Opportunistic scheduling in a constraint-rich world

David Johnstone and Steven Bradley
Department of Computer Science
University of Durham
UK, DH1 3LE
Email: {D.I.Johnstone, S.P.Bradley}@dur.ac.uk

***Abstract*— The latest planners and schedulers allow expressive domain modelling and problem definition, particularly with respect to the inclusion of constrained resource usage and inter-task dependencies. This increased complexity removes the ability to guarantee schedulability of a problem at run-time.**

In hard real-time systems, where 'hard' emphasizes the critical nature of meeting task deadines, the estimated worst-case execution time is used in the task representation. If a solely static framework is used to schedule these systems, the pessimism in the prediction of task execution times will lead to unused resources.

This paper describes a framework to include a local dynamic scheduler with a large-scale planner or scheduler. The local scheduler can take advantage of any unused resources by scheduling additional task sets. It can also handle online plan repair by switching between different quality levels of tasks.

I. Introduction

The role of a planner is to provide a linear ordering on a set of jobs to allow a system to progress from an initial state to a defined goal state[1]. A **plan** is a sequence of jobs typically created prior to execution where each job in the plan is a distinct achieving activity. In the popular Planning Rover Domain [1], examples of jobs include navigation between two locations, photographing an object and data transmission. In 2002, the International Planning Competition (IPC) unveiled PDDL2.1 [2], an extended temporal planning language. It relaxed the classical planning assumption that all actions are simultaneous, allowing the language to model jobs with duration. A **job** in PDDL2.1 comprises start effects and preconditions, end effects and preconditions, invariants and continuous effects. Invariants are predicates that must be true over the duration of the job. Continuous effects allow a numeric resource to change continuously over the duration of a job.

Once a plan has been created each job in the plan must be broken down into low-level **tasks** which can be executed directly. Examples of tasks include starting the rover engine, obtaining the current location, turning towards the goal and driving forwards. Schedulers select an ordering on task execution to maintain each task's internal deadlines and resource constraints. Tasks within a job are typically periodic in nature, so a scheduler must schedule each instance of the task so they complete at regular intervals of time. They can also include inter-task dependencies which affect the relative release of the tasks. Dynamic Scheduling is concerned with on-line scheduling, where tasks are selected by the scheduler to execute based upon assigned priorities. Many analysis techniques have been designed that provide guarantees on the task deadlines for diffferent priority assignment algorithms, including algorithms where the priorities can change during execution or remain constant[2].

[1]Partial Order Planners allow concurrency between jobs, but they will not be considered within the scope of this paper.

A. Planning with Uncertainty

In 2004, the 4th IPC presented a new probabilistic track [4]. The aim was to highlight the advancements made in planners and schedulers which model domains where unexpected events occur. An unexpected event could be a task finishing earlier than anticipated or a deadline being brought forward for a job. An off-line planner or scheduler which creates a single schedule cannot respond quickly to either targets of opportunity or unexpected difficulty. Just-In-Case Scheduling[5] (JIC) is a method which allows contingencies within a schedule. A static schedule is created and analysis takes place off-line to find potential break-points. Starting from the point of highest likelihood and continuing while time and space permit, new schedules branching from each break point are created. If a break occurs during execution, the contingency schedule can be executed from the break point and the system will still respond in expected fashion. JIC works well if there are a small number of high priority schedule breaks, but can suffer in large schedules where each new contingency must be fed through all of the branches created.

A less expensive solution to modelling uncertain domains is to use opportunistic planning. Opportunistic Planning [6] constructs a single plan using estimates of the job durations based on a level of certainty obtained through experimentation. The estimate is conservative to ensure the plan is reliable and robust. The aim is to modify the plan using local extensions to reduce any resource wastage during execution. Each extension or **opportunity** acts as a loop to return the executive to a state from which the remainder of the plan can be executed. An opportunity might be taking an extra photograph whilst navigating through terrain. Selection will be based upon the time the rover must transmit its data back to a control satellite and whether there is sufficient memory to store the additional image. There are areas for improvement within this solution. Due to performance guarantees, opportunities are seen as

[2]Dynamic scheduling has been criticised in the past for allowing non-deterministic behaviour in fault-tolerant systems. This problem has now been addressed[3].

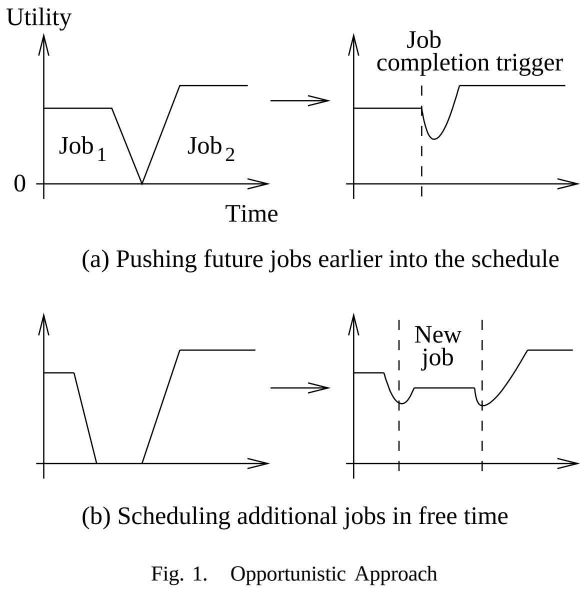
plan extensions rather than as alternatives, and job overrun is not handled. This means the nominal plan must be very conservative and opportunities must be selected quickly and efficiently to maximise the underused resources. This paper proposes a solution to increasing the number of opportunities taken within a plan and minimising the unused resources.

B. Resources

A resource in a planning domain is modelled as a reservoir, a material that can either be produced or consumed by a job. A planner cannot plan down to the task level and hence relies on including factors to model the complex resource dependencies. Jobs using continuous effects define the rate of change of the numerical resources in the domain. The planner handles the resource constraints by ensuring that over the totality of the job the reservoir of the resources will not be exceeded. A system of differential equations must be solved to determine the values of the resources at any time point during the job execution.

There are many reasons why interaction may happen at the task level, including shared memory regions and resources, socket communication and communication through message passing [7]. Schedulers are concerned with the allocation of resource locks to tasks rather than consumption of a physical resource. The problems schedulers face are those concerned with resource contention between multiple tasks. A system can suffer deadlock and individual tasks may miss deadlines when a low priority task prevents a higher priority task from executing because it holds a required resource lock. These problems have led to the necessary development of resource access control protocols, including protocols for fixed priority algorithms [7] and for dynamic priority algorithms [8]. An aim of this research is to handle domains which include complex resource dependencies at both the job and the task levels.

C. Objectives



(a) Pushing future jobs earlier into the schedule

(b) Scheduling additional jobs in free time

Fig. 1. Opportunistic Approach

This paper looks to build upon the Opportunistic framework discussed in Section I-A and improve its efficiency. The primary improvement is to schedule the transition between jobs at the task level. By introducing controlled concurrency between the completing tasks in the old job and the tasks starting in the next job this will provide a greater flexibility within the plan. It will allow future jobs to be pushed further up the schedule and it will also allow additional tasks to be scheduled as part of the opportunistic structure, see Figure 1.

Pedro and Burns [9] provide a mode change analysis which can be performed to check the schedulability of tasks during the change from one mode of operation to another. A mode in their research is equivalent to a job within this work. Their analysis was effective in situations where there were a small number of modes or jobs and all of the transitions could be calculated off-line. Any analysis developed within this paper must be able to handle larger number of jobs and also transitions unknown before execution. For this reason the analysis must be performed during run-time, providing additional information on the current task states but limiting the time available.

The next section of this paper describes an architecture that uses an opportunistic planner and local dynamic scheduler to identify and accept more opportunities. Section 4 is a discussion of an on-line analysis that the scheduler can perform when organising the transition to the next job in the plan. Section 5 describes the test mechanism for proving the reliability and advantages of this research. Section 6 summarises the paper and its potential impact.

II. Structure

This section describes the interactions between a planner and scheduler as part of the three-tier architecture shown in Figure 2.

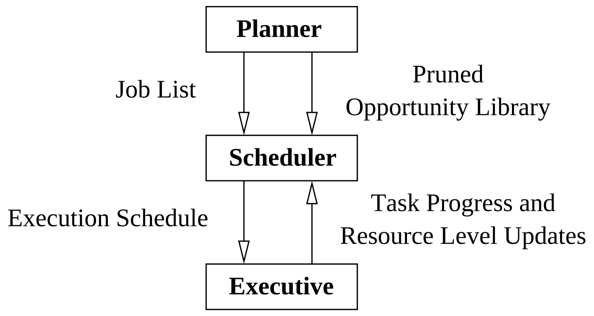


Fig. 2. System Architecture

A planner will pass a job sequence and a library of extra jobs to the scheduler. If an opportunistic planner is used, the job sequence will be a conservative plan and the library will contain a set of additional jobs which will represent opportunities. Each opportunity will include the minimum resource requirements necessary for it to be executed. A position in the job sequence or a time window in which the opportunity must be taken may also be supplied.

Replacement jobs with functionality mirroring that within the job sequence will also be included in the library. This approach is taken from a paper by Marchand and Rutten [10] which concentrates on systems that implement jobs with different levels of quality and cost. The scheduler will select an alternative job to that in the original plan to improve the utility if there are unused resources or to repair a plan that is running behind schedule.

A key advantage in using a dynamic scheduler is exploited by the executive, which can pass back information during run-time to influence the scheduler. In this framework, the progress of tasks and the current resource levels can be obtained during any stage in the execution. This data allows the scheduler to take opportunities or repair a schedule without halting the jobs or referring back to the planner[3]. The algorithm outline is included here to describe when and where these decisions are made.

1) Plan until the horizon
2) Schedule and execute each job in the plan
3) When a job is triggered to complete:
    - Calculate the response time for each old task as it finishes
    - Use these results to calculate the slack and identify unused resources as the job winds down
    - Search the opportunity library for an opportunity or a repair job
    - **If** a job is found **Then** next job = opportunity
      Else next job = next job in plan
    - Calculate start-times for each task in the next job

The algorithm highlights several areas where work is required to implement the opportunistic framework. The next section examines in more detail the scope of the work and the new analysis methods needed.

III. Analysis

The aim of this paper is to add a local on-line scheduler to a powerful off-line system, requiring that the local scheduler is able to model an equal level of complexity as the off-line planner or scheduler. From the planning domain, the scheduler must handle reservoir resources, for example a fuel reservoir where tasks can either consume or supply the fuel. The scheduler must also model the discrete locking of resources, where different tasks require different numbers of locks on a limited resource before they can execute.

One solution is to maintain the blocking criteria proposed by Sha et.al. [7], who implement a priority ceiling protocol in which a task can only be blocked by the longest critical section of any lower priority task. For a discrete resource, this would mean a task could not obtain a lock unless there are enough additional free locks to satisfy a higher priority task with at most one task completing. At implementation this would require one additional item of data stored per resource for each task: the maximum number of locks required of this resource by all higher priority tasks. An additional check can then be placed in the scheduler when allocating a discrete resource.

[3]The planner may be required if an unexpected event occurs which damages the execution sequence and requires a re-plan to correct the job ordering.

At the task level, any analysis must additionally handle **release chains**: a task is only released once another task completes. Release chains are used to model the critical sections and communication in more complex tasks.

Analyses are required to calculate how the old tasks from a job complete gracefully, the search and selection criteria for an opportunity and the start-time allocation for the tasks in the next job. The remainder of this section describes the first of these analysis methods.

When a job signals it has completed and hence will wind-down and end all of its tasks, an exact characterisation can be performed which will calculate the response times for all of the old tasks. These response times can then be used to calculate the slack available during this phase which could be used either to bring forward the tasks from the next job, add in an additional job or repair a schedule running behind time.

This analysis is concerned with at most one period of execution for each task as they complete for the final time. For this reason, the priority assignment algorithm chosen has a lesser impact: using either a static algorithm or one such as the Deadline Monotonic assignment algorithm [11], where a task priority is constant for a period, will lead to a similar analysis. The algorithm outline when using such a priority assignment algorithm is now described.

1) Obtain the list of completing tasks in descending priority order. Neither unreleased tasks in chains or tasks blocked by resource contention will appear in this set.
2) For each task in the list:
    a) Obtain the lists of tasks chained and blocked by this task
    b) Sort the two lists into descending priority order
    c) Pass any resources required to the chained tasks (to represent the larger task defined)
    d) For each blocked task, check if there are enough resources to free:
       If not, remove the task and the remaining tasks in the list. Check and add these removed blocked tasks to one or more of the chained tasks.
    e) Add List of remaining blocked tasks into active list immediately after current task being checked
    f) Merge list of remaining chained tasks into completing task list (use binary chop to find first insertion and then check priorities from that point)
3) Calculate response times and the slack available for each task in the ordered list.

A more complex analysis is required only when the priorities of tasks are allowed to change during execution, for example when using the Least Slack Time assignment algorithm [12]. The additional difficulty is that tasks can preempt the current executing task due to priority changes. Three additional calculations are reqired: when a task can first preempt the executing task, which task will complete first and how much interference will be caused by the pre-emption. It also requires a looser structure as tasks in the original completing list are not guaranteed to finish in that order.

Whichever priority assignment algorithm is chosen, an exact characterisation of the slack available as a job winds-down is available.

IV. Work In Progress

Once the analyses listed in Section III are completed, they will be included within a testing platform. Initially the platform will include a temporal planner and a repair list. The planner will generate real plans from problem domains and pass these along with repair jobs to be scheduled on an implemented simulator. This simulator will schedule and execute the plans whilst running the analyses to make the decisions on assigning start-times and changing jobs within the plan. The aims are two-fold: first to check the correctness of the scheduling by checking the actual slack and deadline potential of the tasks; and second, to judge the effect of repairs on task and job deadlines against schedules where the original plan is strictly followed.

The next stage will be to replace the temporal planner with an opportunistic planner, and include additional opportunities in the repair list. The primary test is to assess the time gained within the schedule and the usage of additional resources which would otherwise have been wasted. As important is the effect of taking opportunities on the rest of the schedule, including missed deadlines or instigating repairs. This may lead to changes in the way that opportunities are represented in the library with respect to their minimum requirements for selection, and a difference in approach when scheduling tasks with hard deadlines to those where meeting the deadline is not critical.

A further decision is when to use a fully-dynamic assignment algorithm. Dynamic priorities are used to provide better processor utilisation, but besides the memory swapping cost there will be the additional cost of performing a more time-expensive analysis for repair and opportunity selection.

The final research goal is to identify the best time to look for opportunities: at the moment this work has assumed the analysis is always performed whilst a job is completing. In less complex domains more time could be spent checking for opportunities. This additional checking may be necessary in unstable domains where there is much unexpected activity. The approach could also provide a benefit when slack time made available in the middle of a job is lost later in the schedule due to the periodic nature of the tasks. A greater frequency of checks will mean more time when tasks are not executing, and taking an opportunity as soon as possible may mean that a larger or more profitable opportunity can not be taken later in the schedule.

Each decision discussed amounts to a trade-off, and the goal of this research is not to provide one definitive answer, but a series of levels which are changed based on the domain and the requirements of the user.

V. Conclusion

Uncertainty occurs in numerous real-world domains. One way to cope with uncertainty is to plan and schedule with conservative estimates on the time and resource requirements of tasks.

One difference between planning and scheduling is the emphasis placed on resources and the way they try to reduce conservatism. Opportunistic Planners seek to identify where a job produces or consumes less resource than anticipated. The Planner can then add extra jobs later in the plan, making use of the excess and maintaining the anticipated levels whilst achieving a greater utility.

A scheduler is primarily concerned with the utilisation of the processor and the extra resources and minimising the time in which they are idle. Improving a schedule means pushing the future jobs earlier if possible or adding additional jobs close to the current time point. Additional tasks in the jobs can then make use of the idle resource.

This paper provides a framework for an Opportunistic Planner and a local dynamic scheduler to work together to minimise both types of conservatism. Tradeoffs have been discussed with respect to the greediness of opportunity selection and the complexity of the domain being modelled. The research plans are to test this architecture and highlight its strength within different application domains featuring both discrete and reservoir resources.

References

[1] M. Fox and D. Long, "The 3rd international planning competition: Results and analysis," in *Journal of Artificial Intelligence Research*, vol. 20, 2003, pp. 1–59.

[2] M. Fox and D. Long, "PDDL2.1 : An extension to PDDL for expressing temporal planning domains," University of Durham, UK, Tech. Rep., April 2003.

[3] S. Poledna, A. Burns, A. Wellings, and P. Barrett, "Replica determinism and flexible scheduling in hard real-time dependable systems," in *IEEE transactions on computers*, vol. 49(2), February 2000.

[4] M. Littman and H. L. S. Younes, "Introduction to the probabilistic track," 4th International Planning Conference Special Issue, Tech. Rep., June 2004.

[5] M. Drummond, J. Bresina, and K. Swanson, "Just-in-case scheduling," in *AAAI-94*, 1994.

[6] M. Fox and D. Long, "Single-trajectory opportunistic planning under uncertainty," in *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, 2002.

[7] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," in *IEEE Transactions on Computers*, vol. 39(9), 1990, pp. 1175–1185.

[8] M.-L. Chen and K.-J. Lin, "Dynamic priority ceilings: A concurrency control protocol for real-time systems," in *Real Time Systems Journal*, vol. 2(4), 1990, pp. 325–346.

[9] P. Pedro and A. Burns, "Schedulability analysis for mode changes in flexible real-time systems," in *10th Euromicro Workshop on Real-Time Systems, Berlin*, June 1998.

[10] H. Marchand and E. Rutten, "Managing multi-mode tasks with time cost and quality levels using optimal discrete control synthesis," in *14th Euromicro Conference on Real-Time Systems (ECRTS'02)*, June 2002.

[11] N. Audsley, A. Burns, and M. Richardson, "Applying new scheduling theory to static priority pre-emptive scheduling," in *Software Engineering Journal*, vol. 8(5), September 1993, pp. 284–292.

[12] J. Y. T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic real-time tasks," in *Performance Evaluation*, vol. 2, 1982, pp. 37–250.