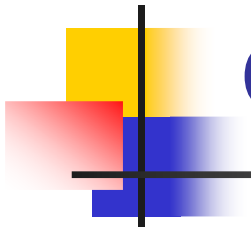
Spare Capacity Distribution Using Exact Response-Time Analysis

Attila Zabos, Robert I. Davis, Alan Burns,
Michael Gonzalez Harbour

Department of Computer Science

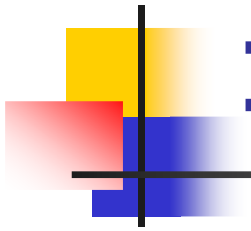
RTS York

THE UNIVERSITY of York



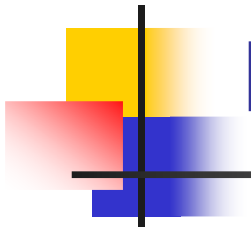
Overview

- Envisaged systems
- Motivation for the work
- Spare capacity distribution (SCD)
- Evaluation results
- Conclusion



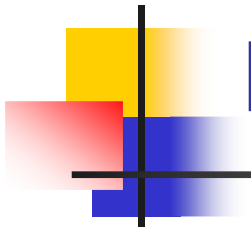
Introduction

- Our focus is on flexible real-time system providing on open environment
  - In these systems we are faced with:
    - Changing scenarios of running applications
    - Flexible applications that can adapt to the available resources
  - Hence, in these systems there is a need for resource management to support
    - Flexibility of real-time applications and
    - Dynamic system behaviour



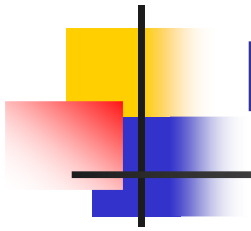
Problem

- The problem with traditional real-time system modelling
  - Its focus is on static configuration
    - Fixed processing resource assignment to applications during the entire runtime of the system
    - But static resource reservation is infeasible for dynamic systems where
      - The system load may change due to a changing set of active applications



Motivation

- Adapt the timing behaviour of flexible applications
  - To maximise the available processing resource utilisation
- Adaptation shall occur for long term system changes of dynamic systems
  - When applications join and leave the system
    - Since not every application needs to execute during the entire lifetime of a system



Motivating examples

- Environment suitable for flexible applications can be found in:
  - Mobile devices
  - Robotics
  - Process control
- Flexible applications
  - Multimedia, e.g. audio/video quality
  - Artificial intelligence, e.g. accuracy of path planning result
  - Control tasks, e.g. control loop result quality

Flexible applications

- Properties of flexible applications:
  - Specify minimal processing resource requirements
  - Capable of using additional resources
  - Different levels of adaptation
    - Continuous
      - Video player: continuous improvement of image quality
    - Discrete
      - Control application: various versions of control algorithm (low, medium, high quality)

Runtime adaptation

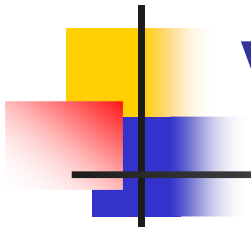
- Exploiting the flexibility of applications to maximise the usage of processing resource
  - Require a runtime adaptation of resource reservation for these applications
- Implementation of the runtime adaption
  - Resource managing middleware on top of a Real-Time OS (FRESCOR framework)
    - Applications specify resource requirement via *contracts*
    - Runtime negotiation of resource assignment

Virtual resource creation

- Contracts specify
  - The minimal resource requirements
  - But also how additional resources can be utilised by the application
- After successful negotiation of the contracts
  - A virtual resource (VR) is created for every new application

Virtual Resources

- Each VR
  - A VR provides resource reservation for the corresponding application
  - Responsible for the management of the associated application‘s resource consumption



Virtual Resources

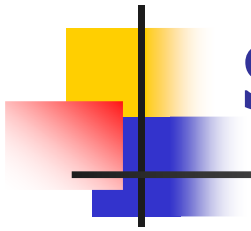
- Analogous to application types
  - We distinguish between continuous and discrete VRs
- VR temporal parameters
  - Continuous: Budget $[C_{min},C_{max}]$ and Period $[T_{min},T_{max}]$
  - Discrete: Budget, Period, Deadline (C_m,T_m,D_m) tuples

Changing resource demand

- Redistribution of resource capacity when
  - Applications join the system and VRs are created; and applications terminate, hence VRs are destroyed
- Since changes of the system load occur during runtime
  - The VR parameter selection has to be an online task
    - Objective: maximise processor utilisation
    - Controlled by *importance* and *weight* attribute
      - Provide system integrator control over SCD

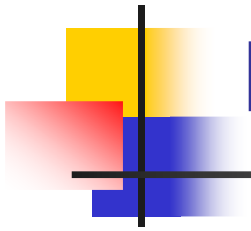
Spare capacity distribution

- Uses exact response time analysis
  - To test the feasibility of VRs in the system
- Implemented as an any-time algorithm
  - Longer runtime provides better distribution of spare capacity
  - Using bisection to determine capacity distribution among running applications
  - Increase processor utilisation by
    - Increasing resource reservation for applications
    - Selecting appropriate VR temporal parameter values



SCD algorithm summarised

- Ensure minimal reservation for each VR
- Spare capacity distributed in decreasing order of VR importance
  - Higher importance infinitely more valuable
  - Capacity increment of each VR expressed as processor utilisation
    - Capacity distribution among VRs at same importance controlled by *weight* attribute (fair share)
  - Capacity increase (resource reservation for applications) within the temporal parameter limits of associated VRs



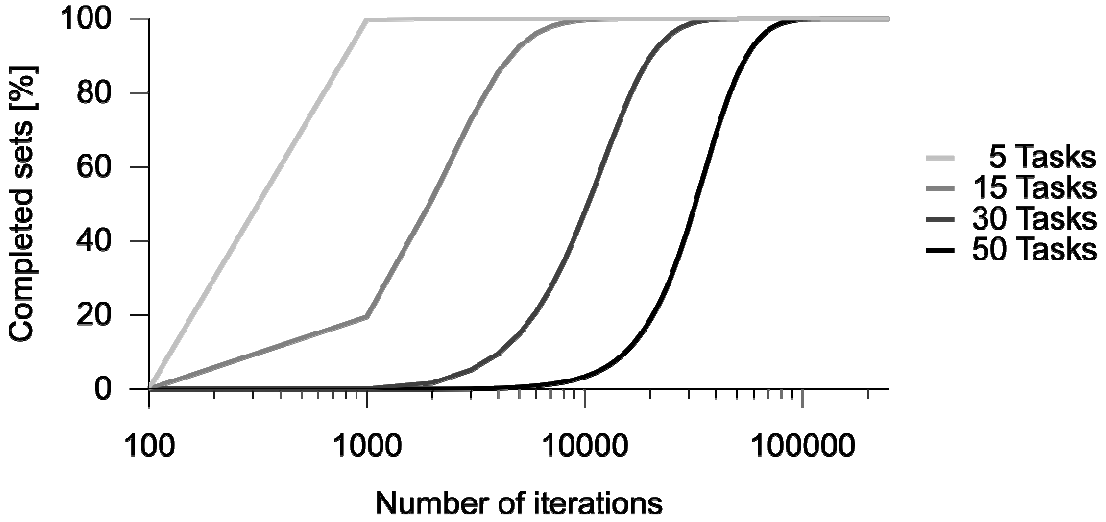
Evaluation

- Empirical evaluation
  - Desktop PC
  - Metric used was the number of ceiling operations of the schedulability test
- Furthermore, we ran execution time measurement
  - On an MPC555 embedded board with 40MHz system clock
- Deriving a measurement based execution time upper bound for the SCD algorithm

Test data generation

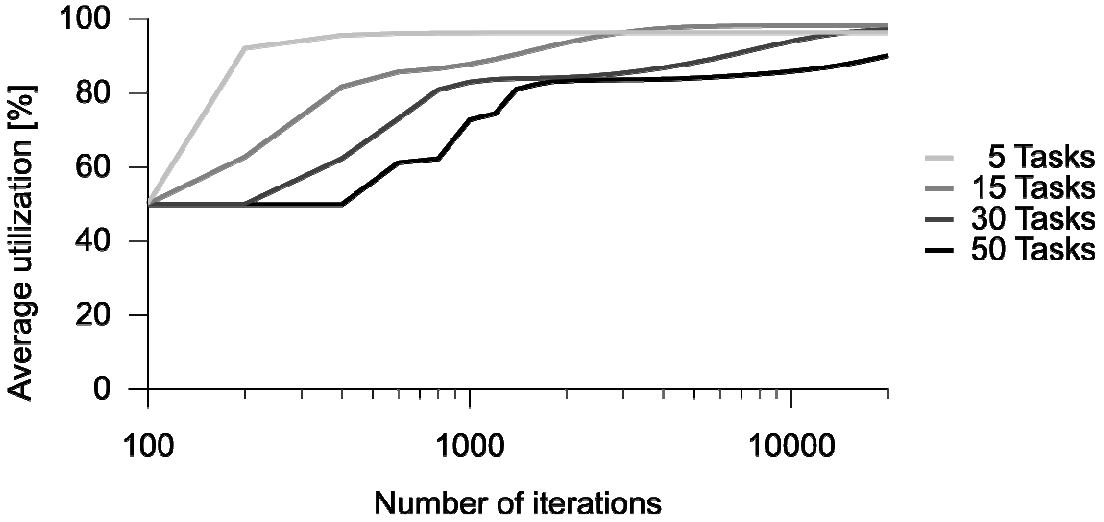
- Test variables
  - Number of VRs
    - 5, 10, 15, ... 40, 45, 50
  - Processor utilisation of VRs
    - 30%, 50%, 80%
  - Type of VR sets
    - Continuous, discrete, mixed
- 100000 different test-cases for each permutation of the test variables

Empirical evaluation



- Cumulative number of terminated test-cases
- Test-cases terminated within a reasonable number of iterations
  - We note that more VRs require higher number of iterations

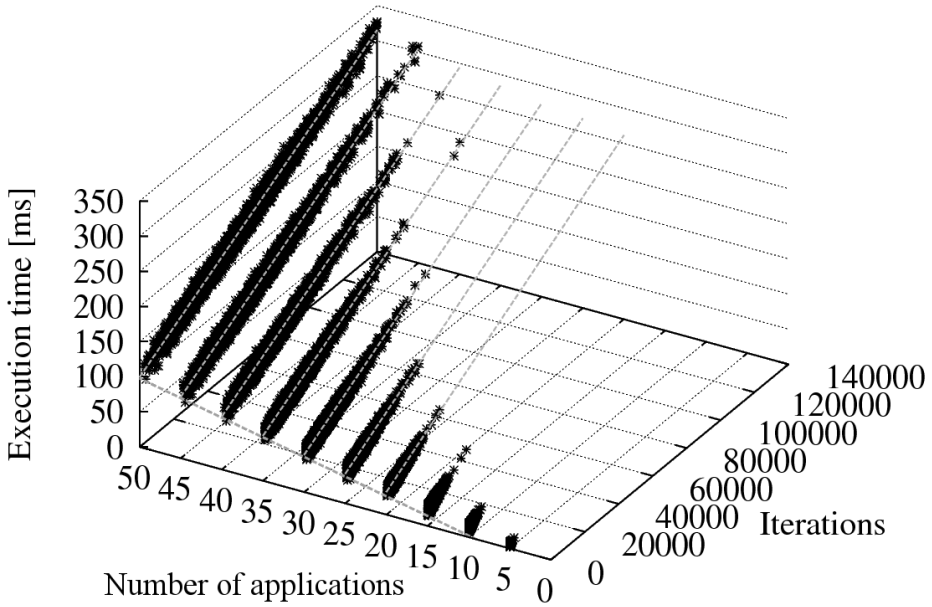
Empirical evaluation



- Processor utilization increases as a function of iterations
- High utilization is achieved within a low number of iterations
  - 80% processor reservation within 2000 iterations

Performance measurements

- Measurements indicate linear dependency of
  - Number of iterations and
  - Number of virtual resources



- SCD algorithm execution time upper bound defined by a plane: $C(n,v)=a_1*n+a_2*v$
- The two coefficients (a_1 and a_2) of the plane equation are hardware dependent

Summary

- Algorithm provided for processor spare capacity distribution
  - Easily and efficiently implementable algorithm
  - Incrementally improves the distribution result
  - Prevent schedulability test based inefficiency due to an exact test
- Extensive evaluation and Proof-of-Concept implementation proved SCD algorithm applicability in real world systems