may have to be reserved to the maximum running times for all tasks. As a result, much more resources are needed and the response time for the DAG is increased significantly.

4.3 Adaptation of children

A schedule is adapted upon Cases 2A or 2B. In the following, we generalize Case 3A to take into account the dynamics of the critical path. In order to optimize the response time, the dynamics of the critical path should be watched closely and tasks on the (new) critical path should be adapted if they could complete earlier. Case 3A only applies to the fixed critical path scenario and it allows a task on the critical path to be adapted if one of its parents completes earlier. If none of its parents completes earlier, a critical task will not be adapted even if other tasks complete earlier and release their resources. Instead, children on these tasks, which are not on the critical path, are adapted. Also, even if a parent of a task completes earlier, the task is not adapted if it is not on the critical path at that time. However, later it may be on the critical path but will not have chance to be adapted.

Given the above considerations, a schedule is adapted if

3A-G. The task is on the critical path when another task (say j) completes earlier than expected.

Here the (critical) task may be on the critical path before task j completes or it is on the (new) critical path after task j completes. Task j may or may not be a parent of the task under consideration. Note that at most one critical task (who has parents running or j is its last parent) is adapted when any task completes earlier. Therefore, the number of adaptations by Case 3A-G for the whole DAG is at most the number of its tasks that

complete earlier than expected. Note that we could further reduce this number by restricting the adaption to happen only when the critical path changes.

Next, we propose a new case for Case 3B and a schedule is adapted if

3B-N. Any of its parents completes earlier than expected and for that parent the task is the child with the largest potential impact on the response time.

Case 3B-N will happen if the task completing earlier (task j) does not have a child on the critical path. It is different from Case 3B as it takes into account the expected start time of a child (based on available information on the actual start times and/or completion times of its parents) and its upward rank. If the critical child completes earlier, the response time could be reduced. Note that for Case 3B-N, at most one child is adapted when any task completes earlier. Therefore, the number of adaptations required for the whole DAG is at most the number of its tasks that complete earlier.

4.4 Scheduling algorithm

A schedule is adapted when Cases 1, 2A, 2B, 3A-G or 3B-N happens. The scheduling algorithm will choose the (new) schedule on the processor where it completes the earliest (with void filling between existing schedules). It benefits the overrunning cases as the new schedule has better chance to complete timely so as to minimize the impact on its children and the number of adaptations. It also benefits the completing earlier case as tasks on the critical path or the critical child may complete earlier leading to reduced response time. The start time of the new schedule is determined by the expected/actual completion times of its parents and the communication time from them to the new schedule. Note that for Cases 1, 3A-G and 3B-N, only one task at a time needs to be adapted. However, in Cases 2A and 2B, multiple children may need to be adapted. The children are adapted according to their potential impact on the response time as well.

References

[1] V. Shestak, J. Smith, A.A. Maciejewski, and H.J. Siegel, "Stochastic robustness metric and its use for static resource allocation," *Journal of Parallel and Distributed Computing*, vol. 68, no. 8, pp. 1157-1173, August 2003.

[2] M. Maheswaran and H.J. Siegel, "A dynamic matching and scheduling algorithm for heterogeneous computing systems," in Proceedings of *the Heterogeneous Computing Workshop*, pp. 5769, 1998.

[3] Z. Shi, E. Jeannot, and J.J. Dongarra, "Robust task scheduling in non-deterministic heterogeneous computing systems," in Proceedings of *the IEEE International Conference on Cluster Computing*, pp. 1-10, 2006.

[4] A.H. Alhusaini, C.S. Raghavendra, and V.K. Prasanna, "Run-time adaptation for grid environments," in Proceedings of *the 15th International Parallel and Distributed Processing Symposium*, pp. 864874, 2001.

[5] K. Kim, L.L. Bello, S.L. Min, and O. Mirabella, "On Relaxing Task Isolation in Overrun Handling to Provide Probabilistic Guarantees to Soft Real-Time Tasks with Varying Execution Times," in Proceedings of *the 14th Euromicro Conference on Real-Time Systems (ECRTS)*, 2002.

[6] A.J. Davenport and J.C. Beck, "A survey of techniques for scheduling with uncertainty," preprint, 2000.

[7] S. Darbha and S. Pande, "A robust compile time method for scheduling task parallelism on distributed memory machines," *The Journal of Supercomputing*, vol. 12, no. 4, pp. 325347, 1998.

[8] L. Boloni and D. Marinescu, "Robust scheduling of metaprograms," *Journal of Scheduling*, vol. 5, no. 5, 2002.

[9] D. England, J. Weissman, and J. Sadagopan, "A new metric for robustness with application to job scheduling," in Proceedings of *HPDC*, pages 135143, 2005.

[10] A. Dogan and F. Ozguner, "Genetic algorithm based scheduling of meta-taskswith stochastic execution times in heterogeneous computing systems," *Cluster Computing*, vol. 7, no. 2, pp. 177190, 2004.

[11] R. Sakellariou and H. Zhao, "A low-cost rescheduling policy for efficient mapping of workflows on grid systems," *Scientific Programming*, vol. 12, no. 4, pp. 253262, 2004.

[12] F. Berman, H. Casanova, A. Chien, K. Cooper, H. Dail, A. Dasgupta, W. Deng, J. Dongarra, L. Johnsson, K. Kennedy, C. Koelbel, B. Liu, X. Liu, A. Mandal, G. Marin, M. Mazina, J. Mellor-Crummey, C. Mendes, A. Olugbile, M. Patel, D. Reed, Z. Shi, O. Sievert, H. Xia, and A. YarKhan, "New grid scheduling and rescheduling methods in the GrADS project," *International Journal of Parallel Programming*, vol. 33, no. 2-3, pp. 209229, 2005.

[13] I. Foster and C. Kesselman, Eds., *The Grid: Blueprint for a Future Computing Infrastructure*. 2nd Edition. Morgan Kaufmann Publishers, 2004.

[14] M. J. Gonzalez, "Deterministic processor scheduling," *ACM Computing Surveys*, vol. 9, no. 3, pp. 173-204, September 1977.

[15] H. Topcuoglu, S. Hariri, and M. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260-274, March 2003.

[16] M.K. Gardner and J.W.S. Liu, "Performance of Algorithms for Scheduling Real-Time Systems with Overrun and Overload," in Proceedings of *the 11th Euromicro Conference on Real-Time Systems*, June 1999.