Minimising Task Migration And Priority Changes In Mode Transitions [*]

Paul Emberson and Iain Bate
Department of Computer Science
University of York
York, YO10 5DD
{paul.emberson, iain.bate}@cs.york.ac.uk

Abstract

Handling mode changes is one of the most complex and important problems for real-time systems designers. The challenge is to move a system from running one set of software to another while still achieving the quality of service guarantees necessary. There has been previous work which concentrated on how to perform scheduling and timing analysis of mode changes. However, a common theme of all this research is that if the system's schedule and allocation is chosen to minimise the set of differences between modes then the mode transition problem can be performed more easily and quickly. This paper investigates how this can be achieved.

1 Introduction

There are many real-time applications where the set of running tasks changes throughout the lifetime of the application. When a system moves from one set of tasks to another, it undergoes a *mode change*. There are typically two incentives for doing this: a change in the mode of operation of the system or to adapt to a change of environment. Examples of a change in mode of operation is a flight control system which has different modes for take-off, in-flight cruising and landing [3, 11]. Other general operational modes can be identified including: initialisation, maintenance, low power, fault recovery and emergency [11].

Given requirements for two different modes, one of four things can happen to a task when the system moves from the first mode to the second: it continues running as before, it stops running, it starts running or it continues running but with a change to its attributes (period, deadline, etc). It is also possible for messages to be removed from or introduced into the system. This could be as a result of tasks stopping and starting, or a task may begin to send messages when a mode change occurs. A change to the attributes of a task may come from a task being required to read data from a sensor more quickly [12].

Systems can handle mode changes in different ways. Some may wait until there is sufficient idle time to perform the mode change [15] without loss of service. Others may have to give a reduced level of service or even temporarily stop operating so that a mode change can take place.

On occasion, it is necessary to migrate a task to another part of the system when a mode change occurs. There is a considerable overhead to task migration, usually associated with transferring a large amount of state information [6]. Depending on the architecture, code has to be migrated or must be present on all nodes on which the task will run. Mode changes should be prompt as tasks running in the new mode may be required to complete before a deadline following the mode change request [11].

There are also benefits in reducing the number of priority changes between modes. Ideally, each task will have a single unique priority. Otherwise, during a mode change, more priority levels may be needed to maintain the required priority ordering before during and after the mode change.

The focus of this work is to configure tasks and messages within a system in such a way that task migrations and priority changes are minimised between mode changes. Whether a migration or priority change is required is based on analysis of the schedulability of tasks and messages before and after the mode change.

The two following small examples will further motivate and explain the aims of this work. To maintain simplicity of the example, task attributes will be reduced to just the utilisation of the task and it will be assumed that the tasks on a processor may be scheduled if the total task utilisation for that processor is less than 100%. More detailed analysis will be used in later examples presented in the evaluation.

Table 1 shows the percentage utilisation requirements of four tasks in two modes. In the transition from mode 1 to mode 2, task A requires an increase in utilisation (e.g. from an increase in frequency or execution time) and task D stops running. The platform on which these tasks must run has

Task	Utilisation		Allocation		Allocation
	Mode 1	Mode 2	Mode 1	Mode 2	Unchanged
A	60	80	P1	P1	✓
B	40	40	P1	P2	✗
C	50	50	P2	P2	✓
D	50	0	P2	-	✓

Table 1. Example 1

[*]This work is funded by the Software Engineering By Automated Search (SEBASE) program, EPSRC Grant EP/D050618/1.

Task	Utilisation	
	Mode 1	Mode 2
A	50	80
B	50	50
C	50	50
D	50	0

Table 2. Example 2

Allocation X				Allocation Y			
Task	M 1	M 2	Alloc. Unch.	Task	M 1	M 2	Alloc. Unch.
A	P1	P1	✓	A	P1	P1	✓
B	P1	P2	✗	B	P2	P2	✓
C	P2	P2	✓	C	P2	P2	✓
D	P2	-	✓	D	P1	-	✓

Table 3. Solutions to example 2

two processors, P1 and P2 available. Assuming homogeneity of processors, so that processor labels can be swapped without loss of generality, there is only a single possible allocation for mode 1 that will allow all tasks to be scheduled. Task A cannot be allocated to the same processor as tasks C or D as this would require over 100% utilisation of the processor. This results in task A being paired with task B in mode 1. When the transition is made to mode 2, task B can no longer run alongside task A and must migrate to the other processor where capacity has become available after task D terminated.

There are two ways in which the tasks may be allocated to be schedulable in mode 2. Either, as shown in table 1, with A on P1 and B and C on P2 or with A on P2 and B and C on P1. Note that when selecting an allocation for the second mode, the processors may no longer be considered homogeneous since the number of migrations must be measured relative to the allocation chosen for mode 1. These two possible solutions for mode 2 require 1 migration and 2 migrations respectively with the former being identified in the right hand column of table 1.

A slightly modified version of the first example is shown in table 2. In this example, all tasks have a utilisation of 50% in mode 1. This means that there are three possible ways of grouping the tasks in the allocation of mode 1 so that they all can be scheduled. Two of these possible solutions are shown in table 3.

If allocation X is selected then at least one migration is required for the tasks to also be schedulable in mode 2. However, if allocation Y is selected then it is possible to move from mode 1 to mode 2 with no task migrations. This illustrates the fact that, when selecting an allocation for a particular mode, information regarding transitions to other possible mode transitions is required to achieve minimal task migration. Note that there is only a single valid task grouping for mode 2. If the allocation for mode 2 had been selected first, then this would have lead more naturally to using allocation Y as a solution for both modes.

The structure of this paper is as follows. Section 2 describes our model of the task allocation problem. Section 3 describes previous work on mode changes and task allocation. Section 4 covers how simulated annealing can be applied to this problem. Section 5 describes three possible solution methods for minimising changes between mode transitions and each of these methods is evaluated in sections 6 and 7 for two modes and then systems with a larger number of modes. Finally, conclusions are drawn in section 8.

User Input				Configuration	
Object	WCET	Period	Deadline	Alloc.	Priority
τ_1	C_1	T_1	D_1	A_1	P_1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
τ_n	C_n	T_n	D_n	A_n	P_n

Table 4. Attribute and configuration data for tasks

2 Task allocation problem

The basic task allocation problem is to find an allocation of tasks to processors so that all tasks can be scheduled. Depending on the analysis used for testing schedulability, this can also involve discovering suitable task attributes such as priorities. For systems where tasks must communicate with messages, an allocation of message to networks must also be found along with suitable message attributes.

2.1 Software architecture

The model used for the software architecture is as follows. Each task or message has a set of timing properties and timing requirements. For a task, these are worst case execution time (WCET), period and deadline. For a message, the timing properties and requirements are worst case communication time (WCCT) derived from the size of the message, period and deadline.

In addition to timing properties, the sending and receiving task must also be specified for each message.

Table 4 shows the data associated with each task. The left hand side columns show attributes which are specified in the requirements. The right hand two columns show the allocation and priority which need to be found. The allocations and priorities of all tasks in the system describe the system configuration.

The input requirements for tasks and messages can be used along with its configuration to perform scheduling analysis which will calculate response times for tasks and messages. If all response times are less than or equal to the deadline for the corresponding object, then the timing requirements for the system have been met.

2.2 Hardware architecture

The hardware architecture model is a set of processors connected with bi-directional network links. Each link has

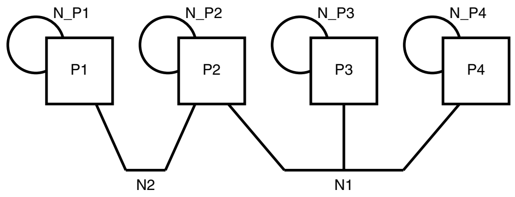

Figure 1. Example hardware architecture

a communication speed and latency which determines how long it takes to send a message over the link. All processing nodes have a link to themselves to allow message passing between tasks on the same processing node. These links will usually have a much higher speed than links between processors so that the impact of sending messages between tasks on the same processor is much lower than sending messages between processors. Links may join more than two nodes. Figure 1 shows four processing nodes connected with two network links and an additional four links for intra-processor communication. The example hardware in figure 1 shows it is not necessary for all processors to be directly connected to each other so the availability of networks for message allocation is dependent upon where the sending and receiving tasks are allocated.

3 Related Work

Much work on mode changes has concentrated on how to schedule tasks before, during and after a mode change [12, 15, 13, 8, 11]. This either assumes a uni-processor system or assumes that mode changes are contained within each processor of a distributed system [8].

Task allocation and distributed scheduling are both NP-hard problems [14]. As distributed real-time embedded systems become larger and more complex, creating an allocation manually becomes infeasible. Automatic generation has been tackled with a range of heuristic techniques including branch and bound [10, 9] and simulated annealing [14, 2]. Simulated annealing is chosen here since it can adapt to a dynamically changing cost function. The need for this is demonstrated in section 5. Our system model is more general and realistic than both [14] and [2] in that we don't assume a fixed time to send messages and allow varying processor - network topologies.

Considering multiple sets of requirements such as multiple modes when selecting a configuration adds significant difficulty to the problem. The rest of this paper is dedicated to describing and evaluating methods by which this may be achieved. The work is an extension of that found in [1], which considered solving task allocation problems in the presence of changing requirements. To our knowledge, there is no previous work which considers allocation problems in relation to minimising the cost of mode changes.

```
$\Omega = \{\omega_0, \ldots, \omega_N\}$          /* Solution space */
$f : \Omega \to [0,1]$                  /* Cost function */
$\psi \in \Omega$                       /* Initial configuration */
$\omega^* = \psi$                       /* Best configuration */
$\omega = \psi$                         /* Current configuration */
$t = t_0$                               /* Set initial temperature */
$\alpha = 0.99$                         /* Cooling factor */
do
    $i = 0$
    do
        $\omega' = modify\_config(\omega)$
        $\delta = f(\omega') - f(\omega)$
        $R =$ random value $\in [0,1]$
        if $(R < e^{-\frac{\delta}{t}})$ then $\omega = \omega'$ endif
        if $(f(\omega) \leq f(\omega^*))$ then $\omega^* = \omega$ endif
        $i = i + 1$
    until ($i = M$ or (stopping condition))
    $t = \alpha t$
until (stopping condition)
```

Figure 2. Simulated annealing

4 Search Method

The solution method chosen makes use of the simulated annealing meta-heuristic search technique [4]. The algorithm makes small adjustments to a potential solution and evaluates each solution for its quality. The quality measurement is given in terms of a cost with the aim being to find the solution which produces the minimal cost value. The algorithm, as applied to the task allocation problem, is described in figure 2. To move from one solution to another, a randomly modified version of the existing solution is chosen by the function $modify\_config()$. This function randomly chooses a task or message. The function then performs a randomly selected change. The changes are changing the object's run-time order (e.g. priority or slot position), or changing the object's allocation.

The cost function used takes a weighted mean of cost function components. Each component is normalised to return a value between 0 and 1 so that the overall cost value is also in the range $[0, 1]$. The set of components currently being used to find the solution to a single problem are described below. Not all of these are required to solve task allocation problems, but previous work has shown them to be useful heuristics for improving search efficiency and effectiveness. For example, *schedulable object sensitivity* helps improve the search as it distinguishes between two solutions where each has the same number of unschedulable tasks. More details and formulae for calculating these components are given in [1].

- Unreachable tasks - dependent tasks allocated to unconnected processors
- Unschedulable tasks/messages - proportion of tasks/messages which miss their deadline.

- Invalid input/output allocations - objects allocated such that they can't receive/send data as their allocated scheduler is not connected to the scheduler where the incoming/outgoing data resides.
- Invalid arrangements - proportion of objects for which scheduling analysis could not be performed. This is usually due to a circular dependency, e.g. the schedule ordering dictating that task B runs before task A where as the precedence order should be task A before B.
- Unschedulable system - binary valued function which indicates whether all objects are schedulable.
- Schedulable object sensitivity - calculates a value based on how much execution times must increase/decrease for the system to become unschedulable/schedulable.
- Load balancing - calculates the variance of utilisation of the processors.
- Dependent schedulable object grouping - a metric for measuring whether tasks/messages which are dependent upon each other are allocated to the same processor/network.
- Separated dependent tasks - calculates whether tasks which are adjacent to each other in a transaction are allocated to the same processor.
- Over utilised schedulers - penalises any allocations where a scheduler is over 100% utilised.

Many of the components are based on results from scheduling analysis. However, the method is independent of the scheduling model. All that is required is that the model can be analysed to produce worst case response times for the schedulable objects. For example, dual priority multiprocessor scheduling with support for aperiodic tasks could be used.

The cost function f is calculated from the scalar product of a vector of the cost function components and a weightings vector.

$$\mathbf{g} = (g_1, \ldots, g_n)^T \tag{1}$$

$$\mathbf{w} = (w_1, \ldots, w_n)^T \text{where } w_i \in \mathbb{R}\; \forall\, i \tag{2}$$

$$f = \frac{\mathbf{g} \cdot \mathbf{w}}{\sum_1^n w_i} \tag{3}$$

4.1 Components To Measure Changes

Since the aim of the methods described in this paper is to minimise migrations and priority changes between modes, cost function components are needed which measure how many changes there are between two configurations. It is assumed the system has a unique mode before and after the change. There are four such cost components which will be described in detail: task allocation changes, message allocation changes, task priority changes and message priority changes. Separate components are defined for tasks and messages so that the importance of minimising change may be weighted differently.

It is assumed that there is a current configuration and a baseline configuration for which the changes between them can be measured. Since these may be for two separate modes, it is possible that not all objects will exist in both configurations. Therefore, change is only measured for objects present in both configurations.

T, M, P, N is the set of tasks, messages, processors and networks in the current configuration respectively. T', M', P', N' are the equivalent sets for the baseline configuration.

The components to measure task allocation changes (g_{tac}) and message allocation changes (g_{mac}) are as follows:

$$g_{tac} = \frac{\#\{i \in (T \cap T') : A_i \neq A'_i\}}{\#(T \cap T')} \tag{4}$$

$$g_{mac} = \frac{\#\{i \in (M \cap M') : A_i \neq A'_i\}}{\#(M \cap M')} \tag{5}$$

where A_i is the current allocation of the object and A'_i is the allocation in the baseline configuration.

There are two factors which prevent a priority change metric based on directly comparing priority values of equivalent objects. The assigned priority values may be different between two configurations with very few differences between priority order. For example, two sets of tasks with the same priority order but different numbering schemes should be considered equivalent as one can easily be remapped to the other. Priority comparisons only make sense for objects allocated to the same scheduler. Therefore priority comparisons are only made between objects which are common to a particular scheduler in both configurations. Note that this differs to the formulae given in [1].

The priority difference metric is based on Spearman's rank correlation coefficient. This requires the two sets of objects for the old and new configurations to be ranked in priority order.

Let X and Y be sets of schedulable objects. For each object $i \in (X \cap Y)$, $\mathrm{R}_X(i)$ is the rank of object i in X and $\mathrm{R}_Y(i)$ is the rank of the same object in Y. The basis of the priority comparison metric is

$$n = \#(X \cap Y) \tag{6}$$

$$p(X,Y) = \frac{3}{n(n^2-1)} \sum_{i \in (X \cap Y)} (\mathrm{R}_X(i) - \mathrm{R}_Y(i))^2 \tag{7}$$

For a given scheduler, s, let $O(s)$ be the set of objects on that scheduler in the current configuration and $O'(s)$ be the equivalent set in the baseline configuration. The components for task priority changes (g_{tpc}) and message priority changes (g_{mpc}) are

$$g_{tpc} = \frac{1}{\#(P \cap P')} \sum_{s \in (P \cap P')} p(O(s), O'(s)) \qquad (8)$$

$$g_{mpc} = \frac{1}{\#(N \cap N')} \sum_{s \in (N \cap N')} p(O(s), O'(s)) \qquad (9)$$

4.2 Component Weightings

When setting weightings for cost components, allocation changes should be weighted more highly than priority changes. Since priority change costs only come into effect when allocations are the same, penalising priority changes too highly will block moves which decrease allocation differences but increase the priority change cost value. Subsequent moves may then be able to decrease the priority ordering differences.

For our work allocation changes are weighted in a ratio of 3 to 1 compared to priority changes. The task and message schedulability tests are weighted twice as highly as the penalties for allocation changes.

5 Solution Methods

In the absence of previous approaches to minimising changes in mode transitions, three approaches to the problem have been identified for consideration. All make use of the simulated annealing algorithm previously described. Following the descriptions of all three methods, more detailed analysis of each one is given.

Each of the methods will be described in terms of changing from mode M_1 to mode M_2. Extensions of the methods to cope with systems with more than two modes will be considered later in section 7.

Sequential Method. The sequential method finds a solution for M_1 without any consideration for M_2. It then tries to find a solution for M_2 using the solution found for M_1 as a starting point and in addition to finding a valid schedulable solution for M_2, attempts to minimise the number of changes between the solutions.

Simultaneous Method. The simultaneous method attempts to simultaneously find a solution for both modes. The design space is over the configurations of all tasks and messages in both systems. This forms a so-called super-configuration for both systems. Tasks or messages which are in both systems are therefore configured identically. If a suitable super-configuration can be found that meets requirements for both modes then a suitable configuration for each mode can be extracted which have zero changes according to the metrics defined in equations (4), (5), (8), (9).

If the super-configuration is only valid for a single mode, say M_1, then the same second step as the sequential method can be applied to generate a configuration for M_2. It is possible to weight the search in favour of finding a feasible solution for one of the modes when searching for the super-configuration.

Parallel Method. The third method, like the simultaneous method, tries to find a configuration for both modes at the same time. However, a separate configuration for each mode is maintained in separate searches. In order for the differences between the two configuration to be minimised each search periodically writes its current best solution and reads the current best solution of the other mode. Each search tries to find a valid solution for its mode while trying to minimise differences from the current best found for the alternate mode. Every time either search finds a new best solution, the baseline configuration for comparison of the other search changes. Whilst the changing of the current best in one search does not change the current best solution in the other, it does change the cost recorded against it. Therefore it may soon become worse that other solutions found which are more similar to that of the other search, hence causing the two solutions to converge.

The dynamically changing landscape for the quality of solutions are a particular feature of this method that make it appropriate for a meta-heuristic search which is able to revisit solutions which may have once been rejected. If such solutions had been bounded in a standard branch and bound style algorithm, this would not be possible.

If there are differences between the solutions found after both searches complete, then a second step can also be run for the parallel method. For example, the solution found for M_1 can be used as a starting point to find a solution for M_2. It is possible that this may produce a solution for M_2 with fewer differences than that found by the search for M_2. The benefit gained from running this second step is likely to depend on how long the first parallel search is run for.

Each of these methods will now be analysed in more detail using small examples which can be analysed manually.

5.1 Sequential Method

The earlier examples and their solutions shown in tables 1, 2 and 3 show that the success of this method may depend on the order in which the configurations are generated. For the example in table 1, both modes only have a single possible grouping under which all all objects are schedulable. In this case, the method would be expected to achieve the optimal result of a single allocation change regardless of whether mode 1 or mode 2 is generated first. However, in the example shown in table 2, mode 1 has multiple solutions under which all objects are schedulable. If allocation X for mode 1 from table 3 was generated then it would be impossible to achieve the optimal solution in the second step. Disregarding other discriminating factors for choosing between solutions in mode 1, the method would only achieve the optimal result on occasions where the randomness in the search algorithm chose allocation Y. This indicates that benefit can be gained from generating a number of initial solutions if this method is used. If mode 2 was generated first, then as in the first example, there is only a single possible feasible grouping. When mode 1 is generated using this solution from mode 2 as a baseline, it would be expected that the method would obtain the optimal result on every occasion.

These observations imply that, when using this method, initial solutions should be generated for both modes in the first step with the second step attempting to minimise the changes in the transition to the alternative mode. Since the quality of the achievable final solution is dependent upon the solution used for the baseline, it is likely to be beneficial to generate multiple baselines for both modes and then attempt to move to the alternate mode. The sequence of modes which is likely to obtain the best solution is that of handling the mode with fewer feasible solutions within the design space first. In general, there is no way to know which mode this is and hence all sequences must be tried. There may be situations where this can be estimated. For example, if two modes are of similar size in terms of tasks and messages but one has significantly higher utilisation.

5.2 Simultaneous Method

The first step of this method is to produce a super-configuration which contains configuration details for all tasks and messages in both modes.

The cost function f, previously described in equation (3) is replaced with the following:

$$F = \frac{W_1 f(\mu_1) + \cdots + W_N f(\mu_N)}{\sum_{i=1}^{N} W_i} \qquad (10)$$

where N is the number of modes. μ_i represents the requirements of mode i with the super configuration applied. The weightings $W_1, \ldots, W_N$ can be used to weight the importance of each mode.

To begin analysing this method, weightings for the modes will be assumed to be the same. It is simple to see that this method should always successfully solve the problem in table 2 since there exists a single configuration for both modes 1 and 2 for which all tasks are schedulable.

Less clear is the fact that it will also successfully find an optimal solution to the problem given in table 1. There is no one single configuration that can schedule both modes so the search should find a solution which minimises the number of unschedulable tasks over both modes. The search may conceivably produce one of two solutions. If it chooses to schedule all tasks under mode 1, then only task B will be unschedulable in mode 2. Alternatively, it may configure tasks A, B and C to be schedulable in mode 2. With this configuration, these tasks will also be schedulable in mode 1, leaving only D unschedulable in mode 1 wherever it is allocated. The super-configuration can then be used as a

Task	Utilisation	
	Mode 1	Mode 2
A	60	80
B	40	40
C	5	5
D	5	5
E	80	0

Table 5. Example 3

Allocation W				Allocation X			
Task	M 1	M 2	Alloc. Unch.	Task	M 1	M 2	Alloc. Unch.
A	P1	P1	✓	A	P1	P1	✓
B	P1	P2	✗	B	P1	P2	✗
C	P2	P2	✓	C	P2	P1	✗
D	P2	P2	✓	D	P2	P2	✓
E	P2	-	✓	E	P2	-	✓
Allocation Y				**Allocation Z**			
A	P1	P1	✓	A	P1	P1	✓
B	P1	P2	✗	B	P1	P2	✗
C	P2	P2	✓	C	P2	P1	✗
D	P2	P1	✗	D	P2	P1	✗
E	P2	-	✓	E	P2	-	✓

Table 6. Solutions to example 3

baseline for the mode which it does not solve to work from. If either of the two solutions are found, a feasible solution can be found for the other mode with a single allocation change.

The task sets in table 5 give an example where this method may be less successful. There is no single configuration where both modes can be scheduled.

A solution is given, called allocation W, in table 6 which shows how the modes can be scheduled with a single task migration. However, if any of the mode 2 allocations in table 6 are taken and applied to mode 1 all of tasks A, B, C and D will be schedulable leaving a single unschedulable task, E. If other differentiating metrics such as sensitivity analysis or load balancing are ignored then any of the mode 2 allocations from X, Y and Z will be considered equivalent to W but do not allow a transition to mode 1 with only a single migration. There is only a single schedulable configuration for mode 1. On some occasions the search will output this configuration, which when applied to mode 2, also leaves only a single task, B, unschedulable. If this is the case then the second step of the method should be able to find the mode 2 allocation in allocation W which requires only a single migration. This indicates that for example 3, it would be better to concentrate on producing a schedulable solution for mode 1 rather than mode 2 in the initial step. If the weighting for mode 1 is higher than for mode 2 then unschedulable tasks in mode 1 will be penalised more heavily than in mode 2 and the method should always produce an initial configuration which allows transition to mode 2 with minimal changes.

Therefore, in the general case it is necessary to run the initial step to find the super-configuration multiple times with each mode weighted higher on different runs. This should produce a schedulable solution to the highly weighted mode so the second step finds a configuration for the lower weighted mode. However, by considering both modes in the initial step, the initial configurations produced should allow transitions to take place with fewer changes than the sequential method. Note that the sequential method is a specific case of this method where the lower weighted mode is given a weighting of 0.

In terms of processing required, this method is more expensive than the first. When no single super-configuration can solve both modes, the search must be run several times with each mode weighted highest. This is the same number of searches as the first method, but in the second method a cost function evaluation must be performed on each mode rather than just a single one as in the sequential method.

5.3 Parallel Method

The final method searches for independent solutions to both modes while exchanging information so that the solutions converge. This method should find solutions which minimise the number of migrations for all of the previous 3 small examples. For example, if the search for a solution to mode 2 initially picks a configuration which does not allow the transition from mode 1 with a single migration, it will subsequently be pulled towards the correct solution, once the search for a mode 1 solution has found the only feasible solution.

In the general case, with limited time to find the initial configuration, the number of differences indicated between the two best solutions may in fact be pessimistic and could quickly be reduced with the use of a second step. This can easily be seen by considering the example of finding configurations for two identical modes. If run for a limited amount of time, each search may find a valid solution which differs from the other. However, with the knowledge that both modes are the same, either of the configurations produced is valid for the other mode with no changes. Therefore, the number of differences reported when both of the parallel searches terminate is an upper bound on the number of changes required. A second step should be taken where each of the configurations produced is used as a baseline to find a configuration for the other mode. However, a benefit of this method is that if it is run for sufficient time, it is expected that it will find a good solution without a need for subsequent searches to be run.

A benefit of this method is that the weightings between task / message and allocation / priority changes are considered in the first step. This allows, for example, initial configurations to be produced which are tuned to preferring the minimisation of message changes to task changes. The previous methods can only do this in the second step and this may not be compatible with the baseline that has been produced in the first step.

6 Two Mode Evaluations

All the discussions so far are abstract of particular scheduling and timing analysis. For the purposes of the evaluation, the scheduling approach assumed is fixed priority scheduling and the analysis is performed using the dynamic offset approach described in [7]. The method could potentially be adapted to other scheduling schemes, such as EDF.

A range of examples were taken to evaluate the different approaches. In the first stage of the evaluation, small examples are used to build an understanding of the methods when applied to a simple two mode system. Then, an investigation is performed of the effectiveness and scalability of the methods for more complex examples.

6.1 Simple Two Mode Systems

Experiment with the three small examples described earlier were used to confirm that the behaviour of each method was in accordance with the analysis. To achieve the correct utilisations, all tasks were given a period of 100 and a worst case execution time equal to that of the appropriate utilisation values in tables 1, 2 and 5.

For the sequential method, 10 baselines were produced for each example, 5 for each mode considered. For each baseline, a subsequent search was performed to produce the mode transition to the other mode, attempting to minimise changes in allocation. For example 1, the optimal solution of a single change was found in all 10 cases. For example 2, the optimal solution was found in all but one case. As expected, this occurred when mode 1 was used to produce the baseline. The fact that the non-optimal solution only occurred once was better than predicted in the analysis. For example 3, the optimal solution was found in all cases. Again, this is better than expected. This is due to the fact that other aspects of the cost function such as the sensitivity metric are intended to increase flexibility in the solution and seem to coincide with finding a baseline solution which allows minimal change.

The simultaneous method also produced 10 baselines for each example. In this case 5 were produced with one mode weighted higher and 5 with the other. The more highly weighted mode was weighted 3 times higher than the lower. For example 1, optimal results were achieved every time when mode 1 was weighted higher and never when mode 2 was given preference. An optimal solution of no changes was achieved for every baseline in example 2. The results for example 3 were as for mode 1, with optimal results only being achieved when mode 1 was given the higher weighting.

The parallel method does not have a preference between modes in its initial search. Therefore, the experiment was conducted 10 times for each example. The optimal solution

Mode	Tasks	Messages	Avg Utilisation
Deps_1	24	22	72.5
Deps_2	24	25	72.5
Pedro_1	35	0	66.56
Pedro_2	35	0	64.24

Table 7. 2 mode example task sets

was found in all cases. No second step was required for the parallel method for these small examples.

6.2 Larger Two Mode Systems

Two further two mode examples were considered. The first one, labelled as *Deps* in table 7 contained messages and hence task dependencies. The hardware platform contained 4 processors, split into two clusters of two processors so communication was possible between clusters.

The same experimental was followed as in the previous examples. For the sequential method, from the 10 trials run, the best solution found achieved the following values for the change metrics: 0.04167, 0.04375, 0.00000, 0.04040 from equations (4), (5), (8) and (9) respectively. Simply summing these values gives a change score of 0.12582. The average total change score was 0.6815 showing that many runs of the sequential method were not near the best score.

The simultaneous method was the best performing method for this example. It found a solution with no changes at all on 9 of the 10 trials. On the other trial, it actually failed to find a feasible solution for the highest weighted mode within the limited number of moves allowed. It is more difficult for this method to find a schedulable solution for the higher weighted mode than for the sequential method in the same amount of time. This is because it is trying to solve more than one problem at a time.

The parallel method found the optimal solution of no changes on 3 of the 10 trials. The average change score was 0.10433, showing it to be better and more consistent than the first method.

These results show the simultaneous method's strength in being able to find the optimal solution when it is possible to find a single super-configuration for both modes. The parallel method always found a schedulable solution and was much better at minimising changes compared to the first method. It did not find the optimal solution as reliably as the simultaneous method.

The second example, labelled *Pedro* in table 7, was taken from an example given in [8]. This example was designed for mode changes taking place within processors only. Therefore, it was known that this example could be solved with no allocation changes. There are also no dependencies in the example which means that optimal priority orderings can be found using deadline monotonic priority ordering [5]. Using deadline monotonic priority ordering for each mode achieves a priority change measurement of 0.326368. All methods were able to obtain solutions with 0 allocation changes and fewer priority changes than this.

The sequential method found a solution with no allocation changes in 3 of 10 trials. Of these the best priority change score was 0.207173. The average total change score, including trials which used allocation changes, was 0.2768.

The simultaneous method failed to find a schedulable solution in the first step for all occasions when mode 2 was weighted more highly. It found solutions with no allocation changes for 4 of the remaining 5 trials. Of these, the smallest priority change measurement was 0.175291. The average total change score for the successful trials was 0.2214.

The parallel method failed to find a schedulable solution for one of the modes on two occasions. For the successful solutions, 6 of the 8 cases found configurations with no allocation changes. The smallest priority change score was 0.131818. The average total change score for the successful trials was 0.1949.

6.3 Computation and Time

The number of search moves used and time required varies between problems. The major factors in how much time it takes to find a solution are the length of each cost function evaluation and the number of moves required. The former depends on the complexity of the task attributes and the size of the problem. A large number of factors affect the latter including the task utilisation, size of problem, topology of hardware platform, etc. A timed trial was run on the Deps example for a million search moves which is easily sufficient to solve a problem of its complexity. The trial was run on an AMD Athlon 64 Dual Core 4400+ and took 15 minutes to complete.

The generation of baselines for both the sequential and parallel methods were able to take advantage of the dual core processor and so could find both baselines in 15 minutes. At this stage the parallel method has also attempted to minimise change. However, the sequential method is likely to find a feasible solution in fewer than one million moves.

For the simultaneous method, each baseline takes twice as much computation time since at each search step both modes are evaluated. This was tested and did indeed take 30 minutes. It is accepted that the software could be redesigned to run the cost function evaluation for each system in a separate thread. On a dual core processor this would take approximately 15 minutes (ignoring overheads) but only one baseline would have been produced. This makes the simultaneous method the most expensive unless it finds a zero change solution in its first run.

7 Extensions to several modes

Most multi-moded systems have more than two modes. For these methods to be more widely applicable, their suitability for systems with more than two modes must be considered and evaluated.

A system with N modes has $N(N-1)/2$ distinct pairs of modes. Therefore, the number of mode transitions that must be dealt with increases quadratically with the number of modes. In most systems, only a subset of these transitions will take place.

To extend the first method, sequences of modes must be considered. A mode is chosen from which to create a baseline. The configuration for the next mode in the sequence is then generated by minimising differences from this baseline. Subsequent solutions use existing configurations as baselines for modes from which a transition might take place. Since the quality of the solution has been shown to be dependent upon the chosen sequence of modes, ideally all $N!$ sequences should be attempted. This soon becomes infeasible for moderately complex systems where each search may take significant time. When combined with other deficiencies in the sequential method, it does not seem suitable for handling systems with several modes.

The simultaneous method has similar problems to the sequential one in that it is necessary to select a mode to prioritise and then use this as a baseline to move to other modes. The method is again dependent upon the sequence in which modes are given the highest weighting. Each search in the sequence requires N times more computation power than that of the sequential method.

Where this method has been shown to be successful is in the specific case where there exists a single configuration which is feasible for all modes. When this situation exists, it can be found by weighting all modes equally and this method is more reliable at generating this optimal solution than either of the other two methods. Therefore, it is recommended that this method be used first to see if it can find such a solution.

The parallel method has the advantage that it can attempt to minimise all transitions at the same time without having to consider a sequence of modes. It was stated earlier that this method can benefit from using each of the configurations created from the parallel search as a baselines and performing a sequence of searches to find configurations for other modes. With an N mode system, this once again becomes infeasible.

The initial parallel search finds solutions which are both feasible for all modes and minimises changes. Our aim is to show that sufficiently high quality solutions can be found using a single parallel search. The number of searches running increases linearly with the number of modes. However, it is possible that each search will need to be run for more iterations as the number of modes and problem size increase.

7.1 3 Mode Example

The first multi-mode experiment used 3 randomly generated modes. Each mode was created by taking a random delta from an original randomly generated system. The delta involved exchanging approximately 20% of the tasks for new tasks and increasing task execution times between 10% and 50%. Messages could also be added. The numbers of tasks and messages in each mode is outlined in table 8. The hardware platform had 5 processors and the task attributes resulted in an average processor utilisation of 78%.

Mode	Tasks	Messages	Avg Utilisation
M_1	25	20	65.22
M_2	29	23	70.5
M_3	28	17	71.7

Table 8. 3 mode example

	$M_1 \leftrightarrow M_2$	$M_1 \leftrightarrow M_3$	$M_2 \leftrightarrow M_3$	Result
Low	0	0	0	0
High	1.25	1.0	1.5446	3.7946
Exp 1	**0.0625**	**0.0667**	**0**	0.1292
Exp 2	**0.0625**	0.6667	**0**	0.0625
Exp 3	0.6875	**0**	**0.1429**	0.1429

Table 9. 3 mode transition costs

The first stage of the evaluation was to apply the simultaneous method to attempt to discover a single configuration for all modes. All modes were given equal weighting. The configuration produced was not a valid solution for mode M_2 but was for M_1 and M_3. This means it is possible to transition between modes M_1 and M_3 with no changes. However, whether this configuration is useful, will depend on the transitions required and other modes in the system.

Three experiments were conducted, each of which assumed different mode transitions were required by the system. The first, *Exp 1*, allowed any mode to move to any other mode. The second experiment, *Exp 2* tried to reduce the size of changes for the sequence $M_1 \leftrightarrow M_2 \leftrightarrow M_3$. The final experiment, *Exp 3* looked at the sequence $M_2 \leftrightarrow M_3 \leftrightarrow M_1$. All experiments used the parallel method running all 3 searches together. The experiments differed in the information exchanged between the parallel searches. For a particular mode, the solution only tried to converge towards those for modes which were relevant for the required mode transitions. The results of the experiments are given in table 9. Each value is the sum of the task and message allocation change metrics and so has a maximum value of 2. To get an estimate of a lower bound, test were performed where only the two modes in the transition were considered. These values are in the row labelled *Low*. For this particular case, all pairs could be handled with no changes. A solution for each mode was also found completely independently of other modes. The differences between mode configurations based on these solutions are in the row labelled *High*. The Low and High values can be used to judge the quality of solutions found in the experiments.

The experiments performed reasonably as expected with the change for transitions relevant to an experiment being smaller than the irrelevant ones. However, it can be seen that the results from the first experiment are actually as good as or better than the values for the other two experiments.

Mode	Tasks	Messages	Avg Utilisation
M_1	27	16	85.17
M_2	28	10	87.95
M_3	24	17	76.8
M_4	30	12	88.38

Table 10. 4 mode example

	$M_1 \leftrightarrow M_2$	$M_1 \leftrightarrow M_3$	$M_1 \leftrightarrow M_4$	
Low	0	0	0.0476	
High	1.333	1.2889	1.2321	
Exp 1	**0.1667**	**0.1667**	**0.1905**	
Exp 2	**0.2778**	0.5889	0.7738	
Exp 3	0.6111	0.2667	**0.0952**	
	$M_2 \leftrightarrow M_3$	$M_2 \leftrightarrow M_4$	$M_3 \leftrightarrow M_4$	Result
Low	0	0	0.04761	0.0952
High	1.0417	1.0191	1.0952	7.01
Exp 1	**0.0625**	**0.1905**	**0.2381**	1.0149
Exp 2	**0.0625**	0.3333	**0.1905**	0.5308
Exp 3	**0.0625**	0.5238	**0.3810**	0.5387

Table 11. 4 mode transition costs

7.2 4 Mode Example

A 4 mode example is given in table 10. Each mode was similar in size in terms of tasks and messages as the 3 mode example. The hardware platform had 7 processors. The average utilisation per processor was higher than the 3 mode example despite the extra processors.

The results from the 4 mode experiment are given in table 11. Once again, separate pairs of modes were used to get a lower bound approximation and solutions with no attempt to minimise change gave high values for comparison. The behaviour for Exp 2 and Exp 3 were as expected. Exp 1 still outperformed the other experiments over their highlighted transitions though not now on each individual transition.

In both the 3 and 4 mode case, a parallel search for all modes performed well in comparison to searches only considering particular transitions. There is obviously some benefit in trying to minimise changes between transitions even if they won't occur in the system. However as the number of modes and differences between modes increases, trying to minimise all transitions is more difficult. It may be the case that all transitions should be considered but more importance should be given to those which are actually relevant to the system.

8 Conclusion

Three methods have been given for finding feasible solutions to task allocation and scheduling problems whilst minimising configuration differences between modes. These were a method which looks at each mode in a sequence, a method which tries to find a single configuration for all modes and a method which uses parallel communicating searches with a search dedicated to each mode. All perform better than finding independent solutions for each mode.

It is recommended that the simultaneously feasible configuration method be tried first. If this fails, then the parallel method proposed gives a good balance between effectiveness and scalability. It is able to find optimal or close to optimal solutions in a reasonable amount of time. It scales better than the other methods as the number of modes increases.

References

[1] I. Bate and P. Emberson. Incorporating scenarios and heuristics to improve flexibility in real-time embedded systems. In *Proceedings of 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 221–230, 2006.

[2] J. Beck and D. Siewiorek. Simulated annealing applied to multicomputer task allocation and processor specification. In *Proceedings of the 8th IEEE Symposium on Parallel and Distributed Processing*, pages 232–239, 1996.

[3] G. Fohler. Realizing changes of operational modes with pre run-time scheduled hard real-time systems. In *Proceedings of Second International Workshop on Responsive Computer Systems*, 1992.

[4] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[5] J. Leung and M. Merrill. A note on preemptive scheduling of periodic, real-time tasks. *Information Processing Letters*, 11(3), November 1980.

[6] D. Milojicic, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou. Process migration. *ACM Computing Surveys*, 32(3):241–299, 2000.

[7] J. Palencia and M. G. Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 26–37, 1998.

[8] P. Pedro. *Scheduling of Mode Changes in Flexible Real-Time Distributed Systems*. PhD thesis, Department of Computer Scheduling, University of York, 1999.

[9] D.-T. Peng, K. Shin, and T. Abdelzaher. Assignment and scheduling communicating periodic tasks in distributed real-time systems. *Software Engineering*, 23(12):745–758, 1997.

[10] K. Ramamritham. Allocation and scheduling of precedence-related periodic tasks. *IEEE Transactions on Parallel and Distributed Systems*, 6(4):412–420, 1995.

[11] J. Real and A. Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Systems*, 26(2):161–197, 2004.

[12] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham. Mode change protocols for priority-driven preemptive scheduling. Technical Report UM-CS-1989-060, University of Massachusetts, 1989.

[13] Y. Shin, D. Kim, and K. Choi. Schedulability-driven performance analysis of multiple mode embedded real-time systems. In *Design Automation Conference*, pages 495–500, 2000.

[14] K. Tindell, A. Burns, and A. Wellings. Allocating hard real-time tasks: An NP-hard problem made easy. *Real-Time Systems*, 4(2):145–165, 1992.

[15] K. Tindell, A. Burns, and A. Wellings. Mode changes in priority pre-emptively scheduled systems. In *IEEE Real-Time Systems Symposium*, pages 100–109, 1992.