IPP HURRAY!

www.hurray.isep.ipp.pt

Technical Report

Time-bounded Distributed QoS-Aware Service Configuration in Heterogeneous Cooperative Environments

Luis Miguel Nogueira

Luis Miguel Pinho

Time-bounded Distributed QoS-Aware Service Configuration in Heterogeneous Cooperative Environments

Luis Miguel Nogueira

Luis Miguel Pinho

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:

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

Abstract

The scarcity and diversity of resources among the devices of heterogeneous computing environments may affect their ability to execute services within the users' requested Quality of Service levels, particularly in open real-time environments where the characteristics of the computational load cannot always be predicted in advance but, nevertheless, response to events still has to be provided within precise timing constraints in order to guarantee a desired level of performance.

This paper proposes a cooperative service execution, allowing resource constrained devices to collectively execute services with their more powerful neighbours, meeting non-functional requirements that otherwise would not be met by an individual execution. Nodes dynamically group themselves into a new coalition, allocating resources to each new service and establishing an initial service configuration which maximises the satisfaction of the QoS constraints associated with the new service and minimises the impact on the global QoS caused by the new service's arrival.

However, the increased complexity of open real-time environments may prevent the possibility of computing optimal local and global resource allocations within a useful and bounded time. As such, the QoS optimisation problem is here reformulated as a heuristic-based anytime optimisation problem that can be interrupted at any time and quickly respond to environmental changes. Extensive simulations demonstrate that the proposed anytime algorithms are able to quickly find a good initial service solution and effectively optimise the rate at which the quality of the current solution improves at each iteration of the algorithms, with an overhead that can be considered negligible when compared against the introduced benefits.

Time-bounded Distributed QoS-Aware Service Configuration in Heterogeneous Cooperative Environments

Luís Nogueira * , Luís Miguel Pinho

School of Engineering of the Polytechnic Institute of Porto (ISEP/IPP)
Rua Dr. António Bernardino de Almeida 431
4200-072 Porto, Portugal
Phone: +351 22 8340529 Fax: +351 228340525

Abstract

The scarcity and diversity of resources among the devices of heterogeneous computing environments may affect their ability to execute services within the users' requested Quality of Service levels, particularly in open real-time environments where the characteristics of the computational load cannot always be predicted in advance but, nevertheless, response to events still has to be provided within precise timing constraints in order to guarantee a desired level of performance.

This paper proposes a cooperative service execution, allowing resource constrained devices to collectively execute services with their more powerful neighbours, meeting non-functional requirements that otherwise would not be met by an individual execution. Nodes dynamically group themselves into a new coalition, allocating resources to each new service and establishing an initial service configuration which maximises the satisfaction of the QoS constraints associated with the new service and minimises the impact on the global QoS caused by the new service's arrival.

However, the increased complexity of open real-time environments may prevent the possibility of computing optimal local and global resource allocations within a useful and bounded time. As such, the QoS optimisation problem is here reformulated as a heuristic-based anytime optimisation problem that can be interrupted at any time and quickly respond to environmental changes. Extensive simulations demonstrate that the proposed anytime algorithms are able to quickly find a good initial service solution and effectively optimise the rate at which the quality of the current solution improves at each iteration of the algorithms, with an overhead that can be considered negligible when compared against the introduced benefits.

Key words: Distributed systems, Dynamic real-time systems, Quality of Service, Anytime algorithms

1 Introduction

Real-time computing systems were originally developed to support safety critical, mission critical, or business critical control applications characterised by stringent timing constraints and, indeed, much of real-time computing is still for these types of applications. In these systems, missing a single deadline can jeopardise the entire system behaviour or even cause catastrophic consequences. Hence, they need to be designed under worst-case assumptions and executed with predictable kernel mechanisms to meet the required performance in all anticipated scenarios.

While the high cost of such approach is acceptable for applications with dramatic failure consequences, it is no longer justified in a growing number of new real-time systems in areas such as multimedia, automotive information and entertainment systems, mobile phone networks, robotics, and radar tracking. A deadline miss does not constitute a system or application failure but it is only less satisfactory for the user and, as such, the approach is generally regarded as cheaper and more flexible. In fact, in order to satisfy a set of constraints related to weight, space, and energy consumption, these systems are typically built using small microprocessors with low processing power and limited resources.

The challenge is how to efficiently execute applications in these new embedded real-time systems while meeting non-functional requirements, such as timeliness, robustness, dependability, performance etc. This is where QoS management applies. QoS-aware applications have an important property, they can perform at degraded levels and still satisfy the users to a certain degree. This property is quite different from traditional applications which either either at a given quality level or not at all.

Reserving resources is basic for supporting QoS mechanisms. An application cannot provide stable QoS characteristics if it has not some guarantees on the available amount of resources. The operating system or middleware reserves a portion of the system's resources to an application, which then has to provide a predefined stable output quality.

However, the move from the traditional self-enclosed real-time system to open real-time systems is also one of moving from static to dynamic environments [1; 2]. Open real-time systems allow a mix of independently developed real-time and non real-time applications to coexist in the same system. As such, the set of applications to be executed and their aggregate resource and timing

---

* Corresponding author.

  Email addresses: `luis@dei.isep.ipp.pt` (Luís Nogueira),
`lpinho@dei.isep.ipp.pt` (Luís Miguel Pinho).

requirements are unknown until runtime, implying that accurate optimisation models are then difficult to obtain and quickly become outdated. The solution is to make these new real-time systems adaptable to the environment, thus capable to react to changes in the operating conditions by acting on the applications' and system's parameters.

Nevertheless, an increasing number of real-time applications need a considerable amount of computation power and are pushing the limits of traditional data processing infrastructures [3]. Consider, for example, the real-time stream processing systems described in [4; 5; 6]. The quantity of data produced by a variety of data sources and sent to end systems to further processing is growing significantly, increasingly demanding more processing power and the challenges become even more critical when a coordinated content analysis of data sent from multiple sources is necessary [5]. Thus, with a potentially unbounded amount of stream data and limited resources, some of the processing tasks may not be satisfyingly answered even within the users' minimum acceptable QoS levels [6].

In this context, a cooperative QoS-aware execution of resource intensive services among neighbour nodes seems a promising solution to address these increasingly complex demands on resources and desirable performance. The CooperatES (Cooperative Embedded Systems) framework [7; 8; 9] facilitates the cooperation among neighbours when a particular set of QoS constraints cannot be satisfyingly answered by a single node. Nodes dynamically group themselves into a new coalition, allocating resources to each new service and establishing an initial Service Level Agreement (SLA). The proposed SLA maximises the satisfaction of the QoS constraints associated with the new service and minimises the impact on the global QoS caused by the new service's arrival [7; 8].

However, the increased complexity of dynamic open scenarios may prevent the possibility of computing optimal local and global resource allocations within a useful and bounded time, as the optimal level of deliberation varies from situation to situation. This is true for many real-time applications, where it may be preferable to have approximate results of a poorer but acceptable quality delivered on time to late results with the desirable optimal quality. For example, it is better for a collision avoidance system to issue a timely warning together with an estimated location of the obstacle than a late description of the exact evasive action. Another example is video and sound processing. While poorer quality images and voices on a timely basis may be acceptable, late frames and long periods of silence often are not. Other examples can be found in route optimisation of automated vehicles [10; 11], computer games [12], and real-time control [13].

It is therefore beneficial to build systems that can tradeoff the computational

3

cost for the quality of the achieved solution [14]. This paper reformulates the distributed resource allocation problem as an anytime optimisation problem in which there are a range of acceptable solutions with varying qualities, adapting the distributed service allocation is to the available deliberation time that is dynamically imposed as a result of emerging environmental conditions [9].

Nodes start by negotiating partial, acceptable service proposals that are latter refined if time permits, in contrast to a traditional QoS optimisation approach that either runs to completion or is not able to provide a useful solution. At each iteration, the proposed QoS optimisation tries to find a new feasible set of QoS levels with an increasing utility. This improvement is larger at the early stages of computation and diminishes over time.

Thanks to the anytime nature of the proposed approach, it is possible to interrupt the QoS optimisation process at any point in its execution and still be able to obtain a service solution and a measure of its quality, which is expected to improve as the run time of the algorithms increases. The binary notion of correctness associated with traditional QoS optimisation algorithms is then replaced by a set of quality measured outputs. For the remaining of this paper, the solution's quality measure indicates how close is the offered QoS level to the user's desired QoS level.

The rest of this paper is organised as follows. The next section analyses related work. Section 3 describes our model and used notation, followed by a detailed description of the structure of the proposed framework for a QoS-aware cooperative execution of resource intensive services in Section 4. Section 5 presents a generic QoS description scheme that guarantees information consistency and compatibility in a community of distributed heterogeneous nodes. In Sections 6 and 7, the proposed anytime approach for a cooperative service execution's configuration that maximises the user's satisfaction with provided service is described and validated in detail. Section 8 presents the results of extensive simulations conducted with the main objectives of analysing the performance of the proposed anytime approach and comparing it against the traditional versions of the algorithms. Finally, Section 9 concludes the paper.

2 Related work

Traditional QoS optimisation algorithms often assumed that for a given invocation of a task, the quality of provided service for a particular amount of resources was constant. While this may be sufficient for some applications, there are others (e.g. radar tracking, multimedia, etc.) where some environmental factors outside the direct control of the system affect the fixed relationship between the provided level of service and resource requirements. As

such, mechanisms for arbitration of QoS levels based on the concept of flexible resource requirements have been studied extensively either at individual nodes or distributed environments.

The concept of online admission control has been applied to resource reservation for dynamically arriving tasks and several efforts appear in the context of real-time operating systems and networks research. Relevant work can be found in [15; 16; 17; 18].

At the network level, scheduling algorithms for package deliberation provide specific quality levels [19], while resource reservation protocols such as the Resource ReSerVation Protocol (RSVP) [20] provide support for end-to-end resource reservation for specific sessions. DiffServ and IntServ [21] are examples of IETF standards that integrate RSVP for the support of real-time as well as the current non real-time service in IP networks.

The Real-Time Transport Protocol (RTP) [22] is a transport protocol for carrying real-time traffic flows in an IP network. It provides a standard packet header format which gives sequence numbering, media-specific time stamp data, source identification, and payload identification, among other things. RTP is usually carried using UDP. RTP is supplemented by the Real-Time Transfer Control Protocol (RTCP), which carries control information about the current RTP session. RTP do not address the issue of resource reservation but relies on reservation protocols such as RSVP.

However, most of this research has been focused on low-level system mechanisms. While individual resource management is an important factor for an efficient QoS management, we believe that it is not sufficient for the ultimate end-users who experience the resulting QoS. It is known that different users might tolerate different levels of service, or could be satisfied with different quality combination choices. In order to achieve the users' acceptance requirements and to satisfy the imposed constraints, a QoS-aware resource management must support QoS negotiation, admission, and reservation mechanisms in an integrated and accessible way.

Jensen et al. proposed a soft real-time scheduling technique based on application benefit [23]. Each application specified a benefit curve that indicated the relative benefit to be obtained by scheduling the application at various times with respect to its deadlines. The goal was to schedule applications so as to maximise an overall system's benefit. While this approach is intuitively very appealing, it is computationally intractable.

Nevertheless, the work of Jensen et al. led to the adoption of utility functions to represent varying satisfaction with service changes in several other works. In [24], Brand et al. propose a mediation method for resource allocation based on the maximums processor usage and users' benefit as measures for QoS levels

and present the Dynamic QoS Manager (DQM) middleware. DQM is based on the notion of applications' specified execution levels that reflect algorithmic modes in which applications can execute. It uses the execution level information and the current state of the system to dynamically determine appropriate QoS allocations for the running applications.

Curescu et al. proposed a time-aware admission control and resource allocation scheme that aims to allocate bandwidth such that the accrued utility of the whole system, accumulated over time, is maximised [25]. The approach synthesises the consequences of resource reallocation for different types of applications and use this information to perform a periodic reallocation optimisation.

In coping with the shortage of QoS support from an end-user point of view, Rajkumar et al. [26] proposed Q-RAM, a QoS-based resource allocation model in which multiple resources are allocated to maximise the overall system's utility. The static resource allocation algorithms of Q-RAM were extended to support a dynamic task traffic model [1] and to handle non-monotonic dimensions [27]. Further improvement techniques to reduce the computation complexity of the initial proposal and their application to radar tracking are described in [28].

Working from the user's perspective and maximising the user perceived quality or utility has also been addressed in several other works. In [17], the user's QoS preferences are considered for the application's runtime behaviour control and resource allocation planning. Example preferences include statements that a video-phone call should pause a movie unless it's being recorded and that video should be degraded before audio when all desired resources are not available. These are useful hints for high-level QoS control and resource planning, but are inadequate for quantitatively measuring QoS, or analytically planning and allocating resources.

Abdelzaher et al. used a similar concept to propose a QoS negotiation mechanism that ensures graceful service degradation in cases of overload, failures, or violation of pre-runtime assumptions [29]. The authors suggest that a user should be able to express his spectrum of acceptable QoS levels, as well as a quantitative perceived utility of receiving service at each of those levels. QoS levels are then statically mapped to certain quality choice combinations. A similar approach is taken in [30]. But neither of the works addresses the balancing of competing resource demands or has developed an effective specification method of QoS preferences or a mechanism to facilitate utility data acquisition.

A generic architecture for resource reservation and allocation that supports flow-specific QoS specifications as well as online monitoring and control of

both individual resources and heterogeneous resource sets was proposed by Foster et al. [31]. The architecture builds on differentiated service mechanisms to enable the coordinated management of distinct flow types, networks, CPUs, and storage systems.

Another architecture for the adaptive management of multiple resources on a general purpose operating system was proposed by Palopoli et al. in [32], extending a prior architecture dedicated exclusively to the adaptation of the CPU's bandwidth for QoS control [33].

In grid environments, a similar concept was used by Chunlin et al. in [34] to present a utility-based QoS optimisation strategy for multi-criteria scheduling. The QoS optimisation problem is split in a task optimisation subproblem performed on behalf of the user and in a resource optimisation subproblem performed on behalf of the grid.

While we certainly share some concerns with these works, and also apply utility-based adaptation strategies [7], we go a step further and propose a QoS-aware cooperative service execution to deal with a large number of tasks, multiple resources, and highly dynamic real-time operation constraints in open real-time systems.

Several studies in computation offloading propose task partition/allocation schemes that allow the computation to be offloaded, either entirely or partially, from resource constrained (wireless) devices to a more powerful neighbour [35; 36; 37]. These works conclude that the efficiency of an application execution can be improved by careful partitioning the workload between a device and a fixed neighbour. Often, the goal is to reduce the needed computation time and energy consumption [38; 39; 40; 41; 42] by monitoring different resources, predicting the cost of local execution and that of a remote one and deciding between a local or remote execution. However, most of the work in this direction is limited to the case where there is only on resource-limited device and one relatively more capable neighbour to offload computation to. Also, none of these works supports the maximisation of each user's specific QoS preferences while offloading computation.

Our work facilitates the cooperation among heterogeneous nodes whenever a particular set of QoS constraints cannot be satisfyingly answered by a single node. The resulting coalition is the one which maximises the satisfaction of the QoS constraints associated with the new service and minimises the impact on the global QoS caused by the new service's arrival.

Furthermore, 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. The fundamental prob-

lem that has to be faced is the uncertainty of the environment. In particular, when considering real-time requirements, uncertainty means that desired bounds may not be met when adapting the system to the dynamically changing environmental conditions.

This idea has been formalised using the concepts of imprecise computation and anytime algorithms. Liu et al.[43] have recognised imprecise computation (for monotone tasks), sieve functions (for non-monotone tasks) and multiple versions as the three ways by which unbounded components can be integrated into real-time systems.

Imprecise computation uses monotone functions to produce intermediate results as a task executes. The value of these results is expected to improve as the execution of the task continues. The computation required to produce a result with minimum quality forms the *mandatory* part of the task. Clearly, this mandatory part must have a worst case execution time that is guaranteed by the schedulability analysis. The rest of the task's execution is called *optional*. The optional part is (usually) an iterative refinement algorithm that progressively improves the quality of the result generated by the mandatory part. These concepts were integrated with replication and checkpoint techniques to reduce the cost of providing fault tolerance and enhanced availability [44].

Anytime algorithms [45; 46; 14] are based on the idea that the computation time needed to compute optimal solutions will typically reduce the overall utility of the system. An anytime algorithm is an iterative refinement algorithm that can be interrupted and asked to provide an answer at any time. It is expected that the quality of the answer will increase (up to some maximum quality) as the anytime algorithm is given increasing time to run, offering a tradeoff between the quality of the results and computational requirements. Associated with an anytime algorithm is a performance profile, a function that maps the time given to an anytime algorithm (and in some cases input quality) to the quality of the solution produced by that algorithm.

Open real-time environments also demand a particular attention to the dynamic scheduling of the framework's management and services' execution. Abeni and Buttazo proposed the Constant Bandwidth Server (CBS) scheduler [47] to efficiently handle soft real-time requests with a variable or unknown execution behaviour under the EDF [48] scheduling policy. To avoid unpredictable delays on hard real-time tasks, soft tasks are isolated through a bandwidth reservation mechanism, according to which each soft task gets a fraction of the CPU and it is scheduled in such a way that it will never demand more than its reserved bandwidth, independently of its actual requests. This is achieved by assigning each soft task a deadline, computed as a function of the reserved bandwidth and its actual requests. If a task requires to execute more than its expected computation time, its deadline is postponed so that its

reserved bandwidth is not exceeded. As a consequence, overruns occurring on a served task will only delay that task, without compromising the bandwidth assigned to other tasks.

The resource reservation approach of CBS has been extended [49; 50; 51; 52; 53], introducing the ability to exploit tasks' earlier completions and reclaim the resulting residual capacities to further increase resource usage and handle soft tasks' overloads more efficiently. Nevertheless, new open real-time systems can benefit from a more flexible overload control, achieved with the combination of guaranteed and best-effort servers and reducing isolation in a controlled fashion in order to donate reserved, but still unused, capacities to currently overloaded servers.

The Capacity Sharing and Stealing (CSS) scheduler [9] proposes to handle overloads with additional capacity that is available from two sources: (i) by reclaiming unused allocated capacity when jobs complete in less than their budgeted execution time; and (ii) by stealing allocated capacities to non-isolated servers used to schedule sporadic best-effort jobs. The integration of the CSS scheduler into the CooperatES framework is discussed in detail in [54].

3 Problem description and system model

We are primarily interested in dynamic scenarios where new tasks can appear while others are being executed, the processing of those tasks has associated real-time execution constraints, and service execution can be performed by a coalition of neighbour nodes. Due to these characteristics, resource availability is highly dynamic as services enter and leave the system at anytime.

Consider a distributed system with several heterogeneous nodes, each with its specific set of resources R_i. For some of those nodes there may be a constraint on the type and size of services they can execute within the users' acceptable QoS levels. Therefore, this work addresses a distributed cooperative execution of resource intensive services in order to maximise the users' satisfaction with the obtained QoS. Nodes may cooperate either because they can not deal alone with the resource allocation demands imposed by users and services or because they can reduce the associated cost by working together.

It is assumed that a service S can be executed at varying levels of QoS to achieve an efficient resource usage that constantly adapts to the devices' specific constraints, nature of executing tasks and dynamically changing system conditions. There will be a set of independent tasks to be executed, resulting from partitioning the resource intensive service S. Correct decisions on ser-

vice partitioning must be made at run time when sufficient information about workload and communication requirements become available [35], since they may change with different execution instances and users' QoS preferences.

Each service has a set of parameters that can be changed in order to configure the supplied QoS and its correspondent resource demand. Each subset of parameters that relates to a single aspect of service quality is named as a QoS dimension. For example, consider the transmission of multiple audio/video streams over a network. This scenario involves a network with a given bandwidth and nodes serving and receiving the streams. Typical audio related parameters are the sampling rate (8, 16, 24, 44, 48 kHz), the sampling bits (8, 16), and the end-to-end latency (100, 75, 50, 25 ms), while in video it is usually considered the picture dimension (SQCIF, QCIF, CIF, CIF4), colour depth (1, 3, 8, 16, ...), and frame rate (1, ..., 30). Each of these QoS dimensions has different resource requirements for each possible level of service.

Different configurations of a stream can have different utility values for different users and applications. For example, for a particular user a transmission of a music concert may place higher quality requirements on audio, although colour video may be also desirable, while another user of a remote surveillance system may require higher video quality with a minimum of gray scale images. Let Q be the set of the user's QoS constraints associated with service S. Each Q_{kj} is a finite set of quality choices for the jth attribute of dimension k. This can be either a discrete or continuous set.

Users provide a single specification of their own range of QoS preferences Q 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 tasks that make up the service. As a result, the user is able to express acceptable compromises in the desired QoS and assign utility values to QoS levels. Note that this assignment is decoupled from the process of establishing the supplied service QoS levels themselves and determining the resource requirements for each level.

Given the spectrum of the user's acceptable QoS levels Q for service S, the coalition formation problem can be described as:

> Given a set of neighbour nodes N and a resource allocation demand enforced by Q, if the resource demand cannot be satisfyingly answered by a single node, neighbour nodes should cooperate to fulfil such resource demand. The selection of a subset of nodes in N to cooperatively execute S should be influenced by either the maximisation of the QoS constraints Q associated with S and by the minimisation of the impact on the current QoS of the previously accepted services caused by the arrival of S.

Searching for an optimal resource allocation with respect to a particular goal

has always been one of the fundamental problems in QoS management. However, as the complexity of open distributed systems increases, it is also increasingly difficult to achieve an optimal resource allocation that deals with both users' and nodes' constraints within an useful and bounded time. Note that if the system adapts too late to the new resource requirements, it may not be useful and may even be disadvantageous.

Our proposal is to quickly establish an initial, sub-optimal, solution according to the set of QoS constraints that have to be satisfied. Then, if time permits, the initial solution is gradually refined until it finally reaches its optimal value or the available deliberation time expires. At each iteration, a new set of SLAs is found with an increasing utility to the user's request under negotiation but these successive adjustments get smaller as the QoS optimisation process progresses.

Please note that the paper is focused on the anytime configuration of a distributed cooperative service execution and not on the coalition's operation phase. The dynamic adaptation of the coalition's service provisioning to changes in the environment is discussed, at a local level, in [55] and in [56], at the global coalition level.

Furthermore, the proposed anytime algorithms also are completely independent from how the code to be executed on the original node's behalf arrives to the coalition members. It is, for now, assumed that only nodes equipped a priori with a service's code blocks respond to a cooperation request, thus eliminating the need to migrate code at runtime and transfer the service's current state.

4 The CooperatES framework

The objective of the CooperatES (Cooperative Embedded Systems) framework is to enable resource-constrained devices to solve computationally expensive services by redistributing parts of the service onto other devices. Such distribution is influenced by the maximisation of the QoS preferences associated with the new service requests, addressing the increasingly complex demands on performance and customisable service provisioning.

Each node has a significant degree of autonomy and it is capable of performing tasks and sharing resources with other nodes. A service can be executed by a single node or by a group of nodes, depending on the user's node capabilities and on the user's imposed quality constraints. In either case, the service is processed in a transparent way for the user, as users are not aware of the exact distribution used to solve the computationally expensive services. The

framework facilitates the tasks' distribution across a community of nodes, forming temporary coalitions for a cooperative service execution. Figure 1 presents the structure of the proposed framework, running on every node of the network.
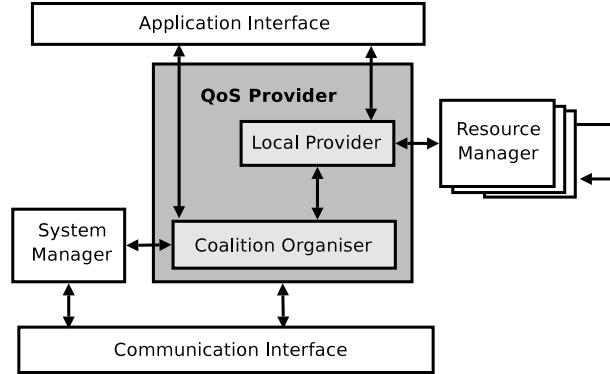

Fig. 1. Framework structure

In the proposed model, QoS-aware applications must explicitly request the service execution to the underlying QoS framework, thus providing explicit admission control, abstracting from the existing underlying distributed middleware and from the operating system. The model itself abstracts from the communication and execution environments.

Central to the behaviour of the framework is the *QoS Provider* of each node, which is responsible for processing both local and remote resource requests. Rather than reserving local resources directly, it contacts the *Resource Managers* to grant specific resource amounts to the requesting task. Note that, in this paper, it is assumed that failures of resources will not occur during services' execution but only that they may get overloaded.

Each *Resource Manager* is a module that manages a particular resource. This module interfaces with the actual implementation in a particular system of the resource controller, such as the device driver for the network, the scheduler for the CPU, or with the software that manages other resources (such as memory).

Resource managers have the ability to use each other in order to allow systems to be built supporting QoS requirements either from the point of view of the user (e.g. user-perceived high quality), of applications (e.g. video frame rate) or of the system (e.g. CPU cost). With the layering represented in Figure 2, an interactive application can be more user friendly and easier to use by providing only high-level user perceptive quality, whilst other applications can be programmed to use application-related QoS constraints.

Local and remote service requests arrive dynamically at any node and are formulated as a set of acceptable QoS levels in decreasing preference order. To guarantee the request locally, the *Local Provider* computes a set of SLAs
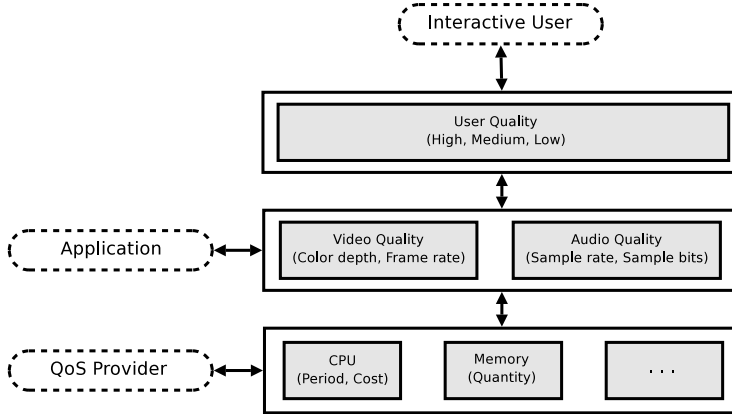
Fig. 2. Resource managers' layering

that tries to maximise the utility associated with the new service's QoS configuration as well as to minimise the impact on the current QoS of previously accepted services.

If the resource demand imposed by the user's QoS constraints cannot be locally satisfied, the *Coalition Organiser* is responsible for the coalition formation process. It broadcasts the service's description as well as the user's quality preferences, evaluates the received service proposals and decides which nodes will be part of the coalition. We consider the existence of an atomic broadcast mechanism in the system, guaranteeing that all nodes receive the same service requests and proposals in the same order.

The *System Manager* maintains the overall system configuration, detects nodes entering and leaving the network, and manages the coalition's operation and dissolution.

Note that although we consider a collaborative environment, proper resource usage must be monitored in run time [57], in order to decide based on the actual system's resource usage and not only on the resource usage assumptions of requesting services. A prototype implementation of the framework is described in [58].

5 Expressing Quality of Service

Given the heterogeneity of services to be executed, users' quality preferences, underlying operating systems, networks, devices, and the dynamics of their resource usages, QoS specification becomes an important issue in the context of a distributed QoS-aware cooperative service execution framework. However, as open distributed systems become more complex, so is the specification of requested and supplied QoS among users and service providers. Nodes must

13

either have a common understanding of how QoS should be specified, or be able to map their individual specifications into a common one.

The definition of such a generic QoS scheme must include quality dimensions, attributes and values, as well as relations that map dimensions to attributes and attributes to values. Adopting a common QoS description scheme in an open distributed environment guarantees information consistency and compatibility in a community of heterogeneous nodes. Information consistency is satisfied when each specific expression has the same meaning for every node. Information compatibility is achieved when any concept is described by the same expression, for all the nodes. Furthermore, a generic QoS scheme should also be extensible to support the later addition of new terms and relations as the system evolves. In this paper, we model each of these diverse requirements by the following structure:

$$QoS = \{Dim, Attr, Val, DAr, AVr, Deps\}$$

where Dim is the set of QoS dimensions, $Attr$ is the set of attributes identifiers and Val is the set of attribute's values identifiers.

Each value is represented by a tuple $Val_i = \{Type, Domain\}$, where $Type = \{integer, float, string\}$, and $Domain = \{continuous, discrete\}$.

The set of relationships DA_r assigns to each dimension in Dim a set of attributes in $Attr$ and is defined as $DA_r : Dim_i \rightarrow Atr, \forall_{Dim_i} \in Dim$.

The set of relationships AV_r assigns to each attribute in $Attr$ a specific value in Val and is represented as $AV_r : Atr_i \rightarrow Val_k, \forall_{Atr_i} \in Atr, \exists^1_{Val_k} \in Val$.

$Deps$ defines the set of existing dependencies among the values of the existing attributes. A dependence between Atr_i and Atr_j is represented as $Dep_{ij} = f(Val_{ki}, Val_{kj}), \forall Attr_i, Attr_j \in Attr$. Such dependency relations specify that a task offers a certain level of QoS under the condition that some specified QoS will be offered by the environment or by other tasks.

Using a video streaming application as an example, the following is a list of quality dimensions that might me associated with any particular application. The list is given to illustrate the proposed model and is not intended to be exhaustive.

```
Dim = {Video Quality, Audio Quality}
Attr = {compression index, color depth, frame size, frame rate,
        sampling rate, sample bits}
Val = {{1,integer,discrete},{3,integer,discrete},...,
      {[1,30],integer,continuous},...}
```

```
DA Video Quality = {image quality, color depth, frame size,
                    frame rate}

DA Audio Quality = {sampling rate, sample bits}

AV compression index = {[0,100]},
AV frame size = {SQCIF,QCIF,CIF,4CIF,16CIF}
AV color depth (bits) = {1,3,8,16,24,...}
AV frame rate (per second) = {[1,30]}
AV sampling rate (kHz) = {8,11,32,44,88}
AV sample bits (bits) = {4,8,16,24}
```

Having such a QoS characterisation of a particular application domain, users and service providers are now able to define service requirements and proposals in order to reach an agreement on service provisioning. Since QoS is often multi-dimensional, a user (or application) might want to make some quality tradeoff, especially when the available resources are scarce. Therefore, it is to the user's advantage to be able to specify a set of personal QoS requirements using an interface that explicitly allows the definition of quality tradeoffs.

Consider the following example. Typically, the video frame rate fluctuates as the system's load fluctuates. However, frame rate is an important QoS parameter for talk shows because it affects lip synchronisation [59]. As such, other QoS parameters like the frame size or image quality may be better candidates for degradation when the needed resources become scarce. On the other hand, in a remote video surveillance system, a grey scale, low frame rate may be sufficient, but a high image quality is important. As such, an efficient system's QoS optimisation policy must consider the specific quality requirements of each user or application.

A flexible approach to deal with the heterogeneity and load variations of dynamic open environments is to define such personal quality requirements through a utility model. Several works associate with each pre-defined QoS level a utility function that specifies the user's benefit in obtaining service within those values [29; 26; 34]. However, it may be clearly infeasible to make the user specify an absolute utility value for every pre-defined quality choice. While we want a semantically rich request in order to achieve a service provisioning closely related to the user's quality preferences, we also want the user to actually be able to express personal QoS preferences in a service request. Equally important, we believe that the system should dynamically determine promised QoS levels according to the each user's accepted QoS values for each QoS dimension and local resource availability. As such, the reward of executing a task at one of those dynamically determined QoS levels will depend on the number, and relative importance, of the QoS dimensions being served closer

to the user's desired QoS level.

Following those goals, a more natural and realistic way is to simply impose a service request based on a qualitative, not quantitative, measure. With a relative decreasing order on quality dimensions, their attributes, and accepted values, a user is able to encode the relative importance of the new service's performance at the different QoS levels without the need to quantify every quality tradeoff with absolute values. For example, a user of a remote video surveillance system can easily state that video is more important than audio, and the image's quality is more important than the obtained frame rate and colour depth with the following service request:

```
1. Video Quality
  (a) compression index : {[0,20]}
  (a) frame rate    : {[5,1]}
  (b) color depth   : {3, 1}
2. Audio Quality
  (a) sampling rate : {11,8}
  (b) sample bits   : {8,4}
```

The evaluation of the user's acceptability of each service proposal with respect to the expressed quality preferences is detailed in the next section.

6 Coalition formation

The coalition formation process should enable the selection of individual nodes that, based on their own resources and availability, will constitute the best group to satisfy the user's QoS requirements associated with a resource intensive service. By best group, we mean the group formed by those nodes which offer service closer to the user's desired QoS level.

A service request is considered to be formulated through the relative decreasing importance $(K = 1 \ldots n)$ of a set of n QoS dimensions, ranging from a desired QoS level $L_{desired}$ and the maximum tolerable service degradation, specified by a minimum acceptable QoS level $L_{minimum}$. For each dimension, a relative decreasing importance order of attributes is also specified $(i = 1 \ldots attr_k)$, where k is the number of attributes of dimension K. Please note that k and i are not the identifiers of dimensions and attributes in a domain's QoS description, but their relative position in user's service request.

When the user's node N_i cannot provide service within the user's acceptable QoS levels Q_i, the *QoS Provider* broadcasts a cooperation request to execute service S_i. The set of tasks that can be remotely executed is determined

by a task partition/allocation scheme that dynamically considers the trade-off between local execution requirements and communication costs [35]. The cooperation request includes a description of each remote task T_i, the user's QoS constraints, and a timeout Δ_t for the reception of service proposals.

Every neighbour node N_j which is able to satisfy the request, formulates a service proposal according to a local QoS optimisation algorithm (see Section 7 for details), and replies to node N_i with both its service proposal P_{ji} and its local reward R_j, resulting from its proposal acceptance. For now, it suffices to say that the local reward is an indicator of the node's local QoS optimisation level, according to the set of services being locally executed and their QoS constraints. How each node measures its local reward will be detailed in Section 7.

It is clear that different groups of nodes will have different degrees of efficiency in the service's cooperative execution performance due to different capabilities of their members and their current state. As such, the coalition's members selection must be determined by the proximity of their service proposals with respect to the expressed user's multi-dimensional QoS constraints. Each admissible proposal[1] P_i is then evaluated by determining, for each QoS dimension, a weighted sum of the differences between the user's preferred values and the values proposed in P_{ji}, using Eq. 1. Recall that the user's QoS constraints are presented in a decreasing preference order.

$$distance(P_i) = \sum_{k=1}^{n} w_k * dif(Q_k) \qquad (1)$$

where n is the number of QoS dimensions and $0 \leq w_k \leq 1$ is the relative importance of QoS dimension k, Q_k, to the user, and can defined as

$$w_k = \frac{n - k + 1}{n} \qquad (2)$$

The degree of acceptability of each proposed attribute's value when compared to the request one is given by Eq. 3, considering continuous and discrete domains.

$$dif(Q_k) = \sum_{i=1}^{attr_k} w_i * |da(Prop_{ki}, Pref_{ki})| \qquad (3)$$

where $attr_k$ is the number of attributes in dimension k.

---

[1] A proposal is admissible if it can satisfy all QoS dimensions within the user's acceptable QoS levels

In Eq. 3, the function $da(Prop_{ki}, Pref_{ki})$ quantifies, for an attribute i, the degree of acceptability of the proposed value $Prop_{ki}$, when compared to the user's preferred value $Pref_{ki}$ and is defined as

$$da = \begin{cases} \dfrac{Prop_{ki} - Pref_{ki}}{max(Q_k) - min(Q_k)} & \text{if continuous } Q_{ki} \\[2em] \dfrac{pos(Prop_{ki}) - pos(Pref_{ki})}{length(Q_k) - 1} & \text{if discrete } Q_{ki} \end{cases} \quad (4)$$

If attribute i has a continuous domain, this quantification is a normalised difference between the proposed value and the preferred one. Examples of QoS attributes with continuous domains include video frame rate, frame size, and network bandwidth.

For discrete domains, Eq. 4 considers the preferences attached to $Prop_{ki}$ and $Pref_{ki}$ by using their relative position in the application's QoS requirements specification. Examples of discrete QoS attributes include the audio and video containers and the data encryption algorithm.

In [60] the authors use the notion of a *Quality Index*, defining a bijective function that maps the elements of a discrete domain into integer values. We use a similar approach, by mapping the position (index) of that attribute in the domain's specification into $Prop_{ki}$'s and $Pref_{ki}$'s scoring values. When the domain's QoS description defines the possible values for some attribute of a QoS dimension Q_k by a set of intervals, Q_k in Eq. 4 must relate to the particular interval where $Prop_{ki}$ is found. In a similar fashion, if the user expresses his set of acceptable values for an attribute of dimension Q_k considering a set of intervals, $Pref_{ki}$ should be the first value on the $Prop_{ki}$'s interval and the relative decreasing order of importance of that interval to the user must be considered.

The best proposal for each of the service's tasks is thus the one that presents the lowest distance to the user's quality preferences in all QoS dimensions. However, rather than assuming that this coalition formation process can have all the time it needs to compute its optimal output, we propose to achieve a time-bounded distributed QoS-aware service configuration among a set of heterogeneous neighbours.

6.1 Anytime global QoS optimisation

The participants in the anytime coalition formation process will be the user's node and the subset of neighbour nodes able to offer service within the user's

required QoS levels. The user's node, playing the role of *organiser*, starts and guides all the negotiation process, broadcasting the service's requirements and user's QoS constraints and evaluating the received service proposals, sent by the *respondent* neighbours.

In order to be useful in practice, an anytime approach must try to quickly find a sufficiently good initial solution and gradually maximise its improvement at each iteration, it if time permits. As such, a particular attention must be devoted to the method used to select the next proposal to be evaluated from the set of received proposals, rather than depending on the order of proposals' reception. The proposed anytime coalition formation algorithm, described in detail in Algorithm 1, uses each node's local reward as a a heuristic to guide the coalition formation process. Clearly, nodes with a higher local reward have a higher probability to be offering service closer to this particular user's request under negotiation since the utility achieved by all services being locally executed is higher. Then, for each remote task $T_i \in S$, the next candidate proposal P_{ki} to be selected from the set of received proposals P_i is the one sent by the node N_k with the greatest local reward R_k.

$$P_{ki} | P_{ki} \in P_i, max(R_k) \tag{5}$$

The proposed Algorithm 1 allows the global QoS optimisation process to return many possible approximate answers for a given input of service proposals to be evaluated. It can be interrupted at any time, providing a solution and a measure of its quality, which is expected to improve as the run time of the algorithm increases.

Algorithm 1 Iterative coalition formation
---
  while $t < Maximum\ execution\ time$ **do**
    for each $T_i \in S$ **do**
      Select next candidate proposal P_{Ki}, maximising local reward
      $E_{P_{ki}} = distance(P_{ki})$
      if $Best_{P_i} - E_{P_{ki}} > \alpha$ **then**
        $Best_{P_i} = E_{P_{ki}}$
        Update coalition with N_k for task T_i
      else if $0 < Best_{P_i} - E_{P_{ki}} < \alpha$ and $R_{P_{ki}} > R_{Best_{P_i}}$ **then**
        $Best_{P_i} = E_{P_{ki}}$
        Update coalition with N_k for task T_i
      end if
    end for
  end while
  return coalition

The quality of each generated coalition is measured by using the evaluation values of the best proposals for each service's task. At each iteration, Eq. 6

returns the quality of the achieved solution. For an empty set of proposals the quality of the coalition is zero. Note that the quality of the coalition is also zero, if there are not any proposals for one or more remote tasks $T_i \in S$.

$$Q_{coalition} = \left\lfloor \frac{|coalition|}{|S|} \right\rfloor * \sum_{i=1}^{|coalition|} \frac{1 - Best_{P_i}}{|coalition|} \tag{6}$$

After determining an initial coalition, the algorithm continues, if time permits, to evaluate the remaining proposals as it tries to improve the quality of the current solution. It is possible that some other node, while achieving a lower local reward, can still propose a better proposal for the specific user's request under negotiation at the expense of a greater downgrade of previously accepted services.

Each node's local reward is also used to improve a global load balancing. Consider two proposals whose evaluation differ by an amount less than α (this value can be defined by the user or by the framework). For a particular user, the perceived utility will be equally acceptable if any of those nodes is selected for participating in the new coalition. Selecting the node with a higher local reward from two similar service proposals, not only maximises service for a particular user, but also maximises the global system's utility.

The algorithm terminates when all the received proposals are evaluated or if it finds that the quality of a coalition cannot be further improved because the local reward of each node that belongs to the current coalition is maximum.

6.2 Formal description of the coalition formation's anytime behaviour

The coalition formation's anytime behaviour can be formally described using the set of axioms presented in [61]. The authors describe the anytime functionality of an algorithm using four axioms, each of which describes a different aspect of the anytime behaviour as follows:

Axiom 1 (Initial behaviour) *There is an initial period during which the algorithm does not produce a coalition for a cooperative service execution*

The algorithm does not immediately produce an intermediate solution, since it must first analyse a proposal for each remote task $T_i \in S$. If t' indicates the duration of this initial step then, if interrupted at any time $t < t'$, the algorithm will return a coalition with zero quality.

$$\forall_{t<t'} \; Q_{coalition}(t) = 0$$

Axiom 2 (Growth direction) *The quality of a coalition only improves with increasing run time*

The coalition's members are only updated if a better proposal for any task $T_i \in S$ is found.

$$\forall_{t'>t} \; Q_{coalition}(t) \leq Q_{coalition}(t')$$

Axiom 3 (Growth rate) *The amount of increase in the coalition's quality varies during computation*

The solution's quality rapidly increases in the first steps of the algorithm and its growth rate diminishes over time, as a result of the heuristic selection of the next candidate proposal submitted to evaluation.

$$\forall_{t'>t} \; Q_{coalition}(t+1) - Q_{coalition}(t) > Q_{coalition}(t'+1) - Q_{colaition}(t')$$

Axiom 4 (End condition) *After evaluating all candidate proposals the algorithm achieves its full functionality*

After evaluating all candidate proposals the anytime version of the algorithm will produce exactly the same solution quality as its traditional version [7] that only produces a solution with quality $Q'_{coalition}$ at the end of computation. If the time required to evaluate a candidate proposal is t_e, the total required runtime of the anytime algorithm is the sum of all n evaluations.

$$Q_{coalition}(n * t_e) = Q'_{coalition}$$

6.3 Conformity of the coalition formation algorithm with the desirable properties of anytime algorithms

Not every algorithm that can produce a sequence of approximate results is a well-behaved anytime algorithm. According to Zilberstein [14] the desired properties of anytime algorithms include the following features: a *measurable quality* that can be determined precisely, a *recognisable quality* that can be easily determined at run time, the *monotonicity* of the result's quality, the *consistency* of the result's quality with respect to computation time and input quality, the *diminishing returns* of the solution's quality over time, the *interruptibility* of the algorithm at any time and its *preemptibility* with minimal overhead. The conformity of the proposed anytime coalition formation algorithm with these desirable properties is checked in the next paragraphs.

Property 6.3.1 (Measurable quality) *A coalition's quality can be determined precisely*

Proof: According to Eq. 6, the quality of the generated coalition at each iteration of the algorithm can be directly computed from the evaluation values of the best proposals found for each of the service's tasks.

<div align="right">□</div>

Property 6.3.2 (Recognisable quality) *The quality of a coalition can be easily determined at run time*

Proof: Let $S = T_1, \ldots, T_n$ be the service under negotiation for a cooperative execution with a set of n tasks.

A coalition c is only updated to c' when a better proposal for task $T_i \in S$ is found. The previous accepted service proposal P_{ki}, from node N_k to task T_i, is replaced with $P_{k'i}$ from node $N_{k'}$, in the previously generated coalition c.

Let $|c|$ be the size of the generated coalition to execute service S, $E_{P_{k'i}}$ be the evaluation value of the new proposal $P_{k'i}$, and $E_{P_{ki}}$ be the evaluation value of the old proposal P_{ki}.

The quality of the updated coalition $Q_{c'}$ can calculated by adding the quality Q_c achieved by coalition c to the weighted difference between $E_{P_{k'i}}$ and $E_{P_{ki}}$.

$$Q_{c'} = Q_c + \frac{E_{P_{k'i}} - E_{P_{ki}}}{|c|}$$

This makes the determination of the new coalition's quality straightforward and within a constant time.

<div align="right">□</div>

Property 6.3.3 (Monotonicity) *The quality of the generated coalition is a nondecreasing function of time*

Proof: Node N_k is only added to a coalition if and only if it proposes a better service for task T_i, that is, if it is closer to the user's quality preferences than the best proposal found so far for task T_i.

The algorithm always returns the coalition with the best proposals evaluated until time t for each of the service's tasks, which can be different from the last set of evaluated proposals. According to Zilberstein [14], this characteristic in addition to a recognisable quality is sufficient to prove the monotonicity of an anytime algorithm.

<div align="right">□</div>

Property 6.3.4 (Consistency) *For a given amount of computation time on a given input, the quality of the generated coalition is always the same*

For a given amount of computation time Δt on a given input of a set of service proposals P and user's QoS preferences Q, the quality of the selected coalition for cooperative service execution is always the same, since the selection of candidate proposals for evaluation is deterministic.

According to Eq. 5, the next proposal to be selected from evaluation P_{ki} for each task $T_i \in S$ is the one sent by the node that has achieved the greatest local reward. As such, the algorithm guarantees a deterministic output quality for a given amount of time and input.

$\square$

Property 6.3.5 (Diminishing returns) *The improvement in the generated coalition's quality is larger at the early stages of the computation and it diminishes over time*

The quality of each generated coalition, given by Eq. 6, is measured using the evaluation values of the best proposals for each task $T_i \in S$. The best proposal is the one that contains the attributes' values more closely related to user's specific QoS preferences, in all QoS dimensions.

Each node's local reward, determined with Eq. 8, expresses a degree of satisfaction for all the users that have tasks being locally executed with specific QoS levels, including the service being currently negotiated.

By selecting for evaluation, for each task $T_i \in S$, the proposal sent by the node that achieved the higher local reward rapidly improves the quality of the generated coalition at an early stage of execution, since a high local reward indicates that the node is offering service closer to the majority of the requested QoS values for all local services. However, some other node may propose a better QoS level for the service under negotiation at the expense of a higher downgrade of previously accepted services, and this achieving a lower local reward. As such, it is still possible that the solution's quality can be further improved in the next iterations of the algorithm, but at a lower increment rate.

The actual concavity of the coalition's formation behaviour is empirically evaluated in Section 8.

$\square$

Property 6.3.6 (Interruptibility) *The algorithm can be stopped at any time and still be able to provide a solution*

Proof: Let t' be the time needed to generate the initial coalition. If interrupted at any time $t < t'$ the algorithm will return an empty coalition, resulting in zero quality.

When stopped at time $t > t'$ the algorithm returns the best coalition determined until time t, which can be different from the last set of evaluated proposals.

$\square$

Property 6.3.7 (Preemptibility) *The algorithm can be suspended and resumed with minimal overhead*

Proof: Since the algorithm keeps the received proposals not yet evaluated and the determined coalition it can be easily resumed after an interrupt.

$\square$

7 Service proposal formulation

All entities that participate in a cooperative QoS-aware service execution negotiation must provide sufficient resources to propose a SLA within the user's acceptable QoS levels. It is therefore the responsibility of each individual QoS Provider (Figure 1) to map the user's QoS constraints to local resource requirements, and then reserve resources accordingly (resource reservations are made through Resource Managers). The interpretation of QoS constraints and consequent mapping on the needed resource quantities has been explored in [26; 62; 63; 64]. This paper is focused in the time-bounded coalition formation process and does not deal with this mapping. The reader can assume that applications make a reasonable accurate analysis of their resource requirements, made a priori through resource monitoring tools and followed by run-time adaptation.

Requests for task execution arrive dynamically at any node and are formulated as a set of acceptable multi-dimensional QoS levels in decreasing preference order. To guarantee the request locally, the QoS Provider executes a local QoS optimisation algorithm that tries to maximise the satisfaction of the new service's QoS constraints as well as to minimise the impact on the current QoS of previously accepted services.

Conventional admission control schemes either guarantee or reject each service request, implying that future requests may be rejected because resources have already been committed to previous requests. We use a QoS negotiation

mechanism that, in cases of overload, or violation of pre-run-time assumptions guarantees a graceful degradation.

The CooperatES dynamically determines promised QoS levels from the user's accepted QoS values for each QoS dimension and local resources' availability. Furthermore, the reward of executing a task at one of those dynamically determined QoS levels depends on the number, and relative importance, of the QoS dimensions being served closer to the user's desired QoS level. Eq. 7 computes the reward r_{T_i} achieved by a specific SLA for task T_i by measuring the distance between the user's desired and the node's proposed values.

$$
r_{T_i} = \begin{cases} 1 & \text{if task is being best} \\ & \text{served in all QoS dimensions} \\ 1 - \sum_{j=0}^{n} w_j * penalty_j & \text{if } Q_{jk} < Q_{best_j} \end{cases} \tag{7}
$$

In Eq. 7 *penalty* is a parameter that decreases the reward value. This parameter can be fine tuned by the user or the framework's manager according to several criteria and its value should increase with the distance to the user's preferred values.

Using the utility achieved by each proposed SLA it is possible to determine a measure of the node's global satisfaction resulting from the acceptance of the new service request. For a node N_j, the local reward R_j achieved by the set of proposed SLAs is given by

$$
R_j = \frac{\sum_{i=1}^{n} r_{T_i}}{n} \tag{8}
$$

Note that unless all tasks are executed at their highest requested QoS level there is a difference between the determined set of SLAs and the maximum theoretical local reward that would be achieved if all local tasks were executed at their highest QoS level. This difference can be caused by either resource limitations, which is unavoidable, or poor load balancing, which can be improved by sending actual local rewards in service proposals, and selecting, for proposals with similar evaluation values, those nodes that achieve higher local rewards, as discussed in the previous section.

7.1 Anytime local QoS optimisation

The high complexity of determining the best set of SLAs, taking into account both the users' QoS preferences and node's resource availability, makes it beneficial to propose an anytime approach that can trade the achieved solution's quality by its computational cost in order to ensure a timely answer to events.

The proposed anytime algorithm, detailed in Algorithm 2, clearly splits the formulation of a new set of SLAs in two different scenarios. The first one involves guaranteeing the new task without changing the QoS level of previously guaranteed tasks. The second one, due to the node's overload, demands service degradation for the previous accepted tasks in order to accommodate the new requesting task. Offering QoS degradation as an alternative to task rejection has been proved to achieve higher perceived utility [29].

Algorithm 2 Service proposal formulation

Each task T_i has user defined QoS dimensions constraints Q_i. Each Q_{kj} is a finite set of n quality choices for the jth attribute, expressed in decreasing order of preference, for all k QoS dimensions.

   while $t < timeout$ **do**

      Step 1: Improve QoS level of new arrived task T_a

      Select the worst requested QoS level in all k dimensions, $Q_{kj}[n]$, for task T_a. Maintain level of service for previously guaranteed tasks.

      while the new set of tasks *is* feasible **do**

         for each QoS dimension in T_a receiving service at $Q_{kj}[m] > Q_{kj}[0]$ **do**

            Determine the utility increase by upgrading attribute j to $m-1$

            Find maximum increase and upgrade attribute x to the $m-1$'s level

         end for

      end while

      Step 2: Find local minimal service degradation to accommodate T_a

      Select for all k dimensions of task T_a the final result of Step 1, $Q_{kj}[m]$

      while the new set of tasks *is not* feasible **do**

         for each task T_i receiving service at $Q_{kj}[m] > Q_{kj}[n]$ **do**

            Determine the utility decrease by degrading attribute j to $m+1$

            Find task T_{min} whose reward decrease is minimum and degrade attribute x to the $m+1$'s level

         end for

      end while

   end while

   return SLA for new task

The algorithm iteratively works on the problem of finding a feasible set of SLAs while maximising the users' satisfaction and produces results that improve in quality over time. Instead of a binary notion of the solution's correctness, the

algorithm returns a proposal and a measure of its quality. The quality of each generated configuration Q_{conf}, given by Eq. 9, considers the reward achieved by the service proposal configuration for the new arriving task r_{T_a}, the impact on the provided QoS of previous existing tasks and the value of the previous generated feasible configuration Q'_{conf}. Initially, Q'_{conf} is set to zero.

$$
Q_{conf} = \begin{cases} \left(r_{T_a} * \dfrac{\sum\limits_{i=0}^{n} r_{T_i}}{n} \right)^{(1-Q'_{conf})} & \text{if feasible} \\ \\ \\ Q'_{conf} & \text{if not feasible} \end{cases} \tag{9}
$$

When a new service request arrives, the algorithm starts by maintaining the QoS levels of previously guaranteed tasks and by selecting the worst requested QoS level, for all dimensions, for the new arrived task. The goal is to quickly find a feasible initial solution, an important property of anytime algorithms. Note that this is the SLA that has a higher probability of being feasible, without requiring any modification on the current QoS of previously accepted services.

The algorithm continues to improve the quality of that initial solution, conducting the search for a better feasible solution in a way that maximises the expected improvement in the solution's quality. With spare resources, the algorithm incrementally selects the configuration that maximises the increase in obtained reward for the new task. When QoS degradation is needed to accommodate the new task, the algorithm incrementally selects the configuration that minimises the decrease in obtained reward for the new set of tasks, which includes the new arrived one.

Note that the algorithm may produce an unfeasible set of SLAs due to local resource availability. As such, since a service proposal can only be considered useful within a feasible set of configurations, the algorithm, if interrupted, always returns the last found feasible solution. Nevertheless, each intermediate configuration, even if not feasible, is used to calculate the next possible solution, minimising the search effort.

The algorithm terminates when the time for the reception of proposals has expired (this time is sent in the cooperation request), when it finds a set of QoS levels that keeps all tasks feasible and the quality of the solution can not be further improved, or when it finds that, even at the lowest QoS level for each task, the new set is not feasible. In this case the new service request is rejected. When it is not possible to find a valid solution for service execution

within available time, then no proposal will be sent to the requesting node, and the node continues to serve existing tasks at their current QoS levels.

7.2 Formal description of the service proposal formulation's anytime behaviour

Similarly to what has been presented for the coalition formation algorithm, the different aspects of the anytime functionality of the service proposal formulation algorithm will be described in this section using the four axioms presented in [61].

In the next paragraphs, the different aspect of the anytime behaviour of the proposed service proposal formulation algorithm will be formally described using the set of axioms presented in [61].

Axiom 5 (Initial behaviour) *Until a feasible set of SLAs is found the new task is rejected*

An intermediate solution can only be considered if it produces a feasible set of SLAs. If t' indicates the time at which the first feasible solution is found then, if interrupted at anytime $t < t'$, the algorithm will reject the new task.

$$\forall_{t<t'} \; Q_{conf}(t) = 0$$

Axiom 6 (Growth direction) *The quality of a feasible set of SLAs can only improve over time*

A new feasible set of SLAs is only considered when it improves the solution's quality.

$$\forall_{t'>t} \; Q_{conf}(t) \leq Q_{conf}(t')$$

Axiom 7 (Growth rate) *The amount of increase in the solution's quality varies during computation*

The solution's quality rapidly increases in the first steps of the algorithm and its growth rate diminishes over time. The algorithm starts by improving the new user's preferred quality attributes until an unfeasible set of SLAs is found. On the other hand, when QoS degradation is needed in the search for a new feasible solution, the algorithm degrades the less important attributes for all local services.

The actual concavity of the algorithm's behaviour is empirically evaluated in Section 8.

$$\forall_{t'>t} \; Q_{conf}(t+1) - Q_{conf}(t) > Q_{conf}(t'+1) - Q_{conf}(t')$$

28

Axiom 8 (End condition) *When is not possible to improve the solution's quality the algorithm achieves its full functionality*

When it runs to completion, the anytime version of the algorithm will produce exactly the same solution as its traditional version [7] that only produces a solution with quality $Q'_c onf$ at the end of its computation time.

The anytime version terminates when it finds a set of QoS levels that keeps all tasks feasible and the quality of that solution can not be further improved, or when it finds that, even at the lowest QoS level for each task, the new set is unfeasible.

If the time required to improve or degrade an attribute and test for the schedulability of the solution is given by t_s, the total required runtime of the anytime algorithm is the sum of all n needed changes in attributes to find the best feasible solution.

$$Q_{conf}(n * t_s) = Q'_{conf}$$

7.3 Conformity of the service proposal formulation algorithm with the desirable properties of anytime algorithms

The conformity of the proposed anytime service proposal formulation algorithm according to the desirable properties of anytime algorithms [14] is checked in the next paragraphs.

Property 7.3.1 (Measurable quality) *The quality of a SLA can be determined precisely*

Proof: At each iteration of the algorithm, Eq. 9 measures the quality of the proposed SLA by considering the proximity of the proposal with respect to the user's request under negotiation and the impact of that proximity on the global utility achieved by the previously accepted tasks.

$\square$

Property 7.3.2 (Recognisable quality) *The quality of a set of SLAs can be easily determined at run time*

Proof: The quality of each generated feasible set of SLAs is determined by using the rewards achieved by all tasks being locally executed, which includes the newly arrived one. According to Eq. 7, the rewards' computation is straightforward and time-bounded.

$\square$

29

Property 7.3.3 (Monotonicity) *The quality of the generated set of SLAs is a nondecreasing function of time*

Proof: The algorithm produces a new set of SLAs at each iteration, as it tries to maximise the utility increase for the new task while minimising the utility decrease for all previously accepted tasks when the resources are scarce to accommodate the new task. Since a service proposal can only be considered useful within a feasible set of tasks, the algorithm always returns the best found feasible solution rather than the last generated SLA.

According to Zilberstein [14], this characteristic in addition to a recognisable quality is sufficient to prove the monotonicity of an anytime algorithm.

□

Property 7.3.4 (Consistency) *For a given amount of computation time on a given input, the quality of the generated SLA is always the same*

Proof: For a given amount of computation time Δt on a given input of a set of QoS constraints Q associated with a set of tasks τ, the quality of the proposed SLA is always the same, since the selection of attributes to improve or degrade is deterministic.

At each iteration, the QoS attribute to be improved is the one that maximises an increase in the reward achieved by the new arrived task, while the QoS attribute to be degraded is the one that minimises the decrease in the global reward achieved by all tasks being locally executed. As such, the algorithm guarantees a deterministic output quality for a given amount of time and input.

□

Property 7.3.5 (Diminishing returns) *The improvement in the quality of the generated SLA is larger at the early stages of the computation and it diminishes over time*

Proof: An initial solution that maintains the QoS levels of the previously guaranteed tasks and selects the worst requested level in all QoS dimensions for the new arrived task is quickly generated. Its quality is given by Eq. 9, considering the rewards achieved by all tasks.

At each iteration, the currently found solution is improved by either upgrading the QoS attribute that maximises an increase in the new service's utility or by downgrading the QoS attribute that minimises the decrease in all local services' utility. As such, the increment in the solution's quality is larger at the firsts iterations and it diminishes over time.

$\square$

Property 7.3.6 (Interruptibility) *The algorithm can be stopped at any time and still provide a solution*

Proof: Let t' be the time needed to generate the first feasible solution. In interrupted at any time $t < t'$ the algorithm will return an empty SLA, resulting in zero quality.

When stopped at time $t > t'$ the algorithm returns the best feasible set of SLAs generated until t, which can be different from the last evaluated set.

$\square$

Property 7.3.7 (Preemptibility) *The algorithm can be suspended and resumed with minimal overhead*

Proof: Since the algorithm maintains the best generated feasible solution and the current configuration values it can be easily resumed after an interrupt.

$\square$

8   Evaluation

The ideal way to evaluate the performance of the several algorithms proposed in this thesis would be to subject them to actual loads from a large portfolio of QoS-aware applications and embedded devices. Nevertheless, we have chosen to evaluate the effectiveness of the CooperatES framework by creating a broad collection of profiles, chosen to cover the spectrum into which real applications and both embedded and their more powerful neighbour devices would fall or likely exhibit.

Furthermore, a special attention was devoted to introduce a high variability in the characteristics of the conducted simulations. It is known that not much can be concluded with a single simulation run. In fact, the results of a given simulation run are just particular instantiations of random variables that may have large variances [65]. While most of the methods for the analysis of the simulation's output data rely on the fact that although the simulation results of a single simulation run are not independent, it is still possible to obtain independent observations across the results of several simulation runs (or simulation replicas) with a reasonably good statistical performance [66].

The main goal of the conducted evaluations was to measure the improvement in the performance of the CooperatES framework when operating in open

and dynamic environments with respect to its previous version, where a traditional QoS optimisation approach was used [7]. In Section 8.2, the performance profiles of the proposed anytime algorithms are determined and Section 8.3 discusses the computational cost of the proposed anytime algorithms when compared against our previous traditional QoS optimisation approach.

The reported results were observed from multiple and independent simulation runs, with initial conditions and parameters, but different seeds for the random values [2] used to drive the simulations, obtaining independent and identically distributed variables. The mean values of all generated samples were used to produce the charts, with a confidence level of 99,9% associated to each confidence interval. A confidence interval specifies a range of values within which the unknown population parameter, in this case the mean, may lie. For each chart, the wider confidence interval is discussed.

8.1 Setup

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 as a scenario for the simulations. The application was composed by a set of source units to collect the data, a compression unit to gather and compress the data that came from the multiple sources, a transmission unit to transmit the data over the network, a decompression unit to convert the data into each user's specified format, and an user unit to display the data in the user's end device.

The number of simultaneous nodes in the system varied from 10 to 100 while the number of simultaneous users varied from 1 to 20, generating different amounts of load and resource availability in the system. Each node was running a prototype implementation of the CooperatES framework, with a fixed set of mappings between requested QoS levels and resource requirements. The code bases needed to execute each of the streaming 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 from the set of characteristics described in Table 1, 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.

Requested QoS levels were randomly generated, at randomly selected end de-

---

[2] The random values were generated by the Mersenne Twister algorithm [67].

Table 1
Possible characteristics of nodes

Resource	Type
cpu	400 MHz, 750 MHz, 1 GHz, 1.5 GHz, 2 GHz, 2.5 GHz, 3 GHz
memory	128 MB, 256 MB, 512 MB, 1 GB, 2 GB, 4 GB, 8 GB
storage	512 MB, 1 GB, 10 GB, 30 GB, 50 GB, 200 GB, 500 GB
network	10 Mbps, 11 Mbps, 54 Mbps, 100 Mbps, 1 Gbps
display	none,240x180,320x240,640x480,720x480, 1024x768, 1280x1024

vices and at randomly generated times, expressing the spectrum of 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.

The QoS domain used to generate the users' service requests was composed by the following list of QoS dimensions, attributes, and possible values.

```
QoS dimensions = {Media Container,Video Quality, Audio Quality}
Media Container = {container format}
Video Quality = {color depth, frame size, frame rate}
Audio Quality = {sampling rate, sample bits}

container format = {3GP, ASF, AVI, QuickTime, RealVideo, WMV}
color depth (bits) = {1, 3, 8, 16, 24}
frame size (pixels) = {240x180, 320x240, 640x480, 720x480,
                       1024x768,1280x1024}
frame rate (per second) = {[1,30]}
sampling rate (kHz) = {8, 11, 32, 44, 88}
sample bits (bits) = {4, 8, 16, 24}
```

The simulator used in the conducted experiments was custom built by the authors in Erlang [68], a functional programming language designed to be run in a distributed environment populated with resource constrained devices.

8.2 Evaluating performance profiles

The behaviour of an anytime algorithm is described by a performance profile. The performance profile describes how the quality of the output gradually increases as a function of the computation time [14]. Such description of its behaviour is very attractive because it allows a tradeoff to be made between

available computation time and output quality.

Since there are many possible factors affecting the execution time of an algorithm, rather than measuring the algorithms' absolute execution time on every simulation run, we have normalised it with respect to its completion time [69]. As such, in the next figures the algorithms' computation times are represented as a percentage of their respective completion times. Nevertheless, both algorithms needed an average time lower than 1 second to compute their optimal solutions on a Intel Core Duo T5500 at 1.66 GHz.

Figure 3 shows the performance profile of the proposed anytime coalition formation algorithm, comparing two methods for selecting the next proposal to be evaluated. The first one selects the proposal sent by the node with the highest local reward, while the second one relies on the order of proposals' reception.
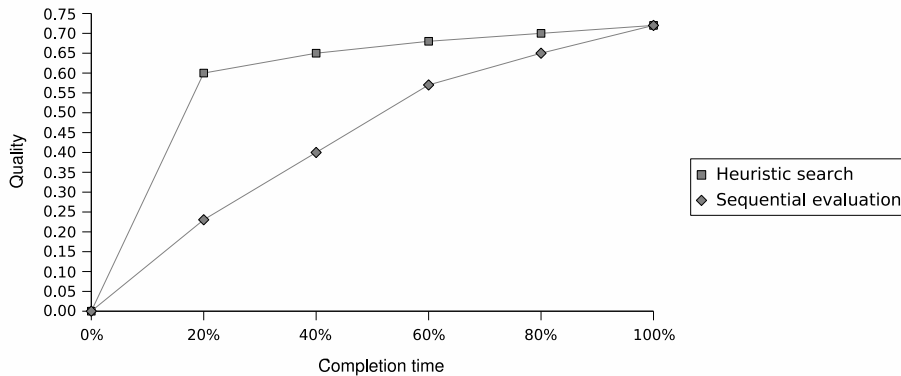
Fig. 3. Performance profile of the coalition formation algorithm

Clearly, the proposed heuristic selection based on the nodes' local reward achieves a better performance and lower variation. At only 20% of the completion time, when selecting proposals based on the nodes' local reward, the algorithm achieves a solution's quality of 83% ± 6% of its optimal solution, with a 99,9% confidence level. On the other hand, when evaluating proposals according to their arrival order, the solution's expected quality varies on a higher degree. At 20% of the completion time, the algorithm achieves a solution's quality of 32% ± 25% of its optimal solution, with a 99,9% confidence level.

Therefore, the proposed heuristic selection of proposals based on the nodes' local reward effectively maximises the expected improvement in the solution's quality at an early stage of the needed computation time. This is a very important property for an anytime algorithm's practical usefulness, since it means that after a small period of the running session, the results are expected to be sufficiently close to the results at completion time.

34

Also note that these results empirically validate the formal analysis of the desirable properties of anytime algorithms discussed in Section 6.3. From the algorithm's performance profile plotted in Figure 3 one can conclude that the coalition's quality measure is a non-decreasing function of time whose increment diminishes over time.

A second study evaluated the behaviour of the anytime service proposal formulation algorithm by measuring its performance profile as well as the impact generated by the arrival of a new service on the QoS level of previously accepted tasks. Recall from Section 7 that the reward of a specific proposal measures how useful it will be for a particular user and that the local reward expresses a degree of global satisfaction for all the users that have tasks being executed at a particular node.

The results were plotted by averaging the results of several independent runs of the simulation, divided in two categories. Figure 4 presents the scenario where the average amount of available resources per node is greater than the average amount of resources demanded by the services being executed. The opposite scenario is represented in Figure 5, where the average amount of resources per node is smaller than the average amount of demanded resources.
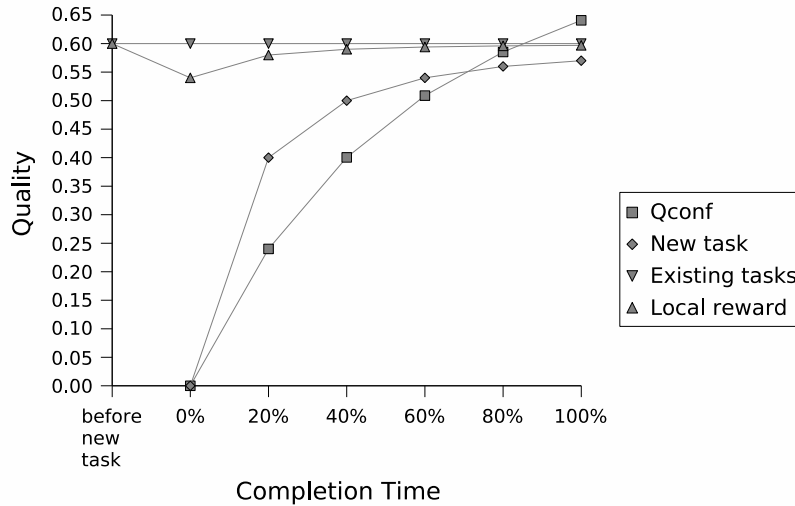

Fig. 4. Expected quality improvement with spare resources

In Figure 4, the increase in the solution's quality Q_{conf} results from the increase in the new task's reward (Step 1 of the algorithm). Recall that with spare resources the QoS levels of previously accepted tasks remains the same. As such, this increase in the new service's reward also increases the node's local reward, that was affected by the initially proposed solution of serving the new arrived service at the minimum requested QoS level. However, due to resource limitations (Figure 5), when trying to upgrade the reward achieved by the new service, the generated configuration may result in an unfeasible set of SLAs. The Step 2 of service formulation algorithm is then executed in

order to try to find a new feasible solution that presents a higher satisfaction for the service request under negotiation. As discussed in Section 6, it is the responsibility of the coalition formation algorithm to select between proposals with similar evaluation values for the new user, those nodes that achieve higher local rewards, promoting load balancing.
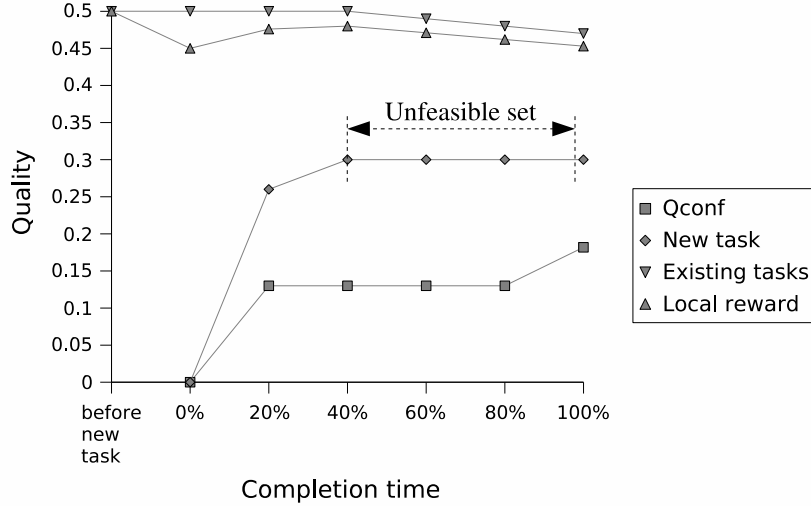

Fig. 5. Expected quality improvement with limited resources

Note that, in both scenarios, the proposed heuristics optimise the rate at which the quality of the determined solution for the new task improves over time. With spare resources (Figure 4), at only 20% of the computation time, the solution's quality for the new arrived task is near 70% ± 5% of the achieved quality at completion time. When QoS degradation is needed to accommodate the new task (Figure 5), its service proposal achieves 87% ± 4% of its final quality at 20% of computation time.

Also note that, the quality of the node's global service solution identified by $Qconf$ in both figures, quickly approaches its maximum value at an early stage of the computation. The diminishing returns property of the anytime service proposal formulation was then empirically verified, after is formal analysis in Section 7.3. The same is true for the quality measure of a service proposal which is a non-decreasing function of time, since the best feasible configuration is only replaced if, and only if, another feasible solution is found and has a higher quality for the user's request under negotiation (Figure 5).

Both studies clearly demonstrate that the proposed anytime algorithms can be useful even when there is no time to compute an optimal local and global resource allocation. The solution's quality can be improved if the algorithms have more time to run, but it rapidly approaches its optimal value at an early stage of the needed computation time. For complex and dynamic real-time systems, allowing a cooperative service configuration to be determined at any time results in a significant improvement of the CooperatES framework.

Throughout the paper the idea that complex scenarios may prevent the possibility of computing optimal resource allocations was stated and anytime algorithms that can tradeoff the needed deliberation time for the quality of the achieved results were proposed. However, it is important to analyse the computational cost required by this approach to reach its optimal solution when compared against the traditional versions of those algorithms [7].

Without any restriction on the needed time to find an optimal resource allocation, both algorithms follow more straightforward approaches than their relative anytime counterparts. The traditional coalition formation algorithm evaluates all received proposals, based on their order of arrival, and selects the best proposal for each of the service's remote tasks. The traditional service proposal formulation starts by selecting the user's preferred QoS level and, at each iteration, it downgrades the QoS attribute that minimises the decrease in the node's local reward, until a feasible set of SLAs is found.

This section evaluates the impact of those different approaches on the needed computation time to find an optimal solution. The results discussed in the next paragraphs were normalised with respect to the completion time of the traditional version of the algorithms.

Figure 6 details the computational cost of both versions of the coalition formation algorithm. The traditional version is slightly faster than its anytime counter part, it requires nearly $95\% \pm 2\%$ of the time needed by the anytime approach to reach its final, and optimal, solution. This difference is explained by the way both algorithms select the next proposal to evaluate. While the traditional version sequentially evaluates service proposals according to their order of arrival, the anytime approach selects proposals based on the nodes' local reward. This implies either to sort the proposals' set before starting the evaluation process or to search, at each iteration, for the maximum remaining local reward.

However, note that the anytime version needs as little as near 10% of its completion time to achieve a solution's quality of $50\% \pm 6\%$ of its optimal solution, and below 40% to achieve $90\% \pm 5\%$ of its optimal solution. As discussed in the previous section, such results cannot be achieved by evaluating proposals based on their arrival order.

The comparison among both versions of the service proposal formulation algorithm is detailed in Figures 7 and 8. Figure 7 compares the needed computation time when the average amount of resources per node is greater than the average amount of resources necessary for each service execution, while Figure 8 compares both versions when the average amount of resources per
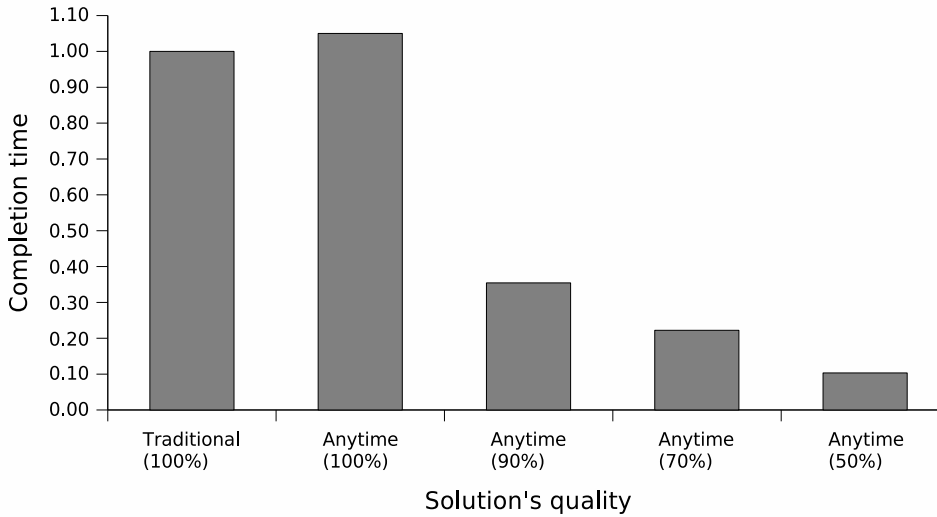
Fig. 6. Coalition formation: Anytime vs Traditional

node is smaller than the average amount of resources necessary for each service execution, demanding QoS degradation of the previously accepted services.
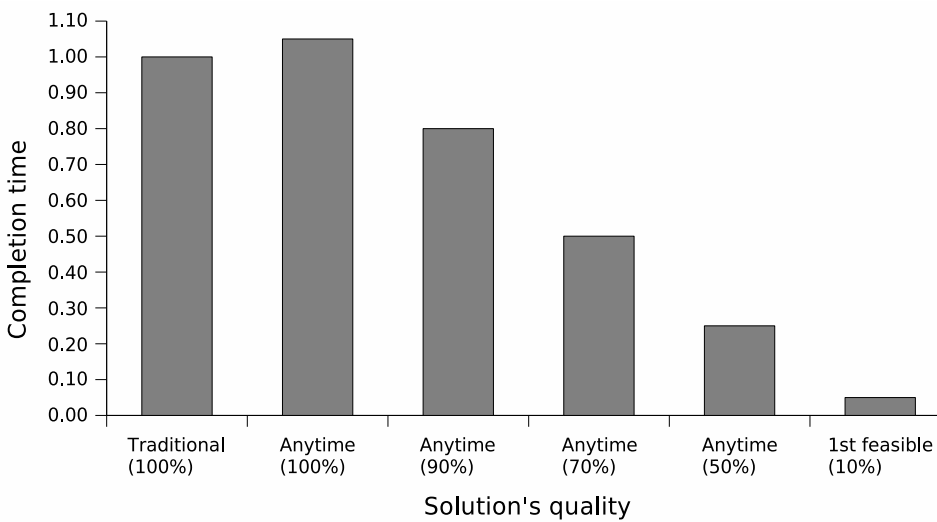

Fig. 7. Proposal formulation: Anytime vs Traditional with spare resources

Both figures allow us to conclude that, similarly to the coalition formation algorithm, the traditional version is slightly faster to achieve its only and optimal solution. While the anytime approach tries to quickly find an initial feasible solution by considering the worst QoS level requested by the user, the traditional version starts by selecting the user's preferred QoS level. As such, with spare resources the traditional version is faster to achieve the optimal resource allocation for the new set of tasks, while with limited resources both versions need almost the same time.

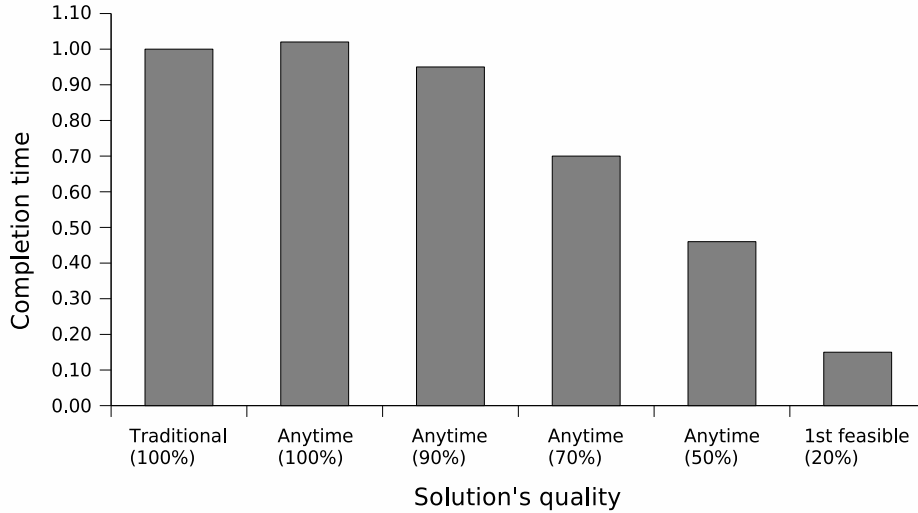However, in both scenarios the anytime version is by far quicker to find a

Fig. 8. Proposal formulation: Anytime vs Traditional with limited resources

feasible solution. With spare resources the needed time to find the first feasible solution with a quality near 10% ± 3% of the optimal solution is less than 5% of its completion time. With limited resources, the anytime version takes about 20% of its computation time to reach a feasible solution with 15% ± 3% of the optimal solution's quality.

With these results for both algorithms, the advantages of the anytime approach are clearly demonstrated, as its overhead can be considered negligible when compared against the introduced benefits. The anytime approach quickly achieves a feasible global and local resource allocation and it maximises the rate at which the quality of that initial solution increases over time.

9 Conclusions

As the complexity of various applications increases, multiple tasks have to compete for the limited resources of a single device. In this context, resource constrained devices may need to collectively execute services with their neighbours in order to fulfil the complex QoS constraints imposed by users and applications. As such, it becomes very important to provide an efficient arbitration of QoS levels in this highly dynamic, open, shared, and heterogeneous environment.

This paper presented the CooperatES framework, a QoS-aware framework that addresses the increasing demands on resources and performance by allowing services to be executed by temporary coalitions of nodes. Users encode their own relative importance of the different QoS parameters for each service and the framework uses this information to determine the distributed resource

allocation that maximises the satisfaction of those constraints.

However, finding an optimal distributed service provisioning that deals with both users' and service providers' quality constraints can be extremely complex and take a long time. Unlike conventional QoS optimisation algorithms that guarantee a correct output only after termination, the proposed anytime approach does not rely on the availability of deliberation time to provide a solution and a measure of its quality. This tradeoff between the available computation time and the quality of the achieved solution is a powerful and useful approach in open dynamic real-time systems where, despite their uncertainty, responses to events still have to be provided within precise timing constraints in order to guarantee a desired level of performance.

The conformity of the proposed algorithms with the desired properties of anytime algorithms was proved and the design decisions of our approach validated through extensive simulations in highly dynamic scenarios. The achieved results clearly demonstrate that proposed anytime algorithms are able to quickly find a good initial solution and effectively optimise the rate at which the quality of the current solution improves at each iteration of the algorithms, with an overhead that can be considered negligible when compared against the introduced benefits.

References

[1] J. P. Hansen, J. Lehoczky, R. Rajkumar, Optimization of quality of service in dynamic systems, in: Proceedings of the 9th International Workshop on Parallel and Distributed Real-Time Systems, 2001.

[2] S. Ghosh, R. R. Rajkumar, J. Hansen, J. Lehoczky, Integrated resource management and scheduling with multi-resource constraints, in: Proceedings of the 25th IEEE Real-Time Systems Symposium, Lisbon, Portugal, 2004, pp. 12–22.

[3] M. Stonebraker, U. Cetintemel, S. Zdonik, The 8 requirements of real-time stream processing, SIGMOD Record 34 (4) (2005) 42–47.

[4] L. Marcenaro, F. Oberti, G. L. Foresti, C. S. Regazzoni, Distributed architectures and logical-task decomposition in multimedia surveillance systems, Proceedings of the IEEE 89 (10) (2001) 1419–1440.

[5] V. S. W. Eide, F. Eliassen, O.-C. Granmo, O. Lysne, Supporting timeliness and accuracy in distributed real-time content-based video analysis, in: Proceedings of the 11th ACM international conference on Multimedia, ACM Press, 2003, pp. 21–32.

[6] S. Schmidt, T. Legler, D. Schaller, W. Lehner, Real-time scheduling for data stream management systems, in: Proceedings of the 17th Euromicro Conference on Real-Time Systems, 2005, pp. 167–176.

[7] L. Nogueira, L. M. Pinho, Dynamic qos-aware coalition formation, in:

Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, Denver, Colorado, 2005, p. 135.

[8] L. Nogueira, L. M. Pinho, Iterative refinement approach for qos-aware service configuration, IFIP From Model-Driven Design to Resource Management for Distributed Embedded Systems 225 (2006) 155–164.

[9] L. Nogueira, L. M. Pinho, Capacity sharing and stealing in dynamic server-based real-time systems, in: Proceedings of the 21th IEEE International Parallel and Distributed Processing Symposium, Long Beach,CA,USA, 2007, p. 153.

[10] J. van den Berg, D. Ferguson, J. Kuffner, Anytime path planning and replanning in dynamic environments, in: Proceedings of the IEEE International Conference on Robotics and Automation, Orlando, Florida, USA, 2006, pp. 2366– 2371.

[11] J. Shackleton, D. Cofer, S. Cooper, Anytime scheduling for real-time embedded control applications, in: Proceedings of the 23rd Digital Avionics Systems Conference, Vol. 2, Salt Lake City, UT, USA, 2004, pp. 101–110.

[12] N. Hawes, Anytime deliberation for computer game agents, Ph.D. thesis, School of Computer Science, The University of Birmingham (November 2003).

[13] R. Bhattacharya, G. J. Balas, Anytime control algorithm: Model reduction approach, Journal of Guidance, Control, and Dynamics 27 (5) (2004) 767–776.

[14] S. Zilberstein, Using anytime algorithms in intelligent systems, Artificial Inteligence Magazine 17 (3) (1996) 73–83.

[15] J. A. Stankovic, K. Ramamritham, The spring kernel: A new paradigm for real-time systems, IEEE Software 8 (3) (1991) 62–72.

[16] C. W. Mercer, S. Savage, H. Tokuda, Processor capacity reserves: Operating system support for multimedia applications, in: Proceedings of the IEEE International Conference on Multimedia Computing and Systems, 1994, pp. 90–99.

[17] M. Jones, P. Leach, R. Draves, J. Barrera, Modular real-time resource management in the rialto operating system, in: Proceedings of the Fifth Workshop on Hot Topics in Operating Systems, IEEE Computer Society, 1995, p. 12.

[18] R. Rajkumar, K. Juvva, A. Molano, S. Oikawa, Resource kernels: a resource-centric approach to real-time and multimedia systems, Readings in multimedia computing and networking (2001) 476–490.

[19] D. Clark, S. Shenker, L. Zhangn, Supporting real-time applications in an integrated services packet network: Architecture and mechanism, in: Proceedings of the SIGCOMM'92 Symposium on Communications Architectures and Protocols, 1992, pp. 14–26.

[20] L. Zhang, S. Deering, D. Estrin, S. Shenker, D. Zappala, Rsvp: A new resource reservation protocol, IEEE Network (1993) 8–18.

[21] R. Branden, D. Clark, S. Shenker, IETF RFC 1633: Integrated services in the internet architecture: an overview (1994).

[22] H. Schulzrinne, S. Casner, R. Frederic, V. Jacobson, IETF RFC 1889: Rtp: A transport protocol for real-time applications (1996).

[23] E. D. Jensen, C. D. Locke, H. Tokuda, A time-driven scheduling model for real-time operating systems, in: Proceedings of the 6th IEEE Real-Time Systems Symposium, 1985.

[24] S. Brandt, G. Nutt, T. Berk, J. Mankovich, A dynamic quality of service middleware agent for mediating application resource usage, Proceedings of the 19th IEEE Real-Time Systems Symposium (1998) 307–317.

[25] C. Curescu, S. Nadjm-Tehrani, Time-aware utility-based qos optimisation, Proceedings of the 15th Euromicro Conference on Real-Time Systems (2003) 83–92.

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

[27] S. Ghosh, R. Rajkumar, J. P. Hansen, , J. P. Lehoczky, Scalable resource allocation for multi-processor qos optimization, in: Proceedings of the 23rd International Conference on Distributed Computing Systems, IEEE Computer Society, Rhode Island, USA, 2003, p. 174.

[28] S. Ghosh, J. Hansen, R. R. Rajkumar, J. Lehoczky, Adaptive qos optimizations with applications to radar tracking, in: Proceedings of the 10th International Conference on Real-Time and Embedded Computing Systems and Applications, Gothenburg, Sweden, 2004.

[29] T. F. Abdelzaher, E. M. Atkins, K. G. Shin, Qos negotiation in real-time systems and its application to automated flight control, IEEE Transactions on Computers, Best of RTAS '97 Special Issue 49 (11) (2000) 1170–1183.

[30] S. Khan, Quality adaptation in a multisession multimedia system: Model, algorithms and architecture, Ph.D. thesis, University of Victoria (1998).

[31] I. Foster, M. Fidlerc, A. Royd, V. Sandere, L. Winkler, End-to-end quality of service for high-end applications, Elsevier Computer Communications Journal 27 (14) (2004) 1375–1388.

[32] L. Palopoli, P. Valente, T. Cucinotta, L. Marzario, A. Mancina, A unified framework for multiple type resource scheduling with qos guarantees, in: Proceedings of the Workshop on Operating Systems Platforms for Embedded Real-Time Applications, Palma de Mallorca, Spain, 2005, pp. 67–75.

[33] T. Cucinotta, L. Palopoli, L. Marzario, G. Lipari, L. Abeni, Adaptive reservations in a linux environment., in: Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium, Toronto, Canada, 2004, pp. 238–245.

[34] C. Li, L. Li, Utility-based qos optimisation strategy for multi-criteria scheduling on the grid, Journal of Parallel and Distributed Computing 67 (2) (2007) 142–153.

[35] C. Wang, Z. Li, Parametric analysis for adaptive computation offloading, in: Proceedings of the ACM SIGPLAN 2004 Conference on Programming

Language Design and Implementation, ACM Press, 2004, pp. 119–130.

[36] X. Gu, A. Messer, I. Greenberg, D. Milojicic, K. Nahrstedt, Adaptive offloading for pervasive computing, IEEE Pervasive Computing Magazine 3 (3) (2004) 66–73.

[37] Z. Li, C. Wang, R. Xu, Task allocation for distributed multimedia processing on wirelessly networked handheld devices, in: Proceedings of the 16th International Symposium on Parallel and Distributed Processing, IEE Computer Society, 2002, p. 79.

[38] Z. Li, C. Wang, R. Xu, Computation offloading to save energy on handheld devices: a partition scheme, in: Proceedings of the 2001 International Conference on Compilers, Architecture and Synthesis for Embedded Systems, ACM Press, 2001, pp. 238–246.

[39] U. Kermer, J. Hicks, J. Rehg, A compilation framework for power and energy management on mobile computers, in: 14th International Workshop on Parallel Computing, 2001, pp. 115–131.

[40] M. Othman, S. Hailes, Power conservation strategy for mobile computers using load sharing, SIGMOBILE Mobile Computing Communications Review 2 (1) (1998) 44–51.

[41] A. Rudenko, P. Reiher, G. J. Popek, G. H. Kuenning, Saving portable computer battery power through remote process execution, Mobile Computing and Communications Review 2 (1) (1998) 19–26.

[42] G. Chen, B.-T. Kang, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, R. Chandramouli, Studying energy trade offs in offloading computation/compilation in java-enabled mobile devices, IEEE Transactions on Parallel and Distributed Systems 15 (9) (2004) 795–809.

[43] J. W. Liu, K.-J. Lin, W.-K. Shih, A. C. shi Yu, J.-Y. Chung, W. Zhao, Algorithms for scheduling imprecise computations, IEEE Computer 24 (5) (1991) 58–68.

[44] J. W. S. Liu, K.-J. Lin, R. Bettati, D. Hull, A. Yu, Use of imprecise computation to enhance dependability of real-time systems, Foundations of Dependable Computing: Paradigms for Dependable Applications (1994) 157–182.

[45] T. Dean, M. Boddy, An analysis of time-dependent planning, in: Proceedings of the 7th National Conference on Artificial Intelligence, 1988, pp. 49–54.

[46] E. J. Horvitz, Reasoning under varying and uncertain resource constraints, in: Proceedings of the 7th National Conference on Artificial Intelligence, 1988, pp. 111–116.

[47] L. Abeni, G. Buttazzo, Integrating multimedia applications in hard real-time systems, in: Proceedings of the 19th IEEE RTSS, Madrid, Spain, 1998, p. 4.

[48] C. L. Liu, J. Layland, Scheduling algorithms for multiprogramming in a hard-real-time environment, Journal of the ACM 1 (20) (1973) 40–61.

[49] G. Lipari, S. Baruah, Greedy reclamation of unused bandwidth in constant-bandwidth servers, in: Proceedings of the 12th ECRTS, Stock-

holm, Sweden, 2000, pp. 193–200.

[50] M. Caccamo, G. Buttazzo, L. Sha, Capacity sharing for overrun control, in: Proceedings of 21th IEEE RTSS, Orlando, Florida, 2000, pp. 295–304.

[51] L. Marzario, G. Lipari, P. Balbastre, A. Crespo, Iris: A new reclaiming algorithm for server-based real-time systems, in: Proceedings of the 10th IEEE RTAS, Toronto, Canada, 2004, p. 211.

[52] M. Caccamo, G. C. Buttazzo, D. C. Thomas, Efficient reclaiming in reservation-based real-time systems with variable execution times, IEEE Transactions on Computers 54 (2) (2005) 198–213.

[53] C. Lin, S. A. Brandt, Improving soft real-time performance through better slack reclaiming, in: Proceedings of the 26th IEEE RTSS, 2005, pp. 410–421.

[54] L. Nogueira, L. M. Pinho, Building adaptable, qos-aware dependable embedded systems, in: Proceedings of the 3rd International Workshop on Dependable Embedded Systems, Leeds, United Kingdom, 2006, pp. 72–77.

[55] L. Nogueira, L. M. Pinho, Dynamic adaptation of stability periods for service level agreements, in: Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, Sydney, Australia, 2006, pp. 77–81.

[56] L. Nogueira, 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, Turin,Italy, 2008, p. 97.

[57] R. Barbosa, L. M. Pinho, Mechanisms for reflection-based monitoring of real-time systems, in: Work-In-Progress Session of the 16th ECRTS, 2004.

[58] L. M. Pinho, L. Nogueira, R. Barbosa, An ada framework for qos-aware applications, in: Proceedings of the 10th Ada-Europe International Conference on Reliable Software Technologies, York, UK, 2005, pp. 25–38.

[59] K. Nakazono, Frame rate as a qos parameter and its influence on speech perception, Multimedia Systems 6 (5) (1998) 359–366.

[60] C. Lee, J. Lehoczky, D. Siewiorek, R. Rajkumar, J. Hansen, A scalable solution to the multi-resource qos problem, in: Proceedings of the 20th IEEE Real-Time Systems Symposium, 1999, pp. 315–326.

[61] F. van Harmelen, A. ten Teije, Describing problem solving methods using anytime performance profiles, in: Proceedings of ECAI'00, Berlin, 2000, pp. 181–186.

[62] K. Fukuda, N. Wakamiya, M. Murata, H. Miyahara, Qos mapping between user's preference and bandwidth control for video transport, in: Proceedings of the 5th International Workshop on Quality of Service, New York,USA, 1997, pp. 291–302.

[63] V. Goebel, T. Plagemann, Mapping user-level qos to system-level qos and resources in a distributed lecture-on-demand system, in: I. C. Society (Ed.), Proceedings of The 7th IEEE Workshop on Future Trends of Distributed Computing Systems, 199, p. 197.

44

[64] H. Berthold, S. Schmidt, W. Lehner, C.-J. Hamann, Integrated resource management for data stream systems, in: Proceedings of the 2005 ACM Symposium on Applied Computing, ACM Press, 2005, pp. 555–562.

[65] N. Pereira, E. Tovar, B. Batista, L. M. Pinho, I. Broster, A few what-ifs on using statistical analysis of stochastic simulation runs to extract timeliness properties, in: Proceedings of the 1st International Workshop on Probabilistic Analysis Techniques for Real-Time Embedded Systems, Pisa, Italy, 2004.

[66] A. M. Law, W. D. Kelton, Simulation modeling and analysis, 3rd Edition, McGraw-Hill, 2000.

[67] M. Matsumoto, T. Nishimura, Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator, ACM Transactions on Modeling and Computer Simulation (TOMACS) 8 (1) (1998) 3–30.

[68] E. C. S. Laboratory, Open source erlang, Available at http://www.erlang.org/.

[69] S. Zilberstein, Operational rationality through compilation of anytime algorithms, Ph.D. thesis, Department of Computer Science, University of California at Berkeley (1993).