Technical Report

Coordinated Runtime Adaptations in Cooperative Open Real-Time Systems

Luís Nogueira

Luís Miguel Pinho

Jorge Coelho

HURRAY-TR-090612

Version: 0

Date: 08-31-2009

Coordinated Runtime Adaptations in Cooperative Open Real-Time Systems

Luís Nogueira, Luís Miguel Pinho, Jorge Coelho

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail: luis@dei.isep.ipp.pt, lpinho@dei.isep.ipp.pt, jcoelho@dei.isep.ipp.pt

http://www.hurray.isep.ipp.pt

Abstract

This paper proposes an one-step decentralised coordination model based on an effective feedback mechanism to reduce the complexity of the needed interactions among interdependent nodes of a cooperative distributed system until a collective adaptation behaviour is determined. Positive feedback is used to reinforce the selection of the new desired global service solution, while negative feedback discourages nodes to act in a greedy fashion as this adversely impacts on the provided service levels at neighbouring nodes. The reduced complexity and overhead of the proposed decentralised coordination model are validated through extensive evaluations.

Coordinated Runtime Adaptations in Cooperative Open Real-Time Systems

Luís Nogueira[1], Luís Miguel Pinho[1], Jorge Coelho[2]
IPP Hurray Research Group[1]
School of Engineering, Polytechnic Institute of Porto, Portugal
LIACC[2]
School of Engineering, Polytechnic Institute of Porto, Portugal
{luis,lpinho,jcoelho}@dei.isep.ipp.pt

Abstract

This paper proposes an one-step decentralised coordination model based on an effective feedback mechanism to reduce the complexity of the needed interactions among inter-dependent nodes of a cooperative distributed system until a collective adaptation behaviour is determined. Positive feedback is used to reinforce the selection of the new desired global service solution, while negative feedback discourages nodes to act in a greedy fashion as this adversely impacts on the provided service levels at neighbouring nodes. The reduced complexity and overhead of the proposed decentralised coordination model are validated through extensive evaluations.

1 Introduction

Adaptive real-time is a relatively new approach to open embedded systems design. It allows an online reaction to load variations, adapting the system to the specific constraints of devices and users, nature of executing tasks and dynamically changing environmental conditions. This ability is essential to efficiently manage the provided Quality-of-Service (QoS) in domains such as telecommunications, consumer electronics, industrial automation, and automotive systems.

While there has been a great deal of research in several aspects of runtime adaptation in embedded real-time systems, there has been very little work which addresses the specific problem of coordinating inter-dependent autonomous adaptations in distributed environments. However, coordination is critical to maintain the correctness of a distributed application during adaptation [1, 8] and also very challenging. In particular, a QoS-aware adaptation in a service's distributed execution can require communication and synchronisation among nodes, used as a building block for a collective adaptation behaviour that emerges from local interactions among nodes.

One main challenge is controlling this exchange of information in order to achieve a convergence to a globally consistent solution without overflowing nodes with messages. This paper shows how it is possible to achieve higher levels of self-adaptiveness in systems required to work in time critical environments through a fast convergence decentralised coordination model. With the increasing size and complexity of open embedded systems the ability to build self-managed distributed systems using centralised coordination models is reaching its limits [13], as solutions they produce require too much global knowledge.

The proposed one-step decentralised coordination model defines a set of rules through which a coalition of nodes can be reconfigured, imposing restrictions on the way their members can self-adapt in response to changes in the environment. By exchanging feedback on the desired self-adaptive actions, nodes converge towards a global solution, even if that means not supplying their individually best solutions. As a result, each node, although autonomous, is influenced by, and can influence, the behaviour of other nodes in the system.

2 System model

Consider an open distributed system with several heterogeneous nodes, each with its specific set of resources, where independently developed services, some of them with real-time execution constraints, can appear while other are being executed, at any time, at any node. Due to these characteristics, resource availability is highly dynamic and unpredictable in advance.

It is assumed that each service S has a set of QoS parameters that can be changed in order to adapt the system's service provisioning to a dynamically changing environment. Each subset of parameters that relates to a single aspect

of service quality is named as a *QoS dimension*. Each of these QoS dimensions has different resource requirements for each possible level of service. We make the reasonable assumption that services' execution modes associated with higher QoS levels require higher resource amounts.

There may exist QoS dependencies among two or more of the multiple QoS dimensions of a service S [18]. Given two QoS dimensions, Q_a and Q_b, a QoS dimension, Q_a, is said to be dependent on another dimension Q_b if a change along the dimension Q_b will increase the needed resource demand to achieve the quality level previously achieved along Q_a.

Users provide a single specification of their own range of QoS preferences Q_{Pref} for a complete service S, ranging from a desired QoS level $L_{desired}$ to the maximum tolerable service degradation, specified by a minimum acceptable QoS level $L_{minimum}$, without having to understand the individual components that make up the service.

For some of the system's nodes there may be a constraint on the type and size of services they can execute within the users' acceptable QoS levels. Therefore, the CooperatES framework [14, 16] allows a distributed cooperative execution of resource intensive services in order to maximise the users' satisfaction with the obtained QoS. By redistributing the computational load across a set of nodes, a cooperative environment enables the execution of far more complex and resource-demanding services than those that otherwise would be able to be executed on a stand-alone basis [16].

Each coalition is dynamically formed as the set of nodes that maximise the satisfaction of the user-imposed QoS constraints associated with the a new service S and minimise the impact on the previous global QoS caused by the new service's arrival [14], establishing an initial Service Level Agreement (SLA), a service description whose parameters are within the range of the user's accepted QoS levels.

Due to the existence of QoS inter-dependencies among the tasks of a service S, there are groups of tasks that must be executed in the same node within a coalition [15]. We refer to such groups as *work units*. Let $W = \{w_1, \ldots, w_n\}$ be the finite set of work units of a service S. We represent the set of inter-dependencies among work units $w_i \in W$ as a connected graph $\mathcal{G}_{\mathcal{W}} = (\mathcal{V}_W, \mathcal{E}_W)$, on top of the service's distribution graph, where each vertex $v_i \in \mathcal{V}_{\mathcal{W}}$ represents a work unit w_i and a directed edge $e_i \in \mathcal{E}_{\mathcal{W}}$ from w_j to w_k indicates that w_k is functionally dependent on w_j. Within $\mathcal{G}_{\mathcal{W}} = (\mathcal{V}_W, \mathcal{E}_W)$, we call *cut vertex* to a node $n_i \in \mathcal{V}_{\mathcal{W}}$, if the removal of that node divides $\mathcal{G}_W$ in two separate connected graphs.

We assume that each work unit $w_i \in W$ is being executed at its current QoS level Q^i_{val} at a node n_i, from a set of predefined QoS levels $\{Q_0, \ldots, Q_n\}$, ranging from the user's desired QoS level $L_{desired}$ to the maximum tolerable service degradation, specified by the minimum acceptable QoS level $L_{minimum}$. This relation is represented by a triple (n_i, w_i, Q^i_{val}). Furthermore, for a given work unit $w_i \in W$, each node n_i knows the set $I_{w_i} = \{(n_j, w_j, Q^j_{val}), \ldots, (n_k, w_k, Q^k_{val})\}$, describing the quality of all the inputs related to work unit w_i coming from its adjacent nodes in $\mathcal{G}_{\mathcal{W}}$.

3 Self-managing cooperative systems

One of the key ideas of the CooperatES framework is that each coalition member should be able to take the initiative and to decide when and how to adapt to changes in the environment [14]. However, whenever such autonomous adaptations have an impact on the outputted QoS of other coalition members the need of coordination arises, that is, how to ensure that local, individual, adaptation actions of a node can produce a globally acceptable solution for the entire distributed service [9].

With the increasing size and complexity of open embedded systems the ability to build self-managed distributed systems using centralised coordination models is reaching its limits [13], as solutions they produce require too much global knowledge. As such, researchers are increasingly investigating decentralised coordination models to establish and maintain system properties [5, 10, 4, 2, 6]. Without a central coordination entity, the collective adaptation behaviour must emerge from local interactions among nodes. This is typically accomplished through the exchange of multiple messages to ensure that all involved nodes make the same decision about whether and how to adapt.

Bridges et al. [3] propose a framework based on Cactus [22] which supports adaptations that span multiple components and multiple nodes in a distributed system. The architecture supports coordinated adaptations across layers using a fuzzy logic based controller module and coordination across hosts using a prototype protocol designed for communication oriented services. However, the authors do not specifically propose a method to obtain a globally coordinated solution.

Ren et al. [19] present a real-time reconfigurable coordination model (RT-RCC) that decomposes dynamic real-time information systems based on the principle of separation of concerns, namely, functional actors which are responsible for accomplishing tasks, and non-functional coordinators which are responsible for coordination among the functional actors. High level language abstractions and a framework for actors and coordinators are provided to facilitate programming with the RT-RCC model.

A similar approach is followed by Kwiat et al. [12] who propose a coordination model for improving software system attack-tolerance and survivability in open hostile environments. The coordination model is based on three active entities: actors, roles, and coordinators. Actors abstract the

system's functionalities, while roles and coordinators statically encapsulate coordination constraints and dynamically propagate those constraints among themselves and onto the actors. Both the coordination constraints and coordination activities are distributed among the coordinators and roles, shielding the system from single points of failure.

However, none of these works proposes support for coordinating inter-dependent autonomous QoS adaptations in cooperative systems, which is the focus of our work. Furthermore, with some decentralised coordination models it becomes difficult to predict the exact behaviour of the system taken as a whole because of the large number of possible non-deterministic ways in which the system can behave [20]. Whenever real-time decision making is in order, a timely answer to events suggests that after some finite and bounded time we would expect the global adaptation process to converge to a consistent solution. Furthermore, optimal decentralised control is known to be computationally intractable [4], although near-optimal systems can be developed for certain classes of applications [11, 7, 6].

Our goal is then to achieve a fast convergence to a global solution through a regulated decentralised coordination without overflowing nodes with messages. The next section details the proposed one-step decentralised coordination model based on an effective feedback mechanism to reduce the complexity of the needed interactions among nodes until a collective adaptation behaviour is determined.

3.1 A one-step decentralised coordination model

The core idea behind the proposed decentralised coordination model is to support distributed systems composed of autonomous individual nodes working without any central control but still producing the desired function as a whole. We model a self-managed coalition as a group of nodes that respond to environmental inter-dependent changes according to a distributed coordination protocol defined by the following phases:

1. **Coordination request.** Whenever $Q^i_{val'}$, the needed downgrade or desired upgrade of the currently outputted QoS Q^i_{val} for a work unit $w_i \in S$, has an impact on the currently supplied QoS level of other work units $w_j \in S$ being executed at other coalition members, a coordination request is sent to the affected partners.

2. **Local optimisation.** Affected partners, executing any $w_j \in S$, become aware of the new output values $Q^i_{val'}$ of w_i and recompute their local set of SLAs in order to formulate the corresponding feedback on the requested adaptation action. We assume that coalition partners are willing to collaborate in order to achieve a global coalition's consistency, even if this might reduce the utility of their local optimisations. However, a node only agrees with the requested adaptive action if and only if its new local set of SLAs is feasible.

3. **Adaptive action.** If the requested adaptive action is accepted by all the affected nodes in the coalition, the new local set of SLAs is imposed at each of those coalition members. Otherwise, the currently global QoS level of service S remains unchanged.

Furthermore, we consider the existence of feasible QoS regions [21]. A region of output quality $[q(o)_1, q(o)_2]$ is defined as the QoS level that can be provided by a work unit when provided with sufficient input quality and resources. Within a QoS region, it may be possible to keep the current output quality by compensating for a decrease in input quality by an increase in the amount of used resources or vice versa. As such, if a node n_j, despite the change in current quality of some or all of its inputs, is able to continue to produce its current QoS level there is no need to further propagate the required coordination request along the dependency graph $\mathcal{G}_{\mathcal{W}}$. Thus, a *cut-vertex* is a key node in our approach.

Consider that a node n_k proposes an upgrade to $Q^k_{val'}$ for a work unit $w_k \in S$. It may happen that some other nodes, precedent in the path until the next cut-vertex n_c, may be able to upgrade to $Q^j_{val'}$ and others may not. Whenever the cut-vertex n_c receives the upgrade request and its new set of inputs, if it is unable to upgrade to $Q^c_{val'}$ then all the effort of previous nodes to upgrade is unnecessary and the global upgrade fails. Otherwise, the upgrade coordination request continues along the graph, until the end-user node n_u is reached.

In the case of a downgrade to $Q^k_{val'}$ initiated by node n_k, it may happen that some other nodes in the path to the next cut-vertex n_c may be able to continue to output their current QoS level despite the downgraded input by compensating with an increased resource usage and others may not. Again, if the next cut-vertex $_c$ is unable to keep outputting its current QoS level then all the precedent nodes which are compensating their downgraded inputs with an increased resource usage can downgrade to $Q^j_{val'}$ since their effort is useless.

The formulation of the corresponding positive or negative feedback, at the *local optimisation* phase, must depend on the feasibility of the requested coordination action as a function of the quality of the node's new inputs I_{w_i} for the locally executed work unit w_i. Such feasibility is determined by the anytime local QoS optimisation algorithm detailed in Algorithm 1 which aims to minimise the impact of the requested changes on the currently provided QoS level of other services. The CooperatES framework differs from other QoS-aware frameworks by considering, due to the increasing complexity of open real-time systems, the needed

tradeoff between the level of optimisation and the usefulness of an optimal runtime system's adaptation behaviour. This idea has been formalised using the concepts of anytime QoS optimisation algorithms [16].

Algorithm 1 Feedback formulation

Let τ^e be the set of previously accepted tasks whose stability period Δ_t has expired.
Let τ^p be the set of all previously accepted tasks whose current QoS cannot be changed.
Each task $\tau_i \in \tau^e \cup \tau^p$ has associated a set of user's defined QoS constraints Q_i.
Let $Q_{kj}[i]$ be the currently provided level of service for attribute j of the k_{th} QoS dimension for each task $\tau_i \in \tau^e$
Let $\mathcal{L}$ be the set of local dependency graphs $\mathcal{G}_{w_i}$ of all work units $w_i \in \tau^e \cup \tau^p$
Let σ be the determined set of SLAs, updated at each step of the algorithm

1: Define SLA'_{w_i}, the requested SLA for the work unit w_i, as a function on the new input values I_{w_i} and the required output level $Q_{val'}$
2: Update σ with SLA'_{w_i}
3: **while** σ is not feasible **do**
4: **if** there is no task τ_i being served at $Q_{kj}[m] > Q_{kj}[n]$, for any j attribute of any k QoS dimension **then**
5: **return FALSE**
6: **end if**
7: **for** each work unit $w_d \subseteq \tau^e \setminus w_i$ **do**
8: **for** each task $\tau_i \in w_d$ **do**
9: **for** each j^{th} attribute of any k QoS dimension in τ_a with value $Q_{kj}[m] > Q_{kj}[n]$ **do**
10: Downgrade attribute j to the previous possible value $Q_{kj}[m+1]$
11: Traverse $\mathcal{G}_{w_d} \in \mathcal{L}$ and change values accordingly
12: Determine the utility decrease of this downgrade
13: **end for**
14: **end for**
15: **end for**
16: Find task τ_{min} whose reward decrease is minimum
17: Define $SLA'_{w_{min}}$ for the work unit where task τ_{min} belongs, setting the QoS values of all affected tasks according with the new value $Q_{yx}[m+1]$ for attribute x of the QoS dimension y for task τ_{min}
18: Update σ with $SLA'_{w_{min}}$
19: **end while**
20: **return TRUE**

Note that the node's local resource usage optimisation associated with the new local set of SLAs can be lower after the coordination request. Recall that we make the assumption that, in cooperative environments, coalition partners are willing to collaborate in order to achieve a global coalition consistency, even if this coordination might reduce the global utility of their local QoS optimisations. The reward achieved by the currently provided SLA for a work unit $w_i \in S$ is measured by considering the proximity of the promised level of service with respect to the weighted user's QoS preferences expressed in decreasing relative order [14].

If all the nodes affected by the requested adaptation sent by node n_i agree with its new service solution, the *adaptive action* phase takes place. A "commit" message is sent by node n_i to its direct neighbours in the dependency graph, which then propagate the message to all the involved nodes in the global adaptation process.

Decentralised control is then a self-organising emergent property of the system. The proposed coordination model is based on these two basic modes of interaction: positive and negative feedback. Negative feedback loops occur when a change in one coalition member triggers an opposing response that counteracts that change at other dependent node. On the other hand, positive feedback loops promote global adaptations. The snowballing effect of positive feedback takes an initial change in one node and reinforces that change in the same direction at all the affected partners. By exchanging feedback on the performed self-adaptations, nodes converge towards a global solution, overcoming the lack of a central coordination and global knowledge.

Note that only one negotiation round is required between any pair of dependent nodes. As such, the uncertain outcome of iterative decentralised control models whose effect may not be observable until some unknowable time in the future is not present in the proposed regulated coordination model. Also note that the normal operation of nodes continues in parallel with the *change acknowledge* and *local optimisation* phases. Every time a node recomputes its set of local SLAs, promised resources are pre-reserved until the global negotiation's outcome is known (or a timeout expires). As such, the currently provided QoS levels only actually changes at the *adaptive action* phase, as a result of a successful global coordination.

Due to the environment's dynamism, more than one coalition member can start an adaptation process that spans multiple nodes at a given time. Such request can either be a downgrade or an upgrade of its current SLA for a work unit w_i of service S. Even with multiple simultaneous negotiations for the same service S, only one of those will result in a successful adaptation at several nodes since, due to local resource limitations, only the minimum globally requested SLA will be accepted by all the negotiation participants. In order to manage these simultaneous negotiations, every negotiation has an unique identifier, generated by the request-

ing node.

3.2 Properties of the coordination model

In this section we provide a global view of what is involved for the general case and analyse some of the properties of the proposed decentralised coordination model. We start with some auxiliary definitions and proofs. For the sake of simplicity, we present the following functions in a declarative notation with the same operational model as a pattern matching-based functional language.

Definition 3.1 *Given a connect graph $\mathcal{G}_W = (\mathcal{V}_W, \mathcal{E}_W)$ and given two work units $w_i, w_j \in \mathcal{V}_W$, we obtain all the nodes in the possible paths between w_i and w_j as the result of the function:*

$$
\begin{array}{ll}
n\_paths(w_i, w_j) = & flatten(n\_paths(w_i, w_j, \emptyset)) \\
 & \\
n\_paths(w_i, w_j, T) = & \emptyset, \text{ if } w_i = w_j \\
n\_paths(w_i, w_j, T) = & \{\{w_i, w_{k_1}\} \\
 & \cup\, n\_paths(w_{k_1}, w_j, T \cup \{w_{k_1}\}), \\
 & \ldots \\
 & \{w_i, w_{k_n}\} \\
 & \cup\, n\_paths(w_{k_n}, w_j, T \cup \{w_{k_n}\})\}, \\
 & \forall w_{k_m} \in \mathcal{V}_W, \text{ such that} \\
 & (w_i, w_{k_m}) \in \mathcal{E}_W \text{ and } w_{k_m} \notin T \\
n\_paths(w_i, w_j, T) = & \bot
\end{array}
$$

Definition 3.2 *Given a set A containing other sets, the function* flatten(A) *is defined as:*

$$
\begin{array}{ll}
flatten(\emptyset) = & \emptyset \\
flatten(A) = & a \cup flatten(A \setminus a), \text{ if } a \in A
\end{array}
$$

Note that the *n_paths* function is a breadth first approach with cycle checking to find nodes in possible paths in graphs. It outputs all the nodes in the possible paths between two nodes n_i and n_j, or returns $\bot$ if there is no path between those two nodes. Nevertheless, for the sake of clarity of presentation, in the remainder of this chapter, we assume that only well-formed dependency graphs are considered in the proposed algorithms.

Proposition 3.1 *Given a connected graph $\mathcal{G}_W = (\mathcal{V}_W, \mathcal{E}_W)$ and two work units $w_i, w_j \in \mathcal{V}_W$, $n\_paths(w_i, w_j, \emptyset)$ terminates and returns all the nodes in the possible paths between w_i and w_j, $\emptyset$ in case $w_i = w_j$, or $\bot$ in case there is no path between $w_i, w_j \in \mathcal{V}_W$.*

Definition 3.3 *Given a node n_i, a work unit w_i , the set of local SLAs $\sigma = \{SLA_{w_0}, \ldots, SLA_{w_p}\}$ for the p locally executed work units, $Q_{val'}$ as the new requested QoS level for a service S, and $I_{w_i} = \{(n_j, w_j, Q^j_{val}), \ldots, (n_k, w_k, Q^k_{val})\}$ as the set of QoS levels given as input to w_i, then the value of $test\_feasibility(n_i, w_i, Q^i_{val'}, I_{w_i})$ is the return value of Algorithm 1 applied to node n_i.*

Lemma 3.1 (Correctness of the feasibility test) *The function $test\_feasibility$ always terminates and returns true if the new required set of SLAs for outputting the QoS level Q'_{val} at work unit w_i is feasible or false otherwise.*

Proof 3.1 *Termination comes from the finite number of tasks τ_i being executed in node n_i and from the finite number of the k QoS dimensions and j attributes being tested. The number of QoS attributes being manipulated decreases whenever a task τ_i is configured to be served at its lowest admissible QoS level $Q_{kj}[n]$, thus leading to termination.*

Correctness comes from the heuristic selection of the QoS attribute to downgrade at each iteration of the algorithm.

Thus, after a finite number of steps the algorithm either finds a new set of feasible SLAs that complies with the coordination request or returns false if the requested SLA for the work unit w_i cannot be supplied.

Definition 3.4 *Given a connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, such that the work unit w_i is being processed by node $n_i \in \mathcal{V}$, and $I = \{(n_j, w_j, Q^j_{val}), \ldots, (n_k, w_k, Q^k_{val})\}$ as the current set of QoS inputs for a work unit w_i, and given T as the set of changed QoS inputs in response to the coordination request, the function $update(I, T)$ updates I with the elements from T:*

$$
\begin{array}{ll}
update(\emptyset, T) = & \emptyset \\
update(I, T) = & \{(n_i, w_i, Q^i_{val'})\} \\
 & \cup\, update(I \setminus (n_i, w_i, Q^i_{val}), T), \text{ if} \\
 & (n_i, w_i, Q^i_{val}) \in I \text{ and } (n_i, w_i, Q^i_{val'}) \in T \\
update(I, T) = & \{(n_i, w_i, Q^i_{val})\} \\
 & \cup\, update(I \setminus (n_i, w_i, Q^i_{val}), T), \text{ if} \\
 & (n_i, w_i, Q^i_{val}) \in I \text{ and } (n_i, w_i, Q^i_{val'}) \notin T
\end{array}
$$

Proposition 3.2 *Given two sets I and T, both with elements of the form (n_i, w_i, Q^i_{val}),* update(I,T) *terminates and returns a new set with the elements of I such that whenever $(n_i, w_i, Q^i_{current}) \in I$ and $(n_i, w_i, Q^i_{new}) \in T$ the pair stored in the returned set is (n_i, w_i, Q^i_{new}).*

Definition 3.5 *Given a node n_i and a work unit w_i, we define the function $get\_input\_qos(n_i, w_i)$ as returning the set of elements (n_j, w_j, Q^j_{val}), where each of these elements represents a work unit w_j being executed at node n_j with an output QoS level of Q^j_{val} used as an input of the work unit w_i at node n_i.*

Given these, th next sections detail how the decentralised coordination model operates on upgrades or downgrades of the currently supplied QoS level of a dependent work unit w_i being executed at node n_i.

3.2.1 Coordinating upgrades

Given the connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with a set of cut-vertices $\mathcal{C}$ and an end-user node n_u receiving the final outcome of the coalition's processing of service S, whenever a node $n_i \in \mathcal{V}$ is able to upgrade the output of its work unit $w_i \in S$ to a QoS level Q'_{val}, the other nodes in the coalition respond to this upgrade request according to Algorithm 2.

Algorithm 2 Coordinating upgrades

$temp := n_i$
$U := \emptyset$
for each $n_c \in \mathcal{C} \cup \{n_u\}$ **do**
  if $upgrade(temp, n_c, \mathcal{G}, Q'_{val}) = (\textbf{TRUE}, T)$ **then**
    $temp := n_c$
    $U = U \cup T$
  else
    $U = \emptyset$
    return
  end if
end for
for each $(n_i, Q_{val}) \in U$ **do**
  Set the new QoS level Q'_{val} for work unit $w_i \in S$
end for

Definition 3.6 *Given the connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with a set of cut-vertices $\mathcal{C}$ and the subgraph that connects node n_i to next cut-vertex $n_c \in \mathcal{C}$, the function $upgrade(n_i, n_c, \mathcal{G}, Q'_{val})$ is defined by:*

upgrade$(n_i, n_c, \mathcal{G}, Q'_{val}) =$
  $T := \{(n_i, w_i, Q'_{val})\}$
  for* each** $n_j \in n\_paths(n_i, n_c) \setminus \{n_i\}$ ***do
    $S := update(get\_input\_qos(n_i, w_i), T)$
    if $test\_feasibility(n_j, w_j, Q'_{val}, S) =$ ***TRUE then***
      $T := T \cup \{(n_j, Q'_{val})\}$
    end if
  end for
  $S := update(get\_input\_qos(n_c, w_c), T)$
  if $test\_feasibility(n_c, w_c, Q'_{val}, S) =$ ***TRUE then***
    return (TRUE, T***)***
  end if
  return (FALSE, $\emptyset$***)***

Lemma 3.2 *Given the connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ such that $n_i \in \mathcal{V}$ and $n_j \in \mathcal{V}$ and a QoS level value Q'_{val}, the call to $upgrade(n_i, n_j, \mathcal{G}, Q'_{val})$ terminates and returns true if n_j is able to output a new QoS level Q'_{val} or false otherwise.*

Proof 3.2 *Since $\mathcal{V}$ is a finite set and since by Proposition 3.1 n_paths terminates and by Proposition 3.2 update terminates, the number of iterations is finite due to the finite number of elements in the paths. Thus, upgrade terminates.*

For any element in the paths between n_i and n_j, the new required QoS level Q'_{val} is tested and, by Lemma 3.1, the upgrade is possible if and only if the new local set of SLAs is feasible. After considering all nodes in the paths, the upgrade function returns true and the set of nodes able to upgrade, if node n_j is able to upgrade to Q'_{val}, or false otherwise. Thus, the result follows by induction on the length of the set of elements in the paths between n_i and n_j.

□

Theorem 3.1 (Correctness of Upgrade) *Given the connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ representing the QoS interdependencies of a service$S$ being executed by a coalition of nodes, such that $n_u \in \mathcal{V}$ is the end-user node receiving the service at a QoS level Q_{val}, whenever a node n_i announces an upgrade to Q'_{val}, Algorithm 2 changes the set of SLAs at nodes in $\mathcal{G}$ such that n_d receives S upgraded to the QoS level Q'_{val} or does not change the set of local SLAs at any node and n_u continues to receive S at its current QoS level Q_{val}.*

Proof 3.3 *Termination comes from the finite number of elements in $\mathcal{C} \cup n_u$ and from Lemma 3.2.*

Algorithm 2 applies the function $upgrade$ iteratively to all nodes in the subgraph starting with n_i and finishing in n_u. The base case is when there are no cut-vertices and there is only one call to $upgrade$. It is trivial to see that the result of $upgrade$ will consist in true and a set of nodes that will upgrade for the new QoS level Q'_{val} or false and an empty set and, by Lemma 3.2, it is correct. The remaining cases happen when there are one or more cut-vertices between n_i and n_u. Here, $upgrade$ will be applied to all subgraphs starting in n_i and finishing in n_d. Each of these subgraphs are sequentially tested and only if all of them can be upgraded the service S will be delivered to node n_u at the new upgraded QoS level Q'_{val}. The result follows by induction in the number of cut-vertices.

3.2.2 Coordinating downgrades

Algorithm 3 Coordinating downgrades

1: $temp := n_i$
2: **for each** $n_c \in \mathcal{C} \cup \{n_u\}$ **do**
3: **if** $downgrade(temp, n_c, \mathcal{G}, Q'_{val}) =$ **FALSE then**
4: $temp := n_c$
5: **else**
6: Downgrade was compensated and n_c continues to output Q_{val}
7: **return**
8: **end if**
9: **end for**

Given the connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with a set of cut-vertices $\mathcal{C}$ and an end-user node n_u receiving the final outcome of the coalition's processing of service S, whenever a node $n_i \in \mathcal{V}$ needs to downgrade the quality of the output of a work unit $w_i \in S$ from its current QoS level of Q_{val} to a downgraded QoS level Q'_{val}, the other nodes in the coalition respond to this downgrade request according to Algorithm 3.

Definition 3.7 *Given the connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with a set of cut-vertices $\mathcal{C}$ and the subgraph that connects node n_i to next cut-vertex $n_c \in \mathcal{C}$, the function $downgrade(n_i, n_c, \mathcal{G}, Q'_{val})$ is defined by:*

downgrade$(n_i, n_c, \mathcal{G}, Q'_{val}) =$
  $T := \{(n_i, Q'_{val})\}$
  for* each** $n_j \in n\_paths(n_i, n_c) \setminus \{n_i\}$ ***do
    $D := update(get\_input\_qos(n_j, w_j), T)$
    if $test\_feasibility(n_j, w_j, Q_{val}, D) =$ ***TRUE then***
      $T := T \cup \{(n_j, Q_{val})\}$
    else
      $set\_qos\_level(n_j, w_j, Q'_{val})$
    end if
  end for
  $D := update(get\_input\_qos(n_c, w_c), T)$
  if $test\_feasibility(n_c, w_c, Q_{val}, D) =$ ***TRUE then***
    return TRUE
  else
    for* each** $n_j \in n\_paths(n_i, n_c) \setminus \{n_i\}$ ***do
      $set\_qos\_level(n_j, w_j, Q'_{val})$
    end for
    return FALSE
  end if

Lemma 3.3 *Given the connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ such that $n_i \in \mathcal{V}$ and $n_j \in \mathcal{V}$ and n_j currently outputs a QoS level Q_{val}, the call to $downgrade(n_i, n_j, \mathcal{G}, Q'_{val})$ terminates and returns true if n_j is able to keep its current output level Q_{val} or false otherwise.*

Proof 3.4 *Since $\mathcal{V}$ is a finite set and since, by Proposition 3.1, n_paths terminates and by Proposition 3.2 $update$ terminates, the number of iterations is finite due to the finite number of elements in the paths. Thus, $downgrade$ terminates.*

For any element in the paths between n_i and n_j, it is tested if the node, given its new set of inputs, can continue to output its current QoS level Q_{val}. After considering all nodes in the paths, the $downgrade$ function returns true, if node n_j is able to continue to output Q_{val}, or sets all the previous nodes in the paths to the downgraded QoS level Q'_{val} and returns false. Again the result follows by induction on the length of the set of elements in the paths between n_i and n_j.

□

Theorem 3.2 (Correctness of Downgrade) *Given the connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ representing the QoS inter-dependencies of a service S being executed by a coalition of nodes such that $n_u \in \mathcal{V}$ is the end-user node receiving S at the QoS level Q_{val}, whenever a node n_i is forced to downgrade the quality of the output of a work unit $w_i \in S$ from its current QoS level of Q_{val} to a degraded QoS level Q'_{val}, Algorithm 3 changes the set of SLAs at nodes in $\mathcal{G}$ such that n_u continues to receive S at its current QoS level Q_{val} or sets all nodes to a degraded QoS level of Q'_{val}.*

Proof 3.5 *Termination comes from the finite number of elements in $\mathcal{C} \cup n_u$ and from Lemma 3.3.*

The correctness trivially follows by the correctness of Lemma 3.3 and by induction on the number of elements in $\mathcal{C} \cup \{n_u\}$.

3.3 Number of exchanged messages

In the previous sections we presented the formalisation of the two main coordination operations, namely upgrades and downgrades of the currently supplied QoS level Q_{val} for a service S, as a reaction to a change in the quality of inter-dependent inputs sent by adjacent nodes. In this section we analyse the number of exchanged messages in such coordination operations. We start with some auxiliary definitions.

Definition 3.8 *Given a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the in-degree of a node $n_i \in \mathcal{V}$ is the number of edges that have n_i as their destination.*

Definition 3.9 *Given a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the out-degree of a node $n_i \in \mathcal{V}$ is the number of edges that have n_i as their starting node.*

Whenever an upgrade to a new QoS level Q'_{val} is requested by a node n_i, if the next cut-vertex n_c in the graph $\mathcal{G}$ on QoS inter-dependencies cannot supply the requested upgrade, then all the precedent nodes between n_i and n_c are kept in their currently supplied QoS level Q_{val}. Thus, the number of needed messages is given by the number of edges in the paths between the n_i and n_c where it was determined that the requested upgrade was not possible. On the other hand, if the upgrade is possible, the number of needed messages is twice the number of edges between n_i and the end-user node n_u. This is because an upgrade is only possible after all the involved nodes are queried and the conjunction of their efforts results in a upgraded QoS level being delivered to n_u.

Whenever, due to resource limitations, a node n_i announces a downgrade to Q'_{val}, the next nodes in the subgraph from n_i to the next cut-vertex n_c try to compensate

the downgraded input quality in order to keep outputting the previous QoS level Q_{val}. When the cut-vertex n_c is reached two scenarios may occur. In the first one, the cut-vertex cannot compensate the degradation, although some of its precedent nodes may. In this case, all the precedent nodes are informed that they can downgrade their current QoS level to Q'_{val} since their compensation effort is useless. Note that, in the worst case, this can be propagated until the final node n_u is reached and all the coalition members will downgrade their current QoS level. As such, in the worst case, a message is sent from each node to its adjacent ones and a reply is received, which demands a total number of messages of two times the number of edges between n_i and n_u. On the other hand, in the second possible scenario, some cut-vertex n_k may be able to compensate the downgraded input quality and continue to produce the current QoS level Q_{val}. In this case, the coordination process is restricted to the subgraph between n_i and n_k. As such, coordination messages are exchanged in this subgraph only.

Thus, in the worst case, the maximum number of exchanged messages in a coordination operation is given by Equation 1.

$$\sum_{n \in \mathcal{V}} (out\_degree(n) + in\_degree(n)) \quad (1)$$

4 Evaluation

An application that captures, compresses and transmits frames of video to end users, which may use a diversity of end devices and have different sets of QoS preferences, was used to evaluate the behaviour of the proposed decentralised coordination model, with a special attention being devoted to introduce a high variability in the characteristics of the considered scenarios. The application is composed by a set of source units to collect the data, a compression unit to gather and compress the data sent from multiple sources, a transmission unit to transmit the data over the network, a decompression unit to convert the data into the user's specified format, and an user unit to display the data in the end device [16].

The number of simultaneous nodes in the system randomly varied from 10 to 100. Each node was running a prototype implementation of the CooperatES framework [17], with a fixed set of mappings between requested QoS levels and resource requirements. The code bases needed to execute each of the application's units was loaded a priori in all the nodes. The characteristics of end devices and their more powerful neighbour nodes was randomly generated, creating a distributed heterogeneous environment. This non-equal partition of resources affected the ability of some nodes to singly execute some of the application's units and has driven nodes to a coalition formation for a cooperative service execution.

At randomly selected end devices, new service requests from 5 to 20 simultaneous users were randomly generated, dynamically generating different amounts of load and resource availability during the previously accepted services' execution. Each service request was formulated as a set of random QoS levels, expressing the spectrum of the user's acceptable QoS levels in a qualitative way, ranging from a randomly generated desired QoS level to a randomly generated maximum tolerable service degradation. The relative decreasing order of importance imposed in dimensions, attributes and values was also randomly generated.

Based on each user's service request, coalitions of 4 to 20 nodes were formed using the anytime coalition formation and service proposal formulation algorithms proposed in [16, 15]. Each node was connect at least to another node in the coalition. The maximum degree of each node, that is, the number of connections to a node was set to 3. After the coalition was formed, a randomly percentage of the connections among its members was selected as a QoS dependency among those work units.

The behaviour of the proposed decentralised one-step coordination model in highly dynamic scenarios was compared to a classic centralised optimal coordination model. With a centralised coordination model, all changes in the output quality of a work unit $w_{ij} \in S_i$ have to be communicated to a single entity with service-wide knowledge. Then, this central coordinator has to determine the impact of those changes in the overall coalition's QoS level and request the adaptation of the involved nodes. To evaluate the success or failure of such dependent adaptation, an adaptation request must be sequentially made along the dependency graph either until a common global service solution is found or one of the coalition member is unable to supply the new desired QoS values.

The conducted evaluation started by comparing the total number of messages that had to be exchanged among nodes when using both approaches to globally coordinate dependent autonomous self-adaptations. The average results of all simulation runs for different coalition sizes are plotted in Figure 1.

As expected, both coordination approaches require more messages to be exchanged among nodes as the complexity of the service's topology increases. Nevertheless, the proposed decentralised coordination model requires around 80% of the needed number of messages required by the centralised model until all the affected coalition members become aware of the coordination request's result.

Less messages should result in a faster convergence to a global common solution. To verify the veracity of such assumption a second study measured the needed average time from the moment a node issued a coordination request until the outcome of the global adaptation process was deter-

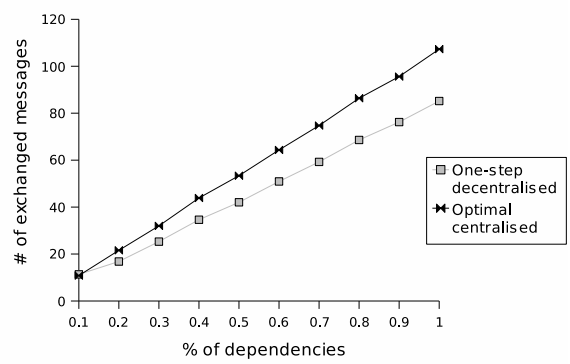

Figure 1. Number of exchanged messages

mined. The deadline used for the anytime local QoS adaptation at each node was set to one second. At the end of the algorithm's execution, the feasibility or unfeasibility of the received coordination request was determined by the node. The obtained results, on an Intel Core Duo T5500 at 1.66 GHz with 2 GB of RAM, are plotted in Figure 2.

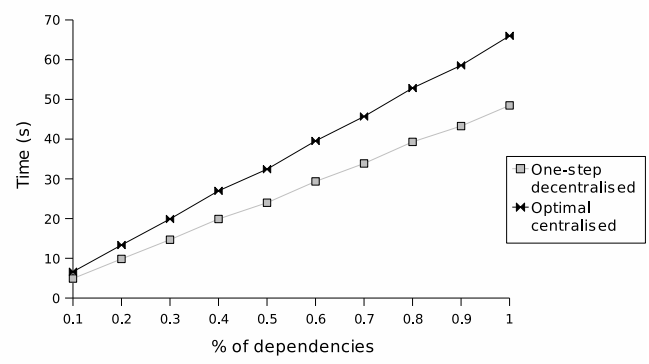

Figure 2. Time for a global adaptation

Clearly, the proposed decentralised coordination model is faster to determine the overall coordination result in all the evaluated services' topologies, needing approximately 75% of the time spent by the centralised optimal model.

Even if the proposed one-step decentralised model requires less messages and is faster than a centralised optimal model to determine a global solution it is still important to evaluate impact of an one-step coordination model on the achieved service solution's quality. Recall that, when adopting the proposed one-step coordination algorithm, if some other dependent node in the coalition is unable to supply the new requested values no other alternative solution is tried and the global adaptation process fails. On the other hand, with the centralised optimal coordination model, a node is able to reply with a service counter-proposal whenever it is unable to coordinate with the currently requested values. As such, it is possible that after some iterations, the node's best possible service solution can be accepted by all the dependent coalition partners as part of a global SLA. Note that such intermediate service solution would not be achieved with the proposed one-step coordination model.

The results were plotted, in Figure 3, by averaging the results over several independent runs of the simulation, divided in two categories: (i) when the average amount of available resources per node is greater than the average amount of resources demanded by the services being executed; and (ii) when the average amount of resources per node is smaller than the average amount of demanded resources.

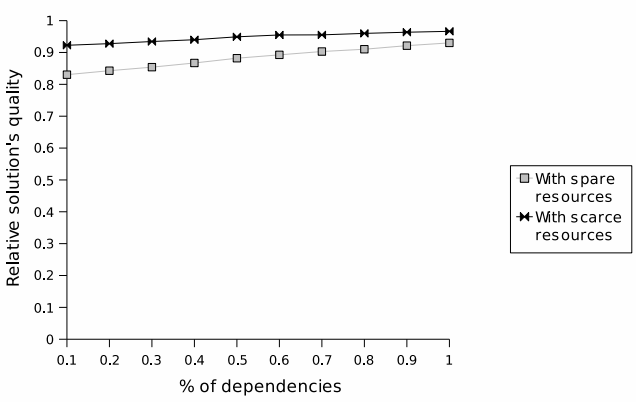

Figure 3. Relative solution's utility

As the coalition's topology complexity increases it is clearly noticeable, in both scenarios, that a near-optimal service solution's quality is achieved when using the one-step coordination model, despite its simpler approach and faster convergence to a common solution. The achieved results can be explained by the fact that as the coalition's topology complexity increases it also increases the probability of one of the involved nodes in the global adaptation process to be unable to use more than its current level of reserved resources for a work unit $w_i \in S$. As the achieved results clearly demonstrate, such probability is even greater when the resources are scarce.

5 Conclusion

This paper addressed the problem of coordinating autonomous dependent adaptations of resource-bounded nodes in dynamic open cooperative real-time environments. The proposed one-step decentralised coordination model imposes restrictions on the way members of dynamically created coalitions can self-adapt in response to changes in

the environment. It is based on an effective feedback mechanism that reduces the complexity of the needed interactions among nodes. Feedback is formulated as a result of a QoS adaptation process that evaluates the feasibility of the new requested service solution. As the achieved results clearly demonstrate, the proposed coordination model has a reduced overhead and enables a faster convergence to a near-optimal global service solution.

Acknowledgements

This work was supported by FCT through the CISTER Research Unit - FCT UI 608 and the research projects CooperatES - PTDC/EIA/71624/2006 and RESCUE - PTDC/EIA/65862/2006.

References

[1] G. Allen, T. Dramlitsch, I. Foster, N. T. Karonis, M. Ripeanu, E. Seidel, and B. Toonen. Supporting efficient execution in heterogeneous distributed computing environments with cactus and globus. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing*, pages 52–52, November 2001.

[2] C. Boutilier, R. Das, J. O. Kephart, G. Tesauro, and W. E. Walsh. Cooperative negotiation in autonomic systems using incremental utility elicitation. In *In Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence*, pages 89–97, Acapulco,Mexico, August 2003.

[3] P. Bridges, W.-K. Chen, M. Hiltunen, and R. Schlichting. Supporting coordinated adaptation in networked systems. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, page 162, Oberbayern, Germany, May 2001.

[4] T. De Wolf and T. Holvoet. Towards autonomic computing: agent-based modelling, dynamical systems analysis, and decentralised control. *Proceedings of the IEEE International Conference on Industrial Informatics*, pages 470–479, August 2003.

[5] M. Dorigo and G. D. Caro. The ant colony optimization meta-heuristic. *New ideas in optimization*, pages 11–32, 1999.

[6] J. Dowling and S. Haridi. *Decentralized Reinforcement Learning for the Online Optimization of Distributed Systems*, chapter in Reinforcement Learning: Theory and Applications, pages 142–167. I-Tech Education and Publishing, Vienna, Austria, 2008.

[7] I. Dusparic and V. Cahill. Research issues in multiple policy optimization using collaborative reinforcement learning. In *Proceedings of the 2007 International Workshop on Software Engineering for Adaptive and Self-Managing Systems*, page 18, Washington, DC, USA, 2007. IEEE Computer Society.

[8] B. Ensink and V. Adve. Coordinating adaptations in distributed systems. In *Proceedings of the 24th International Conference on Distributed Computing Systems*, pages 446–455, Tokyo, Japan, March 2004.

[9] D. Gelernter and N. Carriero. Coordination languages and their significance. *Communications of the ACM*, 35(2):96–107, 1992.

[10] S. Graupner, A. Andrzejak, V. Kotov, and H. Trinks. Adaptive control overlay for service management. In *First Workshop on the Design of Self-Managing Systems*, San Francisco, USA, June 2003.

[11] M. Jelasity, A. Montresor, and O. Babaoglu. A modular paradigm for building self-organizing peer-to-peer applications. In *In Engineering Self-Organising Systems, G. Di Marzo Serugendo*, pages 265–282. Springer, 2004.

[12] K. Kwiat and S. Ren. A coordination model for improving software system attack-tolerance and survivability in open hostile environments. In *Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*, pages 394–402, Taichung, Tawain, June 2006.

[13] A. Montresor, H. Meling, and Ö. Babaoglu. Toward self-organizing, self-repairing and resilient distributed systems. In *Future Directions in Distributed Computing*, pages 119–126, 2003.

[14] L. Nogueira and L. M. Pinho. Dynamic qos-aware coalition formation. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, page 135, Denver, Colorado, April 2005.

[15] L. Nogueira and L. M. Pinho. Dynamic qos adaptation of inter-dependent task sets in cooperative embedded systems. In *Proceedings of the 2nd ACM International Conference on Autonomic Computing and Communication Systems*, page 97, Turin,Italy, September 2008.

[16] L. Nogueira and L. M. Pinho. Time-bounded distributed qos-aware service configuration in heterogeneous cooperative environments. *Journal of Parallel and Distributed Computing*, 69(6):491–507, June 2009.

[17] L. M. Pinho, L. Nogueira, and R. Barbosa. An ada framework for qos-aware applications. In *Proceedings of the 10th Ada-Europe International Conference on Reliable Software Technologies*, pages 25–38, York, UK, June 2005.

[18] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. A resource allocation model for qos management. In *Proceedings of the 18th IEEE Real-Time Systems Symposium*, page 298. IEEE Computer Society, 1997.

[19] S. Ren, L. Shen, and J. Tsai. Reconfigurable coordination model for dynamic autonomous real-time systems. In *Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*, pages 60–67, Taichung, Tawain, June 2006.

[20] G. D. M. Serugendo. *Autonomous Systems with Emergent Behaviour*, chapter Handbook of Research on Nature Inspired Computing for Economy and Management, pages 429–443. Idea Group, Inc., Hershey-PA, USA, September 2006.

[21] M. Shankar, M. de Miguel, and J. W. S. Liu. An end-to-end qos management architecture. In *Proceedings of the 5th IEEE Real-Time Technology and Applications Symposium*, pages 176–191, Washington, DC, USA, 1999. IEEE Computer Society.

[22] C. S. D. The University of Arizona. The cactus project. Available at `http://www.cs.arizona.edu/projects/cactus/`.