Spare Capacity Distribution Using Exact Response-Time Analysis

Attila Zabos, Robert I. Davis, Alan Burns
Department of Computer Science
University of York, UK

Michael González Harbour
Computers and Real-Time Group
Universidad de Cantabria, Spain

Abstract

Real-time systems designed for use in dynamic open environments allow applications to enter and leave the system during runtime. This leads to changing runtime scenarios where the load and the spare capacity of hardware resources is influenced by the resource demand of running applications. Flexible real-time application components, with variable temporal parameters, can adapt their timing behaviour to these changing runtime scenarios and improve both, their Quality of Service (QoS) and the utilisation of system resources. In these open systems application components are often designed independently of each other, introducing the need for system management of resources at runtime. This management requires a mechanism to distribute the available system resources among the running application components, in a way that maximises resource usage and increases QoS with respect to their importance and temporal limits. This paper introduces a runtime algorithm for the distribution of spare capacity in flexible real-time systems. The efficiency of the presented algorithm is evaluated by empirical tests and performance measurements on embedded hardware.

1. Introduction

In embedded real-time systems being developed today it is common to find requirements for flexibility and support for dynamic behaviour that are key driving factors in the design of their architectures and scheduling methods. Manufacturers of these embedded and resource constrained systems are faced with the difficulty of providing guarantees on the real-time behaviour of their applications while at the same time handling flexibility and dynamic changes to the applications being executed. Traditional real-time scheduling focuses on worst-case behaviour and using it for static configurations implies that a large amount of capacity, that is only used occasionally, is statically allocated in order to manage dynamic changes in application demand. This conflicts with the common requirement to be able to use the maximum possible amount of the available resources.

In this context of requirements for flexibility and support for dynamic behaviour it is possible to design applications that adapt themselves to the available resources, trading the quality of the response with the usage of resources. In a system developed with this adaptivity in mind it is possible to maximise resource usage while trying to provide the best possible QoS.

The complexity of modern embedded systems is also driving the need to independently develop applications or application components that may join and leave the system during runtime. The available resources should be dynamically adapted to these changing situations. In most platforms these dynamic changes may be frequent, but not as fast as the regular application periods. We may have new applications that stay in the system for seconds or minutes, while their own internal periods may be in the milliseconds range.

2. Motivation

Flexible applications come in many different forms. For instance, multimedia systems need to process different kinds of video and audio streams that have highly variable computation times but require real-time processing and rendering. It is common that classic industrial control applications, such as a robot control, get mixed together with multimedia activities when the process in which the robot is working requires image capture and analysis, or remote video monitoring. Mobile phones and other embedded devices are turning themselves into devices with multiple capabilities, including audio and video processing, GPS positioning, Internet navigation, and more, in addition to their original radio link capabilities. In these systems it is common that the user starts or downloads new applications that must run together with the previous ones in an environment with limited resources; and many of these applications have real-time constraints.

Not all the applications running in the system are capable of adapting or adapting equally to the available resources. Most applications will require a minimum amount of resources to produce results of the minimum acceptable quality. Then, some applications may be able to use additional resources, while others won't. Of those that may adapt to the available resources the level of adaptation may be different. For instance, a video player may upgrade itself to a higher frame rate if more resources are available, changing the rate in a continuous way, up to a maximum level after which there is no perceived increase in the quality of the output. These type of applications are referred to as *continuous*. A con-

trol algorithm on the other hand may have two versions: a fast one with a low quality output and one requiring more computation time and providing a high quality output. In this case the system should allocate resources to run either one version or the other. Applications with this type of behaviour are referred to as *discrete*. In general we find applications that can operate and generate valid results at different frequencies (i.e. having a variable period), and/or handle different processing time assignments (i.e. having a variable execution time).

The FRESCOR[1] EU project is developing a middleware layer, that is placed on top of a real-time operating system, with the capability of running multiple application components and scheduling the resources that they need to run. Each application component describes the resource requirements through one or more *contracts*. These contracts specify the minimum resource requirements and the way in which the component is able to use any additional available resources. The contract is negotiated with the system in order to verify that the minimum resources required can be granted to the application component. Once a contract has been accepted by the system a *virtual resource (VR)* (which has similar properties to servers [10]) is allocated to the application component. This VR represents a resource reservation, and manages the consumption of the resource by the application component to ensure that it can always get its allocated budget, and that it never exceeds it, so that other application components can also run in the system with full temporal protection. On the other hand, when application components complete processing and terminate, their VRs are destroyed. In this situation the utilisation of the processing hardware resource decreases, causing an increase of the spare capacity. As a consequence this spare capacity can be distributed to all of the currently active VRs by adapting their parameters.

Since the decision making process for the selection of VR parameters is an online task, there will be a trade-off between the selection of optimal VR parameters and the maximal affordable processing time for this task.

These mixed application and system requirements give us the motivation for an online anytime algorithm that is capable of distributing the spare capacity available in a given system resource. This resource is commonly a processor running tasks, but may also be a network transmitting message streams. The algorithm presented in this paper is designed for fixed priorities, although we believe that it could be easily extended to other scheduling policies. We call this algorithm SCD, for *spare capacity distribution*.

The SCD algorithm is based on the idea of maximising the utilisation of the processing hardware resource, whilst optimising the QoS of the applications as dictated by their importance and weight. The two attributes, importance and weight, specified in each contract allow the system integrator to control the spare capacity distribution among active applications in the system by defining their significance to the system and relative to each other. The SCD algorithm maximises the hardware resource utilisation by modifying the VRs' temporal attributes during runtime (i.e., budget, period, deadline and as a consequence their priorities as well), under the condition that their temporal requirements are not violated. The search for a feasible spare capacity distribution is an incremental process and it is based on bisection that provides the foundation for a simple and efficient implementation of the presented algorithm in a runtime framework. Furthermore, the algorithm uses response time analysis to test for schedulability, and since this method is known to be exact no schedulability losses are incurred by the test itself. The transition to a system state where the new temporal values and priority ordering are used, is performed at a feasible time instant (e.g., at an *idle instant* [21], or according to the idle instant optimised protocol [11]). The protocol for the change from one set of applications to another one, is currently a subject of extensive research. The purpose of this paper is to present and evaluate the SCD algorithm.

3. Related work

In real-time systems, servers are often used as a resource reservation mechanism to accomplish temporal partitioning among running application components. They provide a budgeting mechanism, which prevents malicious applications from affecting the operation of other applications in the system. Although server concepts [19, 20, 16] have been introduced to improve the response time of aperiodic tasks, their application has been adapted to provide temporal partitioning among tasks [1].

The composition of independently developed applications has been considered by Deng et al. [9]. They defined a scheme for a two-level hierarchically scheduled open system. Their work was based on dynamic scheduling. Kuo and Li [13] later adapted Deng's approach [9] to a fixed-priority operating system scheduler.

The FIRST[2] project addressed the need for a scheduling framework that could handle applications with varying processing demand [2]. The algorithm to adapt the server parameters used utilisation-based schedulability tests which are not exact, thus causing a lower resource usage in the system.

Utilised in this paper are the improvements to the exact schedulability test for fixed-priority tasks that were presented in [8] by Davis et al. and in [7] by Davis and Burns. In our case, the exact schedulability test is applied to VRs and determines whether the full amount of each VR's budget can be utilised before its deadline by the associated application tasks.

The server attributes, *importance* and *weight*, that are related to the spare capacity distribution, were intro-

[1] Framework for Real-time Embedded Systems based on COntRacts [10]

[2] Flexible Integrated Real-Time Systems Technology

duced in [2]. They were further adopted for VRs in [10]. These attributes allow the applications to influence the outcome of the spare capacity distribution.

Rosu et al. [18] described an adaptive resource allocation mechanism for distributed real-time systems. The expected application resource needs are specified by configurations. The choice of the appropriate configuration and the resulting resource allocations depends on environmental states, availability of resources in the system and the achievable system performance. The resource allocation is carried out as a response to events in the environment and changes in the processing demand of a complex distributed application.

Resource adaptive soft real-time systems were considered by Lin et al. [15]. Here, the Rate-Based Earliest Deadline (RBED) scheduler uses a heuristic algorithm to increase the overall benefit by adjusting the QoS level of the adaptive soft real-time tasks. Since the resource demand varies with the QoS levels, the processing of the adaptive tasks is adjusted so that they can be accommodated on the available resources.

An elastic tasking model has been defined by Buttazzo et al. [5, 6], where the task periods can be adjusted in order to adapt to different load conditions.

Rajkumar et al. [17] introduced a QoS based resource allocation model. Resource allocation to applications is made in terms of resource utilisation, but it is the application's responsibility to choose the appropriate budget and period. The algorithm that determines the resource allocation, requires QoS functions representing resource dependent changes of the application's contribution to the system value. However, the definition of such a function is not always straightforward.

Almeida et al. [3] presented an approach to adapt during runtime the temporal parameters of flexible periodic tasks. Each task's execution time and period is expressed as an *n*-tuple vector for *n* different QoS levels. From the set of all possible combinations of task parameters, a set of schedulable configurations is deduced by an offline method. This set is used by the online QoS manager to adapt the task parameters.

The SCD algorithm presented in this paper addresses the following issues of previous work: The resource allocation model in [18] is designed for high performance distributed systems, rather than for embedded real-time systems. Compared to the resource allocation model in [15], we focus on fixed-priority scheduled systems. The elastic task model [5, 6] addresses resource adaptation by changing task periods but not their allocated processing times. In contrast to the scheduling framework in [2], the SCD algorithm can be used as an anytime algorithm, and terminate after a maximum execution time providing non-optimal but schedulable VR parameters. Furthermore, loss of distributable spare capacity due to the schedulability test of VRs is prevented by using an exact test. The adaptation method presented in [3] conflicts with the notion of open systems due it assumption of static set of tasks that are known before runtime, and is therefore not well applicable to open systems.

4. System model

It is assumed that a set of flexible application components $\{A_1, A_2, \cdots, A_m\}$ are mapped to a set of virtual resources $\Gamma = \{S_1, S_2, \cdots, S_n\}$, with $m \leq n$.

The tasks of each flexible application component are mapped to and are executed by one or more VRs. In the former case the application's taskset is mapped to one VR, where in the latter case the taskset is divided into subsets and each subset is mapped to a VR. Unless a one-to-one mapping of task to VR is used, a hierarchical scheduler can be utilised to determine the order of execution of tasks on the same VR. The presence of one or more tasks on a single VR does not affect the spare capacity distribution, as it is done solely for VRs according to their requirements specified in their respective contracts.

The tasks of one application component are not mixed with tasks of other components in the same VR, in order to ensure the temporal protection among them.

Each *virtual resource* S_i is characterised by its processing *budget* C_i, replenishment *period* T_i, *deadline* D_i, *importance* I_i and *weight* W_i.

Each virtual resource S_i is either defined as a continuous or discrete VR [10] depending on the domain of its temporal parameters:

- For continuous VRs, the operational ranges of period and budget are defined by a lower and upper bound, $[T_i^{min}, T_i^{max}]$ and $[C_i^{min}, C_i^{max}]$, respectively. The actual value assigned to a VR's temporal attribute can take any value from within corresponding operational ranges. The budget and period of continuous VRs are independent of each other and therefore can be adjusted independently.
- For discrete VRs a finite set of (C_j, T_j, D_j) triples are defined. Only values from this set of (C_j, T_j, D_j) triples can be assigned to discrete VRs' temporal attributes. This definition implies that temporal attribute values are linked to each other.

The budget C_i denotes the maximal processing time that can be consumed by a VR before it is suspended. The budget is replenished to its full amount after every period T_i.

The deadline D_i of a VR specifies the relative time from the point when it has been released until it has to complete the supply of its budget to the associated application component. The deadline D_i of a virtual resource S_i can be defined to be:

1. equal to its period T_i for a continuous VR,
2. equal to D_j from the selected (C_j, T_j, D_j) triple for a discrete VR or,
3. a constant value for both types of VRs.

Additionally, it is assumed that the deadline of a VR is always less than or equal to its period, $D_i \leq T_i$.

The priority P_i of a VR S_i is assigned to it using a

fixed-priority assignment policy. The priority of VRs may change during the spare capacity distribution, but it remains fixed during the steady operational state of the system. If a VR's temporal attribute, which is used by the priority assignment policy, is modified during the execution of the SCD algorithm, then the priority assignment has to be updated. Nevertheless, the constraint $D_i \leq T_i$ is not violated by the modification of the period.

For a VR S_i two of its attributes, *importance level* I_i and *weight* W_i, are used to influence the outcome of the SCD algorithm. The precedence, in which the spare capacity is assigned to VRs is determined by the importance level I_i of the VRs. For the spare capacity distribution, VRs with the same importance level are logically combined into groups. A group G_l is the set of all VRs with the same importance level l (see Equation1).

$$G_l = \{S_i \in \Gamma | I_i = l\} \tag{1}$$

Each major iteration of the SCD algorithm is limited to a group G_l of VRs. Usually several major iterations of the SCD algorithm are required until a solution is found for the given set Γ of VRs.

The *weight* attribute W_i influences the fraction of the spare capacity that a VR will get. The fair share value [12] denoted as H_i is used as a factor to determine the fraction of additional capacity that a virtual resource S_i will get when a certain amount of spare capacity is distributed at the currently examined importance level I_C (see Equation 2). In this equation, I_j denotes S_j's importance level.

$$H_i = \frac{W_i}{\sum\limits_{\forall j: I_j = I_C} W_j} \tag{2}$$

The usage of shared resources and the consequent blocking factors have no direct impact on the SCD algorithm itself. Therefore, there are neither assumptions nor restrictions on the usage of shared resources, since the corresponding blocking factors affect only the schedulability test.

The *utilisation* caused by a virtual resource S_i on a processor is the amount of assigned budget C_i per replenishment period T_i. The processor's capacity that is unused by VRs is denoted as *spare capacity*.

The processor's utilisation is calculated as the sum of all VRs' utilisation. The largest utilisation of the processor, determined by the SCD algorithm, at which the set of VRs in the system becomes unschedulable is referred to as the *breakdown utilisation*. The breakdown utilisation mainly depends on the VR types in the system. With only continuous VRs it is likely that close to 100% processor utilisation is achieved since the utilisation assigned to these VRs can be gradually increased. Whereas with discrete VRs the largest processor utilisation is likely to be significantly less than 100% due to the limited number of VR budget and period values.

5. Spare capacity distribution algorithm

5.1. SCD algorithm characteristics

The intention of the SCD algorithm is to allocate as much as possible of the processor's spare capacity U_s to VRs in the system. Of course, after the application of the SCD algorithm, the set of VRs with their new utilisation values has to be schedulable.

The utilisation probe U_p is the amount of spare capacity that can be distributed at once among the VRs at the processed importance level.

Since the presented algorithm is a search based approach, feasibility of various probe values need to be tested. An efficient approach to find a feasible probe value $U_p \in \{0, 0 + \delta U_p, \ldots, U_s - \delta U_p, U_s\}$ is provided by bisecting the interval $[0, U_s]$ and applying the binary search algorithm to it. The binary search algorithm is fast, has minimal memory requirements and the required processing resources are also low.

For a given set of VRs, the maximal distributable spare capacity can be found within $1 + \lceil log_2 N \rceil$ iterations, where N is the number of potential values for U_p. Given the granularity δU_p of the interval $[0, U_s]$, the number of potential values for U_p can be calculated as $N = \frac{U_s}{\delta U_p}$. For example, a typical value of 1% for δU_p and the possible maximum value for U_s of 100%, limits the number of potential values N for U_p to 100. By applying the binary search on the 100 possible values, a feasible U_p can be determined by checking the schedulability of $1 + log_2 100 = 8$ different spare capacity distribution scenarios.

A virtual resource S_i is defined to be available/active for spare capacity distribution if it is able to increase its utilisation due to the following conditions:

- During the last iteration of the spare capacity distribution, S_i did not render the system unschedulable,
- S_i has not reached its predefined maximal utilisation and can therefore utilise a higher spare capacity allocation.

In order to determine the optimality of the SCD algorithm the following assumption is made, driven by the FRESCOR project: The spare capacity supplied at higher importance levels is considered infinitely more valuable than at lower importance levels. This implies that different importance levels are incomparable. The SCD algorithm starts with the spare capacity distribution at the highest importance level. Spare capacity is supplied by decreasing the periods and increasing budget of VRs at this importance level. Only after all possible spare capacity has been allocated at the highest importance level, does the algorithm considers the next highest level and so on. Hence, under the assumption that importance levels are incomparable, the SCD algorithm provides the optimal spare capacity distribution.

The schedulability test used by the SCD algorithm is an exact test for fixed-priority scheduled uniprocessor real-time systems [8]. Before the SCD algorithm is ap-

plied to the VRs in the system, the schedulability using their minimal timing requirements is ensured. Changes to the set of the VRs in the system are only permitted if the changed set remains schedulable using the minimal temporal requirements of the VRs.

5.2. Description of the SCD algorithm

As input, the SCD algorithm requires a priority ordered list of VRs along with their temporal attributes. This *priority ordered list (*Π*)* contains VRs that want to benefit from the additional assignment of spare capacity. Π is also used as the output list.

The SCD algorithm starts with the VRs' temporal parameters set to their minimum timing requirements. The minimum timing requirement is either the minimum budget C_i^{min} and largest period T_i^{max} in the case of a continuous VR, or (C_j, T_j, D_j) triple with the smallest utilisation in the case of a discrete VR. A soon as the SCD algorithm finds an intermediate or final solution, the new temporal values of the VRs are stored in Π.

With the SCD algorithm, at each importance level the search for the largest feasible U_p is carried out in order to determine a feasible spare capacity distribution (see Algorithm 1).

InOut: Π: Priority ordered list of VRs
foreach *importance level l (in decreasing order)* **do**
  Determine H_i for all S_i^l that are active;
  while *not all possible* U_p *values checked* **do**
    Calculate ΔU_i (see Equation 3) and increase the current utilisation of S_i^l to U_i^{new} by ΔU_i;
    Determine S_i^l new parameters using Algorithm 2;
    Determine new priority ordering for the schedulability test;
    if *all VRs are schedulable* **then**
      Store new priority ordering and temporal values of all active S_i^l in Π;

Algorithm 1: Spare capacity distribution

Each major iteration of the SCD algorithm is limited to VRs at a particular importance level, starting at the highest level.

The steps of the SCD algorithm that are executed at every importance level are as follows:

1. Calculate the fair share values.
   The fair share value H_i of every active virtual resource S_i at the currently processed importance level is calculated using Equation 2. This value is used to determine the fraction of capacity that will be assigned to the active VRs.
2. Search for a feasible spare capacity distribution.
   The algorithm considers only VRs that are active (i.e. capable of accepting more than their current utilisation) at the currently examined importance level. Using binary search (represented by the *while*-loop condition in Algorithm 1), the distributable spare capacity U_p is narrowed down to the largest value at which the system is schedulable, but where an additional amount (δU_p) of spare capacity assignment (i.e. $U_p + \delta U_p$) would cause an infeasible schedule.
   For each utilisation probe U_p, the temporal parameters of active VRs at the currently examined importance level are recalculated using Algorithm 2, where U_i^{new} denotes the increased utilisation of S_i by ΔU_i.
   $$\Delta U_i = U_p \cdot H_i \tag{3}$$
3. Increase active VRs' utilisation.
   If a schedulable spare capacity distribution has been found, the new VR temporal parameters are stored in the output list Π regardless of whether these values are intermediate or final.

One possible approach to determine the budget and period of a continuous VR S_i (with an increased utilisation equal to U_i^{new}) is presented in Algorithm 2. If possible, the VR's smallest period T_i^{min} is used and its budget is calculated accordingly. If it is not feasible to use its smallest period, then the minimum budget C_i^{min} is fixed first and the period is calculated accordingly. Under both circumstances, the calculated values are restricted to the specified interval for the budget and the period, respectively.

if $(C_i^{min}/T_i^{min}) > U_i^{new}$ **then**
  $C_i = C_i^{min}$;
  $T_i = \min\big((C_i^{min}/U_i^{new}),\ T^{max}\big)$;
else
  $T_i = T_i^{min}$;
  $C_i = \min\big((T_i^{min} \cdot U_i^{new}),\ C^{max}\big)$;

Algorithm 2: Budget and period calculation

For a discrete VR, the utilisation values of its different discrete (C_j, T_j, D_j) triples are calculated. Then the (C_j, T_j, D_j) triple with the biggest utilisation value, which is less than or equal to U_i^{new}, is selected.

5.3. Priority assignment

In this paper, the deadline-monotonic priority assignment [14], as provided by the FRESCOR framework, was used for the empirical evaluation. The implementation of the SCD algorithm considers two different VR configurations for the deadline:

1. Virtual resource's deadline is equal to its period,
2. Virtual resource's deadline is constant.

In the case that a VR's deadline is configured to be equal to its period, whenever the period is changed during the spare capacity distribution its deadline is adjusted accordingly and priority reordering is carried out.

Furthermore, the schedulability test of the VRs is carried out in decreasing order of their priorities.

6. Empirical evaluation

6.1. Test data generation

To evaluate the performance of the SCD algorithm, VR sets were generated where the variation of particular isolated test-case parameters was examined. For each

of these VR sets 100000 different configurations were created. Each configuration consists of the predefined number of VRs (i.e. 5, 10, 15, ..., 50) for which randomly generated VR parameters (i.e. budget, period and deadline) were created.

The approach, presented by Bini and Buttazzo [4], to create tasks parameters for a given maximal utilisation (subsequently referred to as the *Initial Target Utilisation (ITU)*), was used in this work to generate the random VR parameters.

The ITU was chosen so that the performance of the spare capacity distribution could be observed in different scenarios where more or less spare capacity was available. The chosen ITUs were 30%, 50% and 80%. Before the VR sets were processed by the SCD algorithm, it was ensured that the generated sets were schedulable at the chosen ITU. Test-cases with initially unschedulable VR sets were not considered in the measurement and they were replaced with a new and schedulable VR set.

The period and budget of the VR is derived from its randomly generated utilisation value. First, the VR's period is chosen according to a uniform random distribution from a randomly selected period magnitude range (i.e. $[10^3, 10^4]$, $[10^4, 10^5]$, $[10^5, 10^6]$ or $[10^6, 10^7]$). Given the VR's utilisation and period, the calculation of its budget is straight forward.

To take advantage of the flexibility of VRs, a definition for the upper and the lower bound of their utilisation ranges is required. For each VR the initially generated random utilisation values are considered in the tests as their lower utilisation bounds. This lower utilisation bound is used to derive the VR's minimum budget and maximum period. The minimum budget is then multiplied and the maximum period is divided by a factor in order to determine the VR parameters (i.e. maximum budget and minimum period) for its upper utilisation bound. A factor of 2.0 for the 30% ITU, and a factor of 1.5 for 50% and 80% ITU was used.

For discrete VRs, additional to their lower and upper utilisation bounds, a random number of intermediate utilisation values were generated. The number of intermediate values was in the range of 1 to 3, where 5 is the maximum number of discrete temporal attributes defined by the FRESCOR framework.

6.2. Experiment evaluation

This section evaluates the data, which was collected from various measurements by varying different parameters as indicated in Section 6.1. The diagrams show the progress of the spare capacity distribution as a function of the number of ceiling operations needed by the schedulability analysis, a useful proxy for algorithm execution time [8]. This section uses the number of ceiling operations to give insight into the complexity of the algorithm. Section 7.2 extends this information by mapping it onto absolute time. In the following, the term *number of ceiling operations* will be referred to as *number of iterations*.

The presented experiments contain a mixed set of VRs (both continuous and discrete). The only exception is the experiment in which the impact of the different temporal attribute types (i.e. discrete or continuous) on the SCD algorithm is examined.

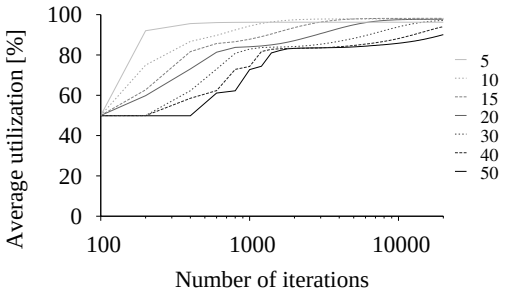
For the sake of clarity, the diagram types used to support the empirical evaluation are introduced in Figure 1. Of main interest are the following properties of the SCD algorithm: how fast can the spare capacity be distributed and the number of iterations required by the algorithm to terminate. The average increase of processor's utilisation as the SCD algorithm progresses, is depicted in Figure 1a. The percentage of test-cases terminated by a certain number of iterations is depicted in Figure 1b. This figure can also be interpreted as the probability that the SCD algorithm will terminate within a given number of iterations for a given number of VRs.

For all experiments the SCD algorithm was executed until it terminated itself. The processor's utilisation achieved in this way is the highest schedulable utilisation of the active VRs in the system.

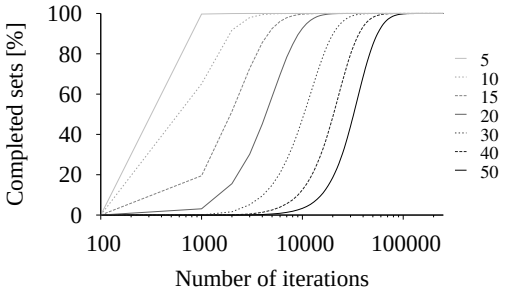
In Figure 1 the effect of the number of VRs on the SCD algorithm is examined. Figure 1a shows that the rate of the average utilisation increase from an ITU of 50% slows up when the number of VRs in the system is increased. The number of iterations required to terminate with a particular probability also increases with the number of VRs (see Figure 1b). The diagrams indicate that more iterations are required to find a feasible spare capacity distribution as the number of VRs in the system increases. This comes about due to the increased number of schedulability tests that are required for each VR utilisation level.

In dynamic real-time systems the processor's utilisation at which application components are submitted into the system cannot be predicted in advance. Therefore, the impact of the initial processor's utilisation on the progress of the SCD algorithm was also considered (see Figure 2). In Figure 2a it can be observed that the ITU has an effect on the processor's utilisation increase only at the beginning of the algorithm. In the long term (i.e. above 2000 iterations) the ITU becomes irrelevant and the utilisation increase is dominated by the number of VRs in the system. The probability for the termination of the SCD algorithm is influenced from the beginning by the number of VRs and the ITU has only a negligible effect (see Figure 2b). The effect of the ITU is small since the number of schedulability tests that are carried out during the runtime of the SCD algorithm stay nearly identical for various ITUs.

The VR's temporal attributes can be either of continuous or discrete type. Hence, there can be three different VR sets in the system. Only continuous, only discrete or a mixed set of VRs. Our experiments show that the temporal attribute type has only a minor effect on the increase of the processor's utilisation and the runtime of

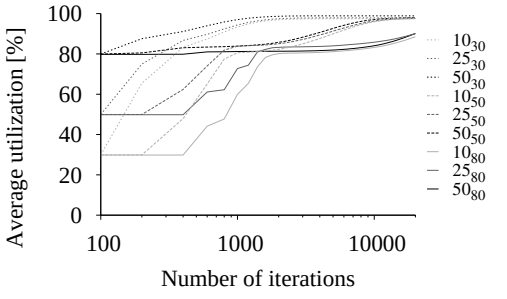


(a) Average utilisation

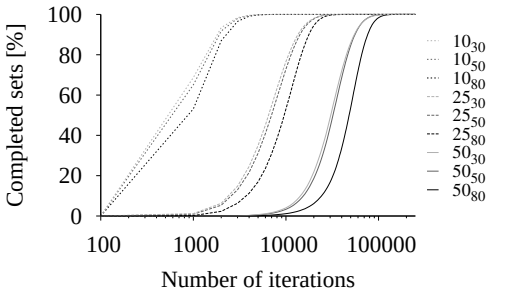


(b) SCD algorithm termination rate

Figure 1. Number of VRs: 5, 10, ..., 50



(a) Average utilisation

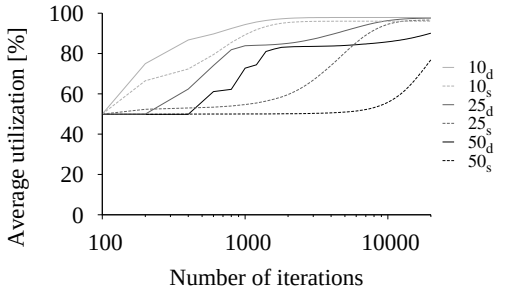


(b) SCD algorithm termination rate

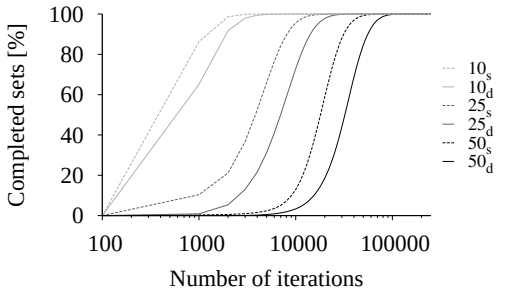
Figure 2. ITU: 30%, 50% and 80%

the SCD algorithm. For space reasons, these results are not included in this paper but are available in a technical report [22].

The VR assignment to importance levels has been analysed in two scenarios (see Figure 3). In one scenario, the VRs have been randomly distributed among the available importance levels. In the other scenario, they have been assigned to a single importance level. Figure 3a shows that the use of a single importance level had a negative effect on the processor's utilisation increases. These are much slower although in long term, the processor's utilisation values for both scenarios converge towards each other. On the other hand, the number of iterations required for the termination of the algorithm decreased if a single importance level was used (see Figure 3b). In this case, performing the schedulability test for the empty importance levels was avoided, which led to the reduction in iterations.



(a) Average utilisation



(b) SCD algorithm termination rate

Figure 3. Importance level allocation

The measured data, presented in this section, reveal that the performance of the SCD algorithm is mainly influenced by the number of VRs in the system.

7. Performance evaluation

This section provides information about the embedded hardware that was used for the performance measurements and the absolute values for the execution times obtained. Additionally, using a pragmatic approach, a linear upper bound equation is derived for the execution time of the SCD algorithm. This allows us to determine the required budget for the SCD algorithm, in order to provide a complete or partial solution for the spare capacity distribution of the system.

7.1. Test environment

The empirical evaluation, which was presented in Section 6, is extended by performance measurements on an embedded platform. The intention of this task is

to determine which parameters influence the SCD algorithm's execution time.

In order to exclude operating system and other undesirable overhead, an embedded system was configure in a way that only the spare capacity distribution was executed. This provided the facility to obtain absolute values for the execution time of the SCD algorithm.

The embedded platform consisted of an MPC555 microcontroller running at 40 MHz system clock.

To create the target executable file, a development environment comprising the GNU C compiler and RapiTime version 1.2, a tool for worst-case execution time analysis, was used. The executable files were optimised by the compiler using the option *-O2*. To avoid falsification of the measurements by intermediate performance monitoring of the SCD algorithm, only end-to-end execution times were recorded.

Due to the limited performance of the embedded system, the number of test cases used was reduced to 3000 randomly generated VR sets for each fixed number of VRs in the set (i.e. 5, 10, 15,. . . , 45, 50 VRs). The ITU (i.e. 30%, 50% or 80%) and the type of the VR sets (only continuous, only discrete or mixed type) were randomly created for each of the 3000 sets. Nevertheless, sufficient data was captured to analyse the behaviour of the SCD algorithm on real hardware.

7.2. Measurement

During the design phase of real-time systems, information is required about the expected complexity of the SCD algorithm. In the following, an upper bound equation will be defined that allows engineers to determine during the design phase the required amount of budget for the SCD algorithm.

As an initial step, the dependency of the SCD algorithm's execution time on the number of iterations and VRs in the system was examined. Figure 4 shows for different numbers of VRs the execution time plotted against the number of iterations and the corresponding regression lines. Since the scatter of the measured values along the x-axis is very narrow for the test-cases with 5, 10 and 15 VRs, their regression lines are omitted.

To determine the necessary values for the linear upper bound equation, the least-squares linear regression method was applied to the collected data. Table 1 summarises the parameters of the regression lines from Figure 4. The data in Table 1, as well as visual inspection of Figure 4 indicate that the slope of each regression line is very similar. This observation suggests a linear dependency of execution time on the number of iterations.

But a single linear equation is not sufficient to express the execution time of the SCD algorithm. Since the regression lines do not overlap but have an offset between them, the dependency of this offset on the number of VRs has been examined as well. In Figure 4 the intersection points of the regression lines with z-y-plane shows that the offset also increases linearly with the number of VRs in the system.

Table 1. Regression line parameters

VR#	Function	Slope (10^{-3})	Y-axis offset
50	$f_{50}(x)$	1.5425	99.287
45	$f_{45}(x)$	1.5520	87.238
40	$f_{40}(x)$	1.5579	75.519
35	$f_{35}(x)$	1.5722	64.308
30	$f_{30}(x)$	1.5862	53.022
25	$f_{25}(x)$	1.6263	42.277
20	$f_{20}(x)$	1.6386	32.486

The equation, which describes the y-axis intersection of the execution time regression lines against the number of VRs, was also determined via linear regression (see Equation 4).

$$y_0(v) = 2.381 \cdot v - 21.397 \tag{4}$$

The former analysis reveals that the execution time depends mainly on two parameters. Figure 4 shows the execution times plotted along the y-axis. The x-axis represents the number of iterations and the z-axis the number of VRs. The observable linear behaviour of the plotted data along the x-axis as well as along the z-axis suggests the definition of the execution time upper bound as a plane equation with the number of iterations and VRs as independent variables.

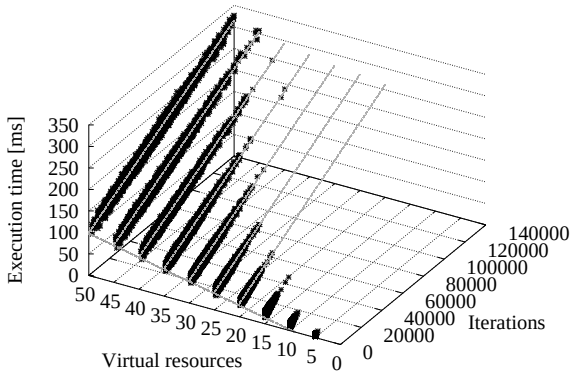


Figure 4. Execution time samples

The general form of the plane equation can be defined as $C(n, v) = a_1 \cdot n + a_2 \cdot v$, with $C(n, v)$ representing the execution time, n the number of iterations and v the number of VRs.

Next, the coefficients, a_1 and a_2, for the plane equation were specified. They were derived from the data that was obtained by measurements on the embedded platform.

The first coefficient, a_1, was determined by calculating the average slope of the execution time regression lines. The average value is approximately $1.58224 \cdot 10^{-3}$, but for ease of use, this value has been rounded up to $1.6 \cdot 10^{-3}$.

Finally, the value of the second coefficient, a_2, was defined. Based on Equation 4 and on the data of Figure 4, the value of 2.8 was determined for a_2. This value was obtained by the application of pragmatic steps in or-

der to simplify Equation 4. The aim was to preserve just a single factor that expresses the dependency between the number of VRs, and the y-axis intersection of the lines that represent the execution time upper bounds for each set of VRs. The value of coefficient a_2 was increased from the value starting at 2.381 (as specified by Equation 4) until the regression lines in Figure 4 became the upper bound on the measured execution times for the corresponding set of VRs (i.e. every execution time sample was below the upper bound).

Using this information, an execution time upper bound equation for the SCD algorithm was derived (see Equation 5). The equation represents a pragmatically derived execution time upper bound for the MPC555 microcontroller that was used to carry out the performance measurements.

$$C(n, v) = 0.0016 \cdot n + 2.8 \cdot v \quad (5)$$

To get an idea of how the execution time upper bound increases with system complexity, the upper bound was plotted against the number of VRs in the system and the number of iterations. The data that was generated by the application of Equation 5 spans a plane in three dimensional space. The plane in Figure 5 illustrates the execution time upper bound of the SCD algorithm. For the sake of clarity contour lines are rendered at 50 ms intervals.

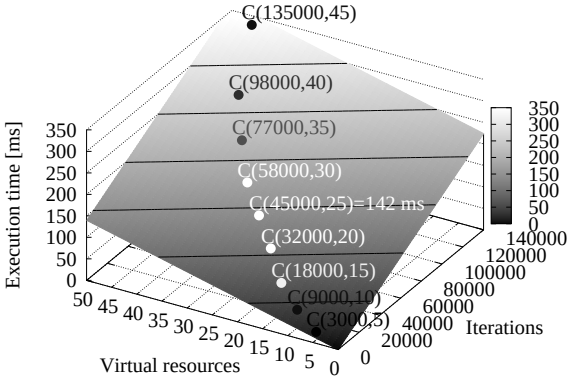


Figure 5. Execution time upper bound

In order to determine the required budget for the SCD algorithm, the expected maximal number of VRs in the system and the granted maximal number of iterations for the runtime of the algorithm has to be specified. The parameter, maximal number of iterations, also influences the probability of the SCD algorithm terminating within the determined budget. The probability of termination within a certain number of iterations has already been investigated during the empirical evaluation in Section 6.2. It can be summarised as follows.

Table 2 shows the maximal number of iterations that were required by the SCD algorithm to terminate for 99.99%, 99.90%, 99.00% and 90.00% of the test-cases. There is a slight difference in the number of required iterations for the algorithm among the test-cases that were performed with 30%, 50% and 80% ITU; but the greatest observed values were chosen.

Table 2. Number of iterations

VR#	99.99%	99.90%	99.00%	90.00%
5	3000	2000	1000	1000
10	9000	6000	4000	3000
15	18000	14000	10000	6000
20	32000	24000	18000	12000
25	45000	36000	27000	18000
30	58000	49000	38000	26000
35	77000	66000	51000	36000
40	98000	86000	68000	49000
45	135000	108000	85000	62000
50	157000	132000	107000	77000

Based on the data that was collected during the empirical evaluation and the performance measurements, the required budget for the SCD algorithm can be derived for a real-time system using an MPC555 microcontroller and a specified maximum number of VRs.

As an example, we now determine the budget for two different configurations. First, the budget for the SCD algorithm on a system with a maximum of 5 application components is calculated. The budget is chosen such that the algorithm can terminate in 99.99% of the cases. Applying the information from Table 2 in Equation 5, provides a budget estimate of $C(3000, 5) = 0.0016 \cdot 3000 + 2.8 \cdot 5 = 18.8ms$. For the second example, we assume a system with at most 25 components. Again, the budget should allow the SCD algorithm to terminate in 99.99% of the cases. Hence, the estimated budget is $C(45000, 25) = 0.0016 \cdot 45000 + 2.8 \cdot 25 = 142ms$. The calculated values for the examples are also illustrated in Figure 5.

The two coefficients, a_1 and a_2, of the linear equation depend on various hardware factors, like the availability and size of data caches, data bus bandwidth, external memory speed, etc. They also depend on the location (i.e. internal or external memory) of the relevant processing data. Therefore, the linear equation representing an upper bound for SCD algorithm's execution time, has to be individually derived for each hardware platform. This can however easily be performed using a suitable program for calibration, such as the one used to generate the results presented here.

8. Conclusion

In this paper we presented an easily and efficiently implementable algorithm for the distribution of a processing resource's spare capacity. The contribution can be summarised as:

- an anytime SCD algorithm that can generate useful results even if the algorithm's execution time is limited,
- runtime adaptation of continuous and discrete VR temporal parameters (i.e. budget and period),
- preventing schedulability test based inefficiency using an exact test.

The efficiency of the SCD algorithm was examined by empirical evaluation and performance measurements on an embedded platform.

The performance evaluation of the SCD algorithm shows that, for example in a system with up to 25 mixed VRs, the algorithm terminates in 99.99% of the cases within 45000 iterations, and within the same number of iterations the processor reaches an average utilisation of 98%. Based on the upper bound equation (Equation 5), the worst-case execution time for 45000 iterations and 25 VRs is equivalent to 142 ms on an MPC555 microcontroller with a system clock of 40 MHz. The MPC555 is not however a sufficiently powerful processor to support 25 application components.

On faster processors, the cost for one iteration of the SCD algorithm, as well as the overall execution time, decreases. Therefore the complexity of a system, measured in terms of the number of active VRs, for which the SCD algorithm can be considered applicable, increases. For example, a processor with approximately 10 times the performance of a 40MHz MPC555 might be used in Avionics or Telecommunications applications that need to support 10 to 25 application components. In this case, such a processor would require at most 15ms to execute the SCD algorithm.

The SCD algorithm shown in this paper has been integrated into the FRESCOR framework, giving it the capability to distribute spare capacity among different application components with the objective of fully using the available processor capacity and maximising the QoS. While the algorithm is based on the current implementation of FRESCOR on fixed priorities, we believe that it is easily extensible to other scheduling policies and tests. As future work we will investigate the mode change protocols used to make effective the calculation of a new resource allocation as a result of the SCD algorithm. Furthermore, the allocation of multiple resources (e.g. processor, network bandwidth, memory, etc.) to application components and their interdependency will be examined.

Acknowledgements

This work was funded in part by the EU FRESCOR project (contract number FP6/2005/IST/5-034026). We would also like to thank Rapita Systems Ltd.[3] for providing the necessary equipment and tools for the performance measurements. Furthermore, we would like to thank Daniel Sangorrín López for his valuable comments.

References

[1] L. Abeni and G. C. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, pages 4–13, Madrid, Spain, Dec 1998.

[2] M. Aldea Rivas, G. Bernat, I. Broster, A. Burns, R. Dobrin, J. M. Drake, G. Fohler, P. Gai, M. González Harbour, G. Guidi, T. L. J. Javier Gutiérrez Garcia, G. Lipari, J. L. M. P. José M. Martínez, J. C. Palencia Gutiérrez, and M. Trimarchi. Fsf: A real-time scheduling architecture framework. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 113–124, 2006.

[3] L. Almeida, S. Fischmeister, M. Anand, and I. Lee. A dynamic scheduling approach to designing flexible safety-critical systems. In *Proceedings of the 7th ACM & IEEE international conference on embedded software*, pages 67–74, 2007.

[4] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[5] G. C. Buttazzo, G. Lipari, and L. Abeni. Elastic task model for adaptive rate control. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 286–295, Washington, DC, USA, 1998. IEEE Computer Society.

[6] G. C. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni. Elastic scheduling for flexible workload management. *IEEE Transactions on Computers*, 51(3):289–302, 2002.

[7] R. I. Davis and A. Burns. Response time upper bounds for fixed priority real-time systems. In *Proceedings of the 29th IEEE Real-Time Systems Symposium*, 2008.

[8] R. I. Davis, A. Zabos, and A. Burns. Efficient exact schedulability tests for fixed priority real-time systems. *IEEE Transactions on Computers*, 57(9):1261–1276, 2008.

[9] Z. Deng, J. W.-S. Liu, and J. Sun. A scheme for scheduling hard real-time applications in open system environment. In *Proceedings of the 9th Euromicro Workshop on Real-Time Systems*, pages 191–199, Toledo, Spain, Jun 1997.

[10] M. González Harbour and M. T. de Esteban. Architecture and contract model for integrated resources. Technical Report D–AC.2v1, Universidad de Cantabria, 2007.

[11] M. González Harbour, D. S. López, and M. T. de Esteban. Mode change protocol for budget changes in contract-based scheduling. Technical report, Universidad de Cantabria, 2008.

[12] J. Kay and P. Lauder. A fair share scheduler. *Communications of the ACM*, 31(1):44–55, 1988.

[13] T.-W. Kuo and C.-H. Li. A fixed-priority-driven open environment for real-time applications. In *IEEE Real-Time Systems Symposium*, pages 256–267, 1999.

[14] J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation*, 2(4):237–250, Dec 1982.

[15] C. Lin, T. Kaldewey, A. Povzner, and S. A. Brandt. Diverse soft real-time processing in an integrated system. In *Proceedings of the 27th IEEE International Real-Time Systems Symposium*, pages 369–378, 2006.

[16] J. Liu. *Real-Time Systems*. Prentice-Hall, Inc., 2000.

[17] R. R. Rajkumar, C. Lee, J. P. Lehoczky, and D. P. Siewiorek. A resource allocation model for qos management. In *IEEE Real-Time Systems Symposium*, pages 298–307, 1997.

[18] D. Rosu, K. Schwan, S. Yalamanchili, and R. Jha. On adaptive resource allocation for complex real-time application. In *Proceedings of the 18th IEEE International Real-Time Systems Symposium*, pages 320–329, 1997.

[19] L. Sha, J. P. Lehoczky, and R. R. Rajkumar. Solutions for some practical problems in prioritizing preemptive scheduling. In *Proceedings of the 7th IEEE Real-Time Sytems Symposium*, pages 181–191, 1986.

[20] B. Sprunt, L. Sha, and J. P. Lehoczky. Aperiodic task scheduling for hard real-time systems. *Real-Time Systems*, 1(1):27–60, 1989.

[21] K. W. Tindell and A. Alonso. A very simple protocol for mode changes in priority preemptive systems. Technical report, Universidad Politecnica de Madrid, 1996.

[22] A. Zabos, R. I. Davis, and A. Burns. Utilization based spare capacity distribution. Technical Report YCS-2008-427, University of York, 2008.

[3] Rapita Systems Ltd. Available at http://www.rapitasystems.com (8 January 2009)