
K. Ecker (TU Clausthal)
D. Juedes, L. Welch, D. Chelberg, C. Bruggeman, F. Drews, D. Fleeman, D. Parrot (Ohio University, Athens)
B. Pfarr (NASA)
Overview

1. Introduction
2. Definition of a system and optimization model
3. Heuristic algorithm for the pre-runtime analysis
4. Run-time system
Introduction

Static real-time systems

Common real-time engineering approach

- use "worst-case" execution times
- allocate computing and network resources to processes at design time
Static vs. dynamic real-time systems

Problems with common approaches:

- may lead to poor resource utilization
- limit the options to handle overloading of system resources
- limit the functionality and flexibility of applications to adapt to new situations
- inappropriate for applications that must operate in highly dynamic environments

Dynamic real-time systems ... response to environmental changes
How to react in response to environmental changes?

1. Reallocation approach
   Dynamically reconfiguring the way in which computing and network resources are allocated to processes

2. QoS (Quality of Service) level and utility approach
   Adapt service levels to the changed requirements such that utility is maximized
**QoS approach: Utility function and service levels**

**Utility function:**

- Defined as the "user perceived benefit by performing a certain action with a particular fidelity within the given time constraints"
- Function that describes the quality of the output of an application
- Specified by the user
QoS approach: Utility function and service levels

Service levels:

- Define the different quality of service (QoS) levels at which an application can operate
- Each service level defines a strategy for doing the applications work

Example: NASA HART compression agent (video/audio server)

<table>
<thead>
<tr>
<th>Service level (compression)</th>
<th>none</th>
<th>lossless</th>
<th>lossy 1</th>
<th>lossy 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Utility</td>
<td>1.0</td>
<td>0.8</td>
<td>0.3</td>
<td>0.1</td>
</tr>
</tbody>
</table>

K. Ecker et al., An Architecture and a Global Optimization Framework …
Our approach

Most existing approaches perform
- either only reallocation of resources
- or only adaptation of QoS levels
in response to dynamic environment changes

Q-RAM [Raikumar et al., 1998]

Our approach: Adaptive Resource Manager

Combining the dynamic reallocation and the service level/utility concept
Overview

1. Introduction

2. Definition of a system and optimization model

3. Heuristic algorithm for the pre-runtime analysis

4. Run-time system
Definition of a system and optimization model

Hardware components:

- Set of hosts: $H = \{h_1, ..., h_i\}$
  Each host $h_i$ is specified by
  - memory size,  
  - SPEC rates, etc.

- Interconnection network
Definition of a system and optimization model

Application software system:

- **Purpose:** control the environment
- **Logical view** distinguishes several layers of abstraction:
  - Highest level: Total system
  - Next lower level: Collection of subsystems $SS_1, SS_2, ..., SS_m$
  - A subsystem is a collection of paths, each with a given period or maximum event rate
  - A path consists of a set of tasks and possible data dependencies
Example: Prototype autonomous NASA satellite system

HART (Hierarchical Agent-based Real-time Technology)
Definition of a system a optimization model

Software components

Set of tasks: \( A = \{T_1, ..., T_n\} \)

Each task is specified by

- execution time profiles and memory profiles
  - depending in the processor \( T_i \) is assigned to,
  - depending on certain operational conditions set by the environment ("extrinsic parameters")
  - depending on certain parameters that can be changed at run time ("service parameters")

- periodic tasks: period \( \pi_i \)

- event-driven tasks: maximum event rate \( r_i \)

- \( r_i \in (0, \infty) \) . . . modeled by \( \pi_i = 1/r_i \)
Extrinsic and service attributes

Extrinsic attributes: $E = (e_1, ..., e_k)$

- express functional conditions or requirements
- are either posted by the environment, or by the status of system components
- cannot be changed by the real-time system

General assumptions:

- values of extrinsic parameters are discrete (integer)
- parameter $e$ ranges in known interval $[0, \text{max}_e]$

Examples: event rate of event-driven tasks, workload, etc.
Extrinsic and service attributes

Service attributes (QoS levels): \( S = (s_1, ..., s_l) \)

- can be changed at any time by the resource manager
- if external conditions change, appropriate settings of the service attributes allow adaptation to the new conditions

General assumptions:

- values of service parameters are discrete (integer)
- parameter \( s \) ranges in known interval \([0, \text{max}_s]\)

Examples: compression type, sample rate, etc.
Utility functions

For local optimization, each subsystem $SS_i$ is provided a local utility function $U_i(S,E)$ that measures the contribution of $SS_i$ to the overall utility.

The overall system utility can be computed from the subsystem utilities by means of some aggregation function

$$U_{\text{System}}(S,E) := AGR(U_i(S,E))$$

Example: $U_{\text{System}}(S,E) := \sum w_i U_i(S,E)$
Objective

Given $E$, find an allocation of tasks to hosts $\text{alloc} : A \rightarrow H$ and settings of service attributes $S$ and settings of maximum manageable extrinsic attributes $E_m \geq E$ such that

1. run time conditions and memory limitations on the hosts are satisfied
   in particular, all the tasks meet their deadlines

2. The overall system utility $U_{\text{System}} (S, E_m)$ is maximum

3. The values of the maximum manageable extrinsic attributes $E_m$ are as high as possible
Service tables

Ideas:

• Using a table look-up technique for the resource manager (RM)

• RM has to maintain tables of "candidate solutions" along with service attribute settings and (maximum) manageable extrinsic attributes that are determined in a pre-runtime analysis

• Service table: for a given allocation $alloc$:

$$ST(alloc) = \{ \langle S, E_m \rangle \mid E_m \text{ is manageable under service settings } S \}$$
Overview

1. Introduction
2. Definition of a system and optimization model
3. Heuristic algorithm for the pre-runtime analysis
4. Run-time system
Heuristic algorithm for pre-runtime analysis

Objective of the pre-runtime analysis

- Investigate operational limits of the control system subject to maximizing the overall system utility

- Provide the allocation manager with a set of "candidate solutions" along with the settings of service attributes, that allow maximum manageable extrinsic attribute values
General Assumptions

Monotonicity of the memory and execution time profiles with respect to the extrinsic attributes

(in general, we do not assume monotonicity of the service attributes)
General Strategy

- A set of allocations

\[ \{\text{alloc}_1, ..., \text{alloc}_K\} \]

is chosen heuristically

- For each allocation and for each setting of service attributes it is checked on which (extrinsic) grid points the allocation is feasible

Maximum points (pair-wise incomparable) are referred to as Pareto points
General Strategy

Pareto points:

For each allocation \( alloc_i \) \( (i = 1, ..., K) \) and a vector \( S \),
\( \text{Pareto}_S(alloc_i) = \{ E_1, E_2, ... \} \) denotes a set of (extrinsic) grid points that have the properties

• \( alloc_i \) is feasible

• an increase in any component of \( E_i \in \text{Pareto}_S(alloc_i) \) leads to an infeasible allocation
Example

For given allocation \( alloc_i \) and a setting of service parameters \( S \):

\[ e_1 \]

\[ e_2 \]

Pareto-optimal solutions

Pareto points of given allocation
Pareto-optimal service tables

Pareto-optimal service tables can now be defined by:

$$ST(\text{alloc}_i) = \{ \langle S, E \rangle \mid E \in \text{Pareto}_S(\text{alloc}_i) \text{ for all } S \}$$

Objective: Provide the allocation manager with "good" service tables

Problem: Given an allocation $\text{alloc}$ and service attribute values $S$, how to determine the corresponding set of all Pareto points $\text{Pareto}_S(\text{alloc})$ efficiently?
Recursive stepwise refinement strategy

Main loop of optimization:

- alloc = GENERATE()
- Determine Service Table ST(alloc)
- Stopping Criterion

Simple strategy "brute force":
- test all the grid points for feasibility
  - high complexity, inefficient

More advanced approach:
- recursive stepwise refinement
  - lower complexity, "nice" stopping criterion possible
Stepwise refinement strategy
Stepwise refinement strategy
Stepwise refinement strategy
Stepwise refinement strategy
Stepwise refinement strategy
Complexity of stepwise refinement strategy

Lemma. Let \( \hat{e}_i \) be the smallest power of two, for which

\[
\hat{e}_i \geq \max_e e_i + 1 \quad (i = 1, \ldots, k)
\]

If \( k = 1 \) (only one extrinsic attribute) then the time complexity is \( O(\log \hat{e}_1) \)

If \( k = 2 \), \( \hat{e}_1 \geq \hat{e}_2 \), then the time complexity is \( O(\hat{e}_1^\alpha) \) with \( \alpha = \log_2 3 = 1.584... \)

Lemma. For grids of arbitrary but fixed dimension \( k \), and \( \hat{e}_1 \geq \hat{e}_2 \geq \ldots \geq \hat{e}_k \), the time complexity is

\[
O(\hat{e}_1 \log_2(2^k-1)) = O(\hat{e}_1^k)
\]
Overview

1. Introduction
2. Definition of a system and optimization model
3. Heuristic algorithm for the pre-runtime analysis
4. Run-time system
Run-time system: the resource manager

Resource Manager $RM$

Allocation Manager $AM$

Global Meta Agent $GMA$

Meta Agent $MA_1$

Meta Agent $MA_2$

Meta Agent $MA_m$

$SS_1$

$SS_2$

$SS_m$
Run-time system: working with service tables

Example: \( l = 1 \) service attribute \((s_1)\) with possible values \( \in \{0,1\} \)
\( k = 2 \) extrinsic attributes, both with possible values \( \in \{0..8\} \)
Consider \( K = 2 \) allocations: \( alloc_1 \) and \( alloc_2 \)
### Example service tables

#### Service table for $alloc_1$:

<table>
<thead>
<tr>
<th>$S = (s_1)$</th>
<th>$E_m = (e_1, e_2)$</th>
<th>$U(S, E)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(0, 8)</td>
<td>8</td>
</tr>
<tr>
<td>0</td>
<td>(3, 6)</td>
<td>9</td>
</tr>
<tr>
<td>0</td>
<td>(5, 4)</td>
<td>9</td>
</tr>
<tr>
<td>0</td>
<td>(6, 1)</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>(7, 0)</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>(1, 8)</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>(4, 6)</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>(5, 2)</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>(6, 1)</td>
<td>8</td>
</tr>
</tbody>
</table>

#### Service table for $alloc_2$:

<table>
<thead>
<tr>
<th>$S = (s_1)$</th>
<th>$E_m = (e_1, e_2)$</th>
<th>$U(S, E)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(0, 6)</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>(2, 4)</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>(4, 3)</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>(6, 1)</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>(8, 0)</td>
<td>8</td>
</tr>
<tr>
<td>0</td>
<td>(0, 7)</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>(2, 6)</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>(3, 3)</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>(5, 1)</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>(7, 0)</td>
<td>8</td>
</tr>
</tbody>
</table>
Cooperation AM $\leftrightarrow$ GMA

For given $E$, the AM chooses a feasible allocation $alloc$, and returns the corresponding service table to the GMA.

The GMA

- selects a table entry that offers maximum system utility $U_{System}(S, E_m)$; thus values for $S$ and $E_m \geq E$ are specified
- and forwards sub-tables with the possible parameter settings to the meta agents
Cooperation AM ↔ GMA

If extrinsic parameters change:

- The corresponding meta agents are allowed to optimize service parameters locally within the frame or their sub-tables.

- If local optimization is not possible, the GMA selects new service parameter settings from its service table.

- If the GMA fails to feasibly handle the new extrinsic values - or if the system utility drops considerably - then GMA triggers the AM to provide another allocation.
Summary

- Model for a dynamic, distributed real-time system
- Architecture of an adaptive resource manager
- Maximizing QoS
- Heuristic algorithmic approach based on table look-up technique
Next steps:

- Application of different local search based heuristic approaches (genetic algorithms, simulated annealing, etc.)
- Comparison with on-line greedy heuristics
- Integration into HART prototype and simulation
- Other applications: RoboCup, Dynbench, etc
- More theoretical investigations