IPP HURRAY!

www.hurray.isep.ipp.pt

Technical Report

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

Luis Nogueira

Miguel Pinho

TR-060105

Version: 1.0

Date: January 2006

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

Luis Nogueira, 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: {luis, lpinho}@dei.isep.ipp.pt

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 perform services with specific Quality of Service constraints, particularly in dynamic distributed environments where the characteristics of the computational load cannot always be predicted in advance. Our work addresses this problem by allowing resource constrained devices to cooperate with more powerful neighbour nodes, opportunistically taking advantage of global distributed resources and processing power. Rather than assuming that the dynamic configuration of this cooperative service executes until it computes its optimal output, the paper proposes an anytime approach that has the ability to tradeoff deliberation time for the quality of the solution. Extensive simulations demonstrate that the 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, with an overhead that can be considered negligible.

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 perform services with specific Quality of Service constraints, particularly in dynamic distributed environments where the characteristics of the computational load cannot always be predicted in advance. Our work addresses this problem by allowing resource constrained devices to cooperate with more powerful neighbour nodes, opportunistically taking advantage of global distributed resources and processing power. Rather than assuming that the dynamic configuration of this cooperative service executes until it computes its optimal output, the paper proposes an anytime approach that has the ability to tradeoff deliberation time for the quality of the solution. Extensive simulations demonstrate that the 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, with an overhead that can be considered negligible.

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

[*] Corresponding author.
Email addresses: `luis@dei.isep.ipp.pt` (Luís Nogueira), `lpinho@dei.isep.ipp.pt` (Luís Miguel Pinho).

1 Introduction

Several real-time applications have the ability to provide higher or lower levels of quality based on the amount of available resources. However, an increasing number of those applications need a considerable amount of computation power and are pushing the limits of traditional data processing infrastructures as they introduce heavy resource requirements on the client side [1].

This calls for an architecture that supports the distribution of the processing task to different nodes in order to meet acceptable non-functional requirements such as timeliness, robustness, dependability, performance, etc. This is where Quality of Service (QoS) applies and the reason for the increment on work on QoS management during the last years.

The possibility to partition resource intensive services into tasks and distribute them across the subset of neighbours that provide service closer to each user's QoS preferences can benefit a wide range of application domains as an increasing number of real-time applications need a considerable amount of computation power. Consider, for example, the real-time stream processing systems described in [2; 3; 4]. 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. The challenges become even more critical when coordinated content analysis of data sent from multiple sources is necessary [3]. Thus, with a potentially unbounded amount of stream data and limited resources, some of the processing tasks may not be satisfyingly answered [4].

Our work addresses these increasingly complex demands on resources and performance requirements, reflected in multiple attributes over multiple quality dimensions in dynamic distributed systems where the characteristics of the computational load cannot always be predicted in advance, by allowing resource constrained devices to cooperate with more powerful (or less congestioned) neighbour nodes, opportunistically taking advantage of global distributed resources and processing power. This will be achieved via the formation of temporary groups of individual nodes, which, due to their higher flexibility and agility, are capable of effectively respond to new, challenging, requirements. We call these groups *coalitions*.

Furthermore, since different users of the same application may have different service preferences or device-related constraints, supporting the maximisation of those preferences while offloading service execution is a key issue. This paper proposes a method for finding and selecting the best subset of service providers among the set of neighbour nodes according to each user's particular QoS preferences, taking advantage of global resources and processing power through a cooperative service execution.

It is clear that such distribution presents very significant challenges, particularly at the architectural level. Distribution in a heterogeneous environment demands a global approach to achieve efficient resource usage that constantly adapts to devices' specific constraints, nature of executing tasks (hard real-time, soft real-time, non real-time) and dynamically changing system conditions. Major developments are required in the fields of communications protocols, data processing and application support. Our goal is to develop an architecture which enables the creation of a new generation of nodes that can effectively network together, providing a flexible platform for the support of distinct network applications.

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 heterogeneous neighbour nodes. Such scenarios may prevent the possibility of computing optimal resource allocations before execution. Instead, nodes should negotiate partial, acceptable service proposals that can be latter refined if time permits. Moreover, taking the cost of decision-making into account is not an easy task, since the optimal level of deliberation varies from situation to situation. It is therefore beneficial to build systems that can tradeoff computational resources for quality of results.

This paper reformulates the distributed resource allocation problem as an optimisation problem in which there are a range of acceptable solutions with varying qualities, extending the work reported in [5]. The proposed distributed service allocation adapts itself to the available time that is dynamically imposed as a result of emerging environmental conditions and exploits the fact that the nodes within the system are cooperative. The QoS optimisation 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 algorithms increase. This ability to create temporary solutions and incrementally improve them both locally and globally allows the integration of unbounded computations into real-time systems.

Meeting non-functional requirements while providing support for an efficient execution of complex applications in dynamic real-time systems is very challenging. The system must guarantee predictable performance under specified load and failure conditions and ensure graceful degradation when those conditions are violated. This is strictly related to the capacity of controlling the incoming workload, preventing abrupt and unpredictable degradations and achieving isolation among services. The interaction between the proposed anytime algorithms and an efficient overload control is discussed in detail in [6].

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

Since in a large number of applications the quality of provided service directly depends on the available amount of resources, providing support for an efficient execution of complex applications on resource constrained devices is one of the most interesting challenges in real-time systems.

Recent work in computation offloading proposes task partition/allocation schemes that allow the computation to be offloaded, either entirely or partially, from resource constrained devices to a more powerful neighbour [7; 8; 9; 10; 11; 12; 13; 14]. The authors 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 objective is to reduce computation time and energy consumption by monitoring different resources, predicting the cost of local execution and that of a remote one and deciding between local and remote execution.

The computation offloading approach makes it feasible to have a set of heterogeneous computing devices working in a cooperative manner to tackle the resource limitation problem. 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 to offload computation. Furthermore, a general cooperative approach has only recently been addressed considering both performance and users' QoS preferences maximisation [15; 16]. Our preliminary work proposes a system where heterogeneous nodes organise themselves into a coalition for cooperative service execution, dictated by computational capabilities. The proposed algorithms for coalition formation and service proposal formulation return the best possible solution and are sound and complete. Both algorithms iterate over a set of candidates C to produce the best possible solution S, applying an evaluation measure $eval(c_i)$ to each individual candidate c_i.

However, most of today's real-time systems are required to work in dynamic environments, where the characteristics of the computational load cannot always be predicted in advance. Nevertheless, response to events still have to be provided within precise timing constraints in order to guarantee a desired level of performance. This paper relaxes the assumption that the algorithms can have all the time required to compute their outputs and improves our distributed service allocation model with the ability to adapt itself to dynamically changing system conditions, introducing flexibility in the execution times of the proposed algorithms.

For large and complex problems, finding the best possible solution may take a long time and a sub-optimal solution that can be found quickly may be more useful. 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. This idea has been formalised using the concepts of imprecise computation and anytime algorithms.

Liu et al.[17] 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 [18].

Anytime algorithms [19; 20; 21] 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.

Changes in the availability of resources introduce a problem for a distributed service allocation, since the optimal (or even a good) mapping of the application depends on the system and its status. As such, the needed time to find the best possible solution for a cooperative service execution can only be determined at run time. We propose anytime algorithms for a time-bounded distributed QoS-aware service configuration in heterogeneous cooperative environments that takes into consideration users' expressed preferences on the provided level of service.

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 is 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 a fixed relationship between the provided level of service and resource requirements. A QoS manager must react to these dynamic changes in the environment, adjusting the level of provided service and reallocating resources efficiently.

The concept of online admission control has been applied to resource reservation for dynamically arriving tasks. Several efforts appear in the context of real-time operating systems research. Relevant work can be found in [22; 23; 24; 25].

Rather than changing the operative system or investigating any particular application-dependant QoS policy we consider the design of adaptive QoS middleware services on top of best-effort operating systems. The proposed generic framework allows the definition of desired QoS parameters and how these parameters may be degraded in case of overload.

In [26] the authors propose a mechanism for QoS negotiation as a way to ensure graceful degradation in cases of overload, failures, or violation of pre-runtime assumptions. They suggest that a user should be able to express in his service request the spectrum of acceptable QoS levels, as well as a quantitative perceived utility of receiving service at each of those levels, which are statically mapped to certain quality choice combinations. We share the need of allowing each user to specify his own QoS preferences but propose a more natural and realistic way to base a service request on a qualitative measure. Our service negotiation protocol dynamically determines the best possible compromise on QoS and its relative utility based on user's specified preferences.

An architecture for supporting the adaptive management of multiple resources on a general purpose operative system is introduced in [27]. The work extends a prior architecture for adaptation of the CPU bandwidth for QoS control [28].

The QoS-based Resource Allocation Model (Q-RAM) [29; 30; 31] is an analytical approach for satisfying multiple QoS dimensions in a resource-constrained environment. System resources are apportioned across multiple applications such that the net utility that accrues to the end-users of those applications is maximised. The static resource allocation algorithms of Q-RAM have been extended to support a dynamic task traffic model [32] and to handle non-monotonic dimensions [33]. Further improvement techniques to reduce the computation complexity of the initial proposal and their application to radar tracking are described in [34].

Our approach to deal with a large number of dynamic tasks, multiple resources, and real-time operation constraints is different. We consider a collaborative environment populated with nodes with a significant degree of autonomy, capable of performing tasks and sharing resources with other nodes. When the specific resource demand imposed by the user's QoS preferences cannot be satisfied by a single node, the proposed framework promotes the cooperation between neighbours to fulfil that resource demand. Furthermore, the formation of such a coalition is influenced by each user's specific QoS preferences trying to maximise the utility for each particular user rather than the sum of the utilities of all users. Most important, due to the high complexity of an optimal distributed service allocation both global and local QoS optimisations are performed by anytime algorithms, imposing time constraints to these computations.

3 Problem description and system model

Consider a distributed system formed by several nodes with a set of shared resources R, where real-time and non real-time applications co-exist. Such an environment is expected to be heterogeneous, consisting of nodes with several resource capabilities. For some of those there may be a constraint on the type and size of applications they can execute with users' required quality of service.

Therefore, this work addresses a distributed cooperative execution of resource intensive services in order to maximise each user's satisfaction with achieved QoS, addressing the increasing demands on resources and performance. Nodes may cooperate either because they can not deal alone with resource allocation demands or because they can reduce the associated cost of service execution by working together.

Each service is assumed to have a set of parameters that can be changed to configure its QoS and resource demands. Each subset of parameters that relates to a single aspect of service quality is named as a QoS dimension. Let Q be the set of user's QoS constraints associated with service S. Each Q_{kj} is a finite set of quality choices for the j^{th} attribute of dimension k. This can be either a discrete or continuous set.

There will be a set of independent[1] tasks T to be cooperatively executed, resulting from partitioning a resource intensive service S. Correct decisions on service partitioning must be made at run time when sufficient information about workload and communication requirements become available [7], since computation workload and communication cost may change with different execution instances and different users. The underlying QoS-aware framework negotiates with neighbour devices a cooperative execution of the resource intensive application and selects a subset of those neighbours to form a coalition. The basic coalition formation problem can be described as:

> Given a set of nodes N and a resource allocation demand enforced by Q they have to satisfy, if the resource demand cannot be satisfied by a single node or when a single node handles the request inefficiently, the nodes should cooperate to fulfil the resource demand. The selection of a subset of N to perform a cooperative service execution of S should be influenced by Q, maximising user's satisfaction with provided service.

Various groups of nodes may have different degrees of efficiency in tasks' execution performance due to different capabilities of their members and their current state. Coalition's members selection is determined by the proximity of service proposals with respect to expressed user's multi-dimensional QoS constraints. As such, nodes must be enhanced with the ability to propose and evaluate multi-dimensional service proposals taking user's constraints into account.

Rather than assuming that the 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 available heterogeneous neighbours. For large and complex problems, finding the best possible solution may take a long time and a sub-optimal solution that can be found quickly may be more useful.

The participants in this anytime coalition formation process are the user's device and the subset of neighbour nodes that are capable to offer service at least with the minimum quality level requested by the user. The user's node, playing the role of *Organiser*, starts and guides all the negotiation process, broadcasting service requirements and evaluating received service proposals sent by *Respondent* neighbours willing to belong to the coalition.

The Organiser node models each negotiation process through the tuple $Neg^{Org} = \{S, Q, L, H\}$, where S identifies the service under negotiation, Q the user's quality constraints associated to service S, L the list of neighbours that can provide service S according to Q, and H is the negotiation's history.

[1] Dependence relations will be addressed in future work

Each element of H groups information related to a single task T_i of service S as $H_i = \{Prop_{ki}, Eval_{ki}\}$, where $Prop_{ki}$ is the service proposal sent by node N_k for task T_i, and $Eval_{ki}$ is the evaluation value achieved by that proposal according to the user's QoS preferences.

Each Respondent node N_k models the negotiation process through the tuple $Neg^{Rsp} = \{T_i, Q, O, C\}$, where T_i is the description of a task of service S, Q are the user's quality constraints associated to service S, O identifies the Organiser agent that started the negotiation process, and C is the comment sent by the Organiser node to proposal $Prop_{ki}$. This comment indicates if the proposal was or was not selected for the cooperative service execution under negotiation. Until the reception of this comment, the resource quantities reserved for the formulation of service proposal $Prop_{ki}$ must be considered as unavailable and cannot be used for the formulation of other service proposals in concurrent negotiations.

This paper is focused in the dynamic formation of cooperative coalitions, taking the maximisation of each user's specific QoS constraints into account. The reader should refer to [2; 10; 3; 8; 7] as an example of work on applications' partitioning schemes and to [29; 35; 36; 37] for representative works on interpretation of QoS constraints and consequent mapping on resource quantities. Both subjects are outside the scope of this paper.

4 Cooperative service execution framework

The objective of the proposed framework is to enable resource-constrained devices to solve computationally expensive services by redistributing parts of the service onto other devices, maximising user's satisfaction with provided service. 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 user's node capabilities and 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 distribution of the resource intensive service's tasks across a community of nodes, forming temporary coalitions for a cooperative service execution that maximise users' QoS constraints. Figure 1 presents the structure of the proposed framework, running on every node of the network.

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

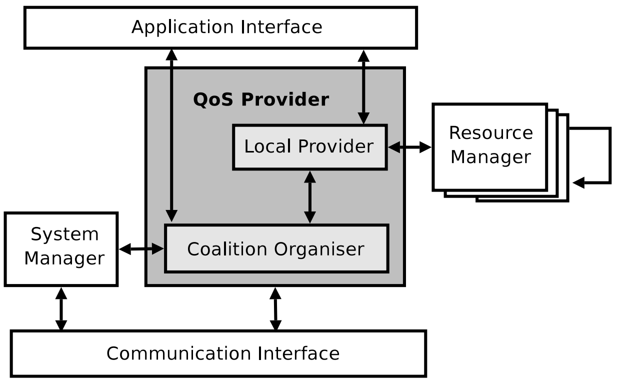

Fig. 1. Framework structure

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 distributed resource requests. Rather than reserving local resources directly, it contacts the *Resource Managers* to grant specific resource amounts to the requesting task.

Within the QoS Provider, the *Coalition Organiser* is responsible for the coalition formation process, atomically distributing service requests, receiving individual nodes' proposals and deciding which node(s) will provide the service. 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 *Local Provider* is responsible for replying to service requests with service configuration proposals, and for maintaining the state of node's resource allocations and services provided.

The *System Manager* maintains the overall system configuration, detecting nodes entering and leaving the system, manages coalition operation and its dissolution.

Each *Resource Manager* is a module that manage 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 by software that manages other resources (such as memory).

One important characteristic of the framework is the ability of resource managers 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.

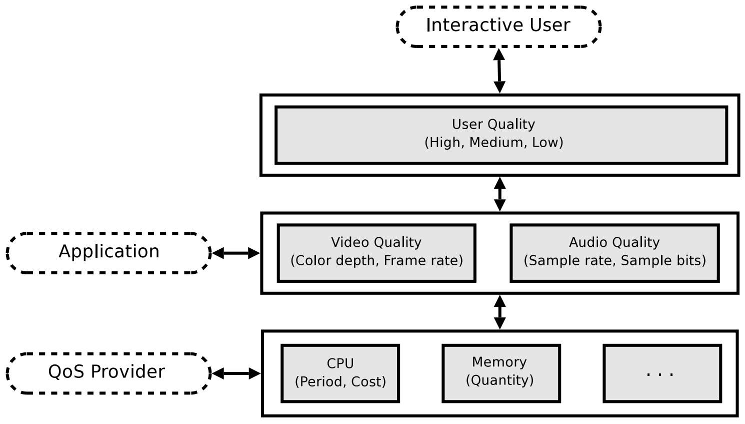

Fig. 2. Resource managers' layering

Although we consider a collaborative environment, proper resource usage must be monitored in run time [38], in order to decide based on actual resource usage of the system and not only on the resource usage assumptions of requesting services. Then, the QoS adaptation mechanism should apply users' preferences to system's resource usage information.

5 Expressing quality of service

There may be in the system several instances of an application used by many different users. There is an increasing need for customisable services that can be tailored to each user's specific QoS requirements [4]. Furthermore, user's QoS requirements may even change throughout a session and the user should be given the opportunity to make informed decisions about application's adaptation [39].

Unfortunately, in most systems users do not have any real influence over the QoS they can obtain or how their applications will adapt to environmental changes, since service characteristics are fixed when the systems are initiated. Since QoS is often multi-dimensional an user (or application) might want to make some quality tradeoff, especially when available resources are scarce. Therefore, it is to the user's advantage to be able to specify his own QoS requirements using an interface that allows implicit or explicit quality tradeoffs.

Adopting a common QoS description scheme guarantees information consistency and compatibility for 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. Knowledge representation becomes an important issue in the context of QoS-aware cooperative service execution as well.

The definition of a generic QoS scheme must include dimensions, attributes and values, as well as relations that map dimensions to attributes and attributes to values. It also must be extensible to support the later addition of news terms and relations.

Such a scheme can be represented 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 dependencies between attributes' values. A dependence between attribute i and attribute j is represented as $Dep_{ij} = f(Val_{ki}, Val_{kj}), \forall Attr_i, Attr_j \in Attr$.

Each application domain has its own QoS requirements. Using the generic QoS scheme it is possible define the set of quality dimensions that are associated with any particular application. As an example of this QoS description, a video streaming application may define the following set of QoS dimensions (the following list is not intended to be exhaustive):

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

DA Video Quality = {color depth, frame rate}
DA Audio Quality = {sampling rate, sample bits}

AV color depth = {1,3,8,16,24}
AV frame rate = {[1,30]}
AV sampling rate = {8,16,24,44}
AV sample bits = {8,16,24}
```

Having 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 about service provisioning. Such a service configuration implies taking into account multiple quality attributes.

Attaching utility values to different issues helps to solve the problem of multi-issue evaluation. However, it may be clearly infeasible to make the user specify an absolute value for every quality choice. While we want a semantically rich request to try to achieve a service closely related to user's preferences, we also want that a user be actually able to express his preferences in a service request. A more natural and realistic way is to simply impose a service request based on a qualitative, not quantitative, measure. As such, a preference order is imposed over the dimensions, its attributes and their values in every user's service request. The relative decreasing order of importance imposed in dimensions, attributes and values expresses user's preferences, that is, elements identified by lower indexes are more important than elements identified by higher indexes.

Suppose that, in a remote surveillance system, video is much more important to a particular user than audio. Assuming that for that user a grey scale, low frame rate is fine for video, and he does not demand any dependence between the values of the presented attributes, his request could be as follows:

```
1. Video Quality
  (a) frame rate  : {[10,7], [6,4]}
  (b) color depth : {3, 1}
2. Audio Quality
  (a) sampling rate : {16, 8}
  (b) sample bits   : {16, 8}
```

In the example above, video is more important than audio, and frame rate is more important than colour depth in the Video Quality dimension. In a similar way, the audio sampling rate is more important than the sampling size. For each of these attributes, a preference order for the QoS values is as well expressed. This way, preferences are clearly expressed without the need to quantify with absolute terms every quality tradeoff.

Note that it is possible to define multiple intervals in decreasing preference order for the user's acceptable values of a particular attribute. The same is possible for the QoS characterisation of a particular application domain. The evaluation of user's acceptability of each service proposal presented in the next section can deal with this type of attributes' definition.

6 Coalition formation

A 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 user's QoS requirements associated with a resource intensive service. By best group, we mean the group formed by those nodes who offer service closer to user's QoS preferences expressed in his request.

A service request is considered to be formulated through the relative decreasing importance ($K = 1 \dots n$) of a set of n QoS dimensions. Furthermore, for each dimension a relative decreasing importance order of attributes is also specified ($i = 1 \dots attr_k$), 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.

Given a user's service request on node N_i for the execution of a resource intensive service S with specific user's QoS constraints Q_i that cannot be locally satisfied, the *QoS Provider* broadcasts the description of each task T_i that can be remotely executed, user's QoS constraints Q_i as well as a timeout Δ_t for proposals reception. Each T_i can be determined by a task partition/allocation scheme that dynamically considers the tradeoff between local execution requirements and communication costs.

Every node N_j which is able to satisfy the request, formulates a proposal according to a local QoS optimisation algorithm (see Section 7), and replies to node N_i with proposal P_{ji} and its local reward W_j, resulting from its proposal acceptance. For now, it is suffice to say that local reward R_k is an indicator of node's local QoS optimisation, 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.

Each admissible proposal[2] P_i is evaluated according to user's QoS preferences specified in relative decreasing order in his service request. For each QoS dimension, Eq. 1 determines an weighted sum of the differences between user's preferred values and the values proposed by a specific node to that dimension's attributes evaluates proposal's distance to user's request.

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

where n is the number of QoS dimensions and $0 \leq w_k \leq 1$ is the relative

[2] A proposal is admissible if it can satisfy all QoS dimensions at least with the minimum level requested by the user

importance of QoS dimension k, Q_k, to the user, and can defined as

$$w_k = \frac{n - k + 1}{n} \tag{2}$$

In Eq. 1, QoS dimensions are presented in relative decreasing order of importance to the user. This order is specified in user's service request, encoding user's preferences in a qualitative way.

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

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

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

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 user's preferred value $Pref_{ki}$ and is defined as

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

If attribute i has a continuous domain, this quantification is a normalised difference between the proposed value and the preferred one.

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. In [30] the authors use the notion of *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 specification into $Prop_{ki}$'s and $Pref_{ki}$'s scoring values.

When the definition of the possible values for some attribute of QoS dimension Q_k considers a set of intervals, Q_k in Eq. 4 must relate to the particular interval where $Prop_{ki}$ is found. In the same way, if the user's acceptable values of an attribute of dimension Q_k considers a set of intervals, $Pref_{ki}$ should be the first value on the $Prop_{ki}$'s interval and the relative importance of that interval on user's expressed preferences in decreasing order must be considered, similarly to what is done to all the attributes and dimensions of user's request.

The best proposal is thus the one that presents the lowest distance to user's preferences, in all QoS dimensions, since it is the one that contains the attributes' values more closely related to user's request.

6.1 Anytime global QoS optimisation

An anytime approach to the coalition formation problem implies trying to quickly find a good initial solution and gradually improve that solution if time permits. Furthermore, the increase in solution's quality should be maximised at each iteration. Relying in the order of proposals' reception and perform a sequential evaluation does not seems to be an effective approach. The selection from the set of received service proposals of the next candidate proposal for evaluation should be optimised to achieve the desired goal of maximising the expected improvement in solution quality.

We propose to use the local reward achieved by a node as an heuristic to guide the coalition formation process. Nodes with higher local reward have a higher probability to be offering service closer to this particular user's request under negotiation. For each task $T_i \in S$, the next candidate proposal P_{ki} 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 anytime coalition formation algorithm, seeking distributed QoS optimisation by selecting those nodes that offer service closer to user's preferred QoS values, is described in Algorithm 1. Since the formation of a coalition is aimed at maximising the benefits associated to a cooperative service execution, the quality of each generated coalition can be 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

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

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 tasks T_i of service S.

After building an initial coalition, the algorithm continues, if time permits, to evaluate received proposals trying to improve the quality of the current solution. Some other node, while achieving a lower local reward, can still propose a better proposal for the specific user's service request under negotiation.

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
```

Node's local reward is also used to improve load balancing between cooperative neighbours. 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 one of those nodes is selected for participating in the coalition. However, if one of them has a lower local reward, it means that the previous set of tasks being locally executed had to suffer an higher local QoS degradation in order to accommodate this new task. Selecting the node with a higher local reward from two similar service proposals, not only maximises service for a particular user, but also maximises 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

Coalition formation's anytime behaviour can be formally described using the set of axioms proposed in [40]. The authors propose to 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 service execution*

The algorithm does not immediately produces an intermediate solution, since it must first analyse a proposal for each task T_i of service 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*

A coalition is only updated if a better proposal for any task T_i of service S is found.

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

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

The solution 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 [15] 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 [21] the desired properties of anytime algorithms include the following features: *measurable quality* that can be determined precisely, *recognisable quality* that can be easily determined at run time, *monotonicity* of the quality of the result, *consistency* of the quality with respect to computation time and input quality, *diminishing returns* of the quality over time, *interruptibility* of the algorithm at any time and *preemptibility* of the algorithm with minimal overhead.

The conformity of the proposed anytime coalition formation algorithm is checked in the next paragraphs.

Property 6.3.1 (Measurable quality) *The quality of a coalition can be determined precisely*

Proof: According to Eq. 6 the quality of the proposed coalition can be calculated directly from the evaluation values of the best proposals found for each service's task, after each iteration of the algorithm.

□

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 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 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.

□

Property 6.3.3 (Monotonicity) *The quality of the generated coalitions 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 user's preferences than the best proposal determined so far for task T_i.

The coalition formation algorithm always returns the coalition formed by the best proposals evaluated until time t rather than the last evaluated service proposals. According to Zilberstein [21], this characteristic in addition to a recognisable quality is sufficient to prove the monotonicity of an anytime algorithm.

□

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 candidate proposal P_{ki} for evaluation for each task T_i of service 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.

□

Property 6.3.5 (Diminishing returns) *The improvement in the quality of the generated coalitions 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 of service S being negotiated for cooperative service execution. 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 cooperative service execution being currently negotiated.

Selecting for evaluation, for each task T_i of service S, the candidate proposal sent by the node that presents the higher local reward, indicating that it is offering service closer to the majority of requested QoS values, rapidly improves the quality of the generated coalition at an early stage of execution.

Solution's quality can further improve, but at a lower rate, in the next steps of the algorithm. Since the objective is to find the best coalition for this particular user's request under negotiation, the algorithm continues to evaluate received proposals as some other node may propose a better service for this user at the expenses of previously guaranteed services, and, as such, achieving a lower local reward.

The actual concavity of the coalition formation is empirically evaluated in the next section.

□

Property 6.3.6 (Interruptibility) *The algorithm can be stopped at any time and provide a coalition for cooperative service execution*

Proof: When stopped at time t the algorithm returns the coalition formed by the best evaluated proposals for each task T_i of service S under negotiation.

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

Each iteration of the algorithm forms a contract algorithm. As such, a new coalition can only be generated at the end of a new iteration.

□

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 generated coalition it can be easily resumed after an interrupt.

□

7 Service proposal formulation

All entities that participate in a cooperative QoS-aware service execution negotiation must provide sufficient resources to propose a service configuration that respects the QoS requirements expressed in user's request. It is therefore the responsibility of each individual QoS Provider (Figure 1) to map user's QoS constraints to local resource requirements, and then reserve resources accordingly (resource reservations are made through Resource Managers). This paper is focused in an automatic time-bounded coalition formation and do not deals 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, 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, QoS Provider executes the local QoS optimisation algorithm described in Algorithm 2. In order to make a proposal, each QoS Provider contacts the required Resource Managers for resource availability. Conventional admission control schemes either guarantee or reject each 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 graceful degradation.

A specific service configuration proposed for a task T_i will obtain a reward r_{T_i} determined by the proximity of the proposal with respect to the QoS preferences specified in the user's service request.

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

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

An utility model for QoS control is also used in [26]. The authors also suggest that users should be able to express in their service requests the spectrum of QoS levels they can accept from a service provider. However, the user must specify in his request a set of pre-defined QoS levels as well as the achieved reward of providing service in one of those levels. Here, the proposed QoS level is dynamically built according to user's accepted values for each QoS dimension and local resources' availability. As such, the reward of executing a task at a specific QoS level depends on the number, and relative importance, of the QoS dimensions being served closer to user's requirements.

Along with the service proposal each node sends a measure of global satisfaction resulting from its proposal acceptance. The local reward R_j expresses the degree of global satisfaction for all the users that have tasks being executed by a particular node N_j, with specific QoS levels. The reward of each task being locally executed is combined to determine the global satisfaction of the proposed solution. For a node N_j, the local reward R_j of a set of service proposals is given by

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

Unless all tasks are executed at their highest QoS level, there is a difference between the actual total local reward achieved by the current QoS levels selected and the maximum possible 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. Selecting the node with higher local reward for similar service proposals, not only maximises service satisfaction for a particular user, but also maximises global system's utility, since a higher local reward clearly indicates that the previous set of tasks being locally executed had to suffer less QoS degradation in order to accommodate the new task.

In [30], it was demonstrated that the QoS optimisation problem involving multiple resources and multiple QoS dimensions is NP-hard. An optimal solution based on dynamic programming and an approximation scheme based on a local search technique was presented. However, the computation time needed to find an optimal solution can reduce the overall utility of the system. In addition, the deliberation cost is dependent on local resources' availability and user's QoS constraints. Therefore, it is beneficial to build systems that can trade the quality of results against the cost of computation [21].

7.1 Anytime local QoS optimisation

The proposed anytime algorithm, detailed in Algorithm 2, clear splits the formulation of a new proposal for service execution in two different scenarios. The first one involves guaranteeing the new task without changing the level of service of previously guaranteed tasks. The second one, due to node's overload, demands service degradation in existing tasks in order to accommodate the new requesting task. The local QoS optimisation (re)computes the set of QoS levels for all local tasks, including the new requested one. Offering QoS degradation as an alternative to task rejection has been proved to achieve higher perceived utility [26].

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

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

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. As such, the reward of the initial service configuration for the new task is low (the exact value is determined by the penalty factors used in a particular system), affecting node's local reward. On the other hand, the impact of this new task on the provided level of previously existing tasks is inexistent. Also, this initial solution is the service configuration that has a higher probability of being feasible, considering the new task. 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 solution's quality.

At each iteration, the algorithm produces a new service configuration that may not be feasible due to local resources availability and user's QoS constraints for the new task. 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. However, each intermediate configuration, even if not feasible, is used to calculate the next solution, since it is one step closer of the next solution.

When it is not necessary to degrade the QoS of previously existing tasks to provide service to the new request, 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 (including the new arrived one).

The algorithm always improves or maintains the quality of the solution as it has more time to run. This is done by keeping the best feasible solution so far, if the result of each iteration is not always proposing a feasible set of tasks.

The algorithm terminates when the time for the reception of proposals has expired (this time is sent in user's 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,

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 $j^{th}$ 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 at-
        tribute $x$ to the $m + 1$'s level
      end for
    end while
  end while
  return service configuration for new 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 proposed in [40].

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

An intermediate solution can only be considered if it produces a feasible set of tasks. 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 tasks can only improves over time*

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

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

Axiom 7 (Growth rate) *The amount of increase in the quality achieved by a feasible set of tasks varies during computation*

The solution's quality rapidly increases in the first steps of the algorithm and its growth rate diminishes over time. When there are spare resources the algorithm starts to improve user's preferred attributes. On the other hand, when QoS degradation is needed in the search for a new feasible solution, the algorithm starts to degrade users' less important attributes.

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

Axiom 8 (End condition) *When is not possible to improve the quality of the current service proposal the algorithm achieves its full functionality*

When runs to completion, the anytime version of the algorithm will produce exactly the same solution quality as its traditional version [15] that only produces a solution with quality Q'_conf at the end of computation. The algorithm terminates 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.

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 with the desirable properties of anytime algorithms [21] is checked in the next paragraphs.

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

Proof: At each iteration of the algorithm, Eq. 9 measures the quality of the proposed service configuration by considering the proximity of the proposal with respect to user's request under negotiation and the impact of that proximity on the global satisfaction achieved by the previous existing tasks.

$\square$

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

Proof: The quality of each generated feasible service configuration is determined by using the rewards achieved by all tasks being locally executed as well as the new arrived task. According to Eq. 7 the determination of the reward achieved by each task is straitforward and time-bounded.

A non-feasible service configuration as zero quality.

$\square$

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

Proof: The algorithm produces a new service configuration at each iteration, as it tries to maximise the utility increase for the new task while minimises the utility decrease for all tasks being locally executed 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 service configuration.

According to Zilberstein [21], this characteristic in addition to a recognisable quality is sufficient to prove the monotonicity of an anytime algorithm.

$\square$

Property 7.3.4 (Consistency) *For a given amount of computation time on* *a given input, the quality of the generated service configuration 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 service configuration 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 service configuration 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.

The quality of a service configuration is given by Eq. 9 that takes into consideration the rewards achieved by all tasks, considering proposed and requested values for the several QoS dimensions.

When there are spare resources the initial solution is improved by selecting for upgrade the QoS attribute that maximises user's satisfaction, according to the preferences expressed in his request in relative decreasing order.

On the other hand, when QoS degradation is needed in order to accommodate the new task, the next configuration tested for schedulability is the one that minimises the rewards' decrease achieved by all tasks being locally executed.

The improvement in solution quality is then larger at the early stages of the computation and it diminishes over time.

□

Property 7.3.6 (Interruptibility) *The algorithm can be stopped at any time and provide a service configuration proposal*

Proof: When stopped at time t the algorithm returns the best feasible service configuration generated until time t, according to user's request and local resources availability.

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 service configuration, resulting in zero quality.

Each iteration of the algorithm forms a contract algorithm. As such, a new service configuration can only be generated at the end of a new iteration.

□

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.

□

8 Evaluation

In order to validate the design decisions of our approach we have conducted extensive simulations to evaluate the formation of coalitions for cooperative service execution among a set of distributed heterogeneous nodes, using the anytime algorithms proposed in this paper. Since we are primarily interested in dynamic scenarios a special attention was devoted to introduce a high variability in the conducted simulations' characteristics.

There were 6 different resources, randomly partitioned among the nodes. As a result of this non-equal partition, some nodes could have amounts of some resources which are significantly different from the average, generating a heterogeneous environment. This affected their ability to perform some tasks and has driven nodes to coalition formation for cooperative service execution.

The number of simultaneous users varied from 1 to 100, the number of independent tasks per requested service varied from 5 to 20, and available nodes varied from 10 to 100. At each node, the QoS Provider had a fixed set of mappings between user's requested QoS levels and resource requirements, and then reserved resources accordingly (resource reservations were made through Resource Managers). The domain of the simulations was characterised by 4 QoS dimensions, each with 5 attributes. Each attribute had 6 possible values (continuous or discrete).

At randomly generated times, one or more clients generated new service requests at randomly selected nodes. Each service request had a set of multi-dimensional QoS constraints, ranging from a minimal quality to a randomly generated maximum value. Thus, QoS degradation within users' requested values could be performed in order to accommodate new tasks.

It is known that not much can be concluded with a single simulation run [41]. In fact, the results of a given simulation run are just particular instantiations of random variables that may have large variances. Most of the methods for the analysis of simulation output data rely on the fact that although the simulation results of a single simulation run are not independent, it is possible to obtain independent observations across the results of several simulation runs (or simulation *replicas*) [42]. The replication/deletion method is a fairly simple approach, with a reasonably good statistical performance.

The conducted simulations had two main objectives: analyse the performance profiles of the proposed anytime algorithms and evaluate the computational cost of those algorithms when compared against their traditional versions.

The results reported in the next subsections were observed from multiple and independent simulation runs, with initial conditions and parameters, but different seeds for the random values [3] 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 Evaluating performance profiles

The performance profile of an anytime algorithm denotes the expected output quality as a function of the execution time [21]. Since there are many possible factors affecting the execution time of an algorithm, the performance profile has to be in many cases determined empirically and not analytically. Rather than measuring the absolute execution time of the algorithms on every run of the simulation, we have normalised it with respect to the completion time, which is the minimal time when the expected quality is maximal [44].

In Section 6, we argue that the selection of the next proposal to be evaluated should be done in a order that maximises the expected improvement in solution quality. In the next paragraphs, we compare the proposed heuristic search of a better solution against a sequential evaluation of received proposals.

[3] The random values were generated by the Mersenne Twister algorithm [43].

The recipient nodes managed the coalition formation process by broadcasting the service description, evaluating the set of received proposals formulated by available service providers, and selecting those proposals that were closer to user's preferences, forming a new coalition for a cooperative service execution.

The anytime coalition formation algorithm ran until completion, using both methods of next candidate proposal's selection. The mean values for each percentage of completion time obtained with the set of conducted simulations were used to obtain the performance profile presented in Figure 3.

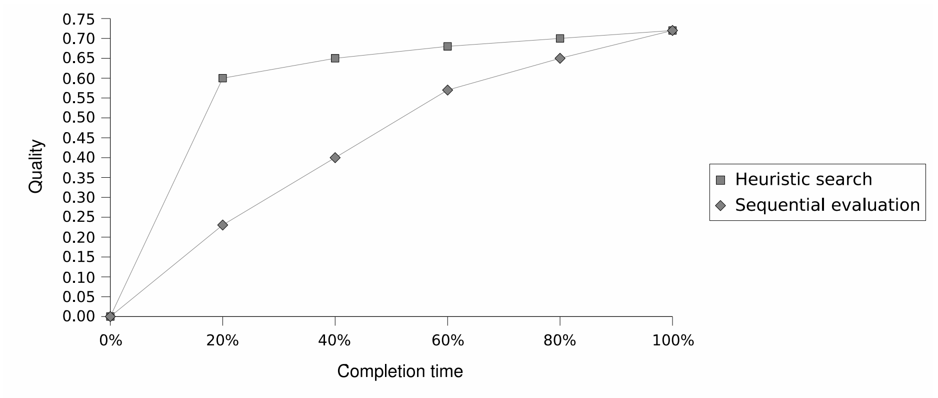

Fig. 3. Performance profile of the coalition formation algorithm

Two important conclusions can be taken, considering the desirable properties of an anytime algorithm reported in [21]. First, the coalition's quality measure is a non-decreasing function of time. Only a better proposal for a specific task $T_i \in S$ updates the coalition formation, increasing its quality. The second conclusion refers to the diminishing returns property. It 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. At only 20% of the completion time, the anytime coalition formation algorithm when using the proposed heuristic selection of the next candidate proposal can achieve a solution's quality of 83% $\pm$ 6% of the optimal solution, with a 99,9% confidence level. On the other hand, when evaluating proposals according to their arrival time, at 20% of the completion time the algorithm achieves a solution's quality of 32% $\pm$ 25%. We see that a sequential evaluation is very sensitive to the order of proposals' reception. The proposed heuristic effectively maximises the expected improvement in solution's quality at a early stage of the needed computation time, effectively resulting in a concave function of time.

Another set of simulations was conducted to analyse the impact generated by the arrival of a new task on the level of provided service of previously existing tasks. Recall from Section 7 that the reward of a specific proposal measures how useful it will be for a specific user with respect to his service request, 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 resources per node is greater than the average amount of resources necessary for each service execution. Figure 5 presents the scenario where the average amount of resources per node is smaller than the average amount of resources necessary for each service execution.

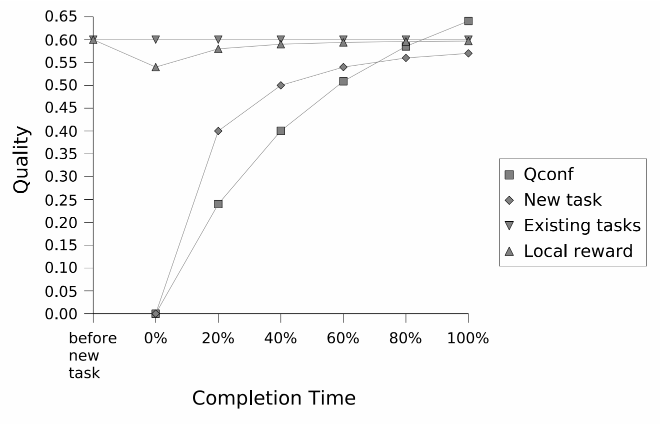

Fig. 4. Expected quality improvement with spare resources

When there are enough resources to improve the initial solution until one of the user's requested levels of service, the quality of the next feasible solution increases as the reward of the new task's service increase. In Figure 4, the increase in solution's quality Q_{conf} results from the increase in the new task's reward, since the average level of service of the previously existing tasks remains the same. This also increases node's local reward, that was affected by the initial proposed solution of serving the new arrived task with the minimal requested QoS level.

Figure 5 shows the algorithm behaviour when QoS degradation is needed in order to provide one of the requested levels of service for the new arrived task. When trying to upgrade the reward achieved by the new task, the generated configuration may result in an unfeasible set of tasks. The algorithm minimises the reward decrease of all tasks when trying to find a new feasible solution that presents a higher satisfaction for the service request under negotiation. It is the responsibility of the coalition formation algorithm to select between proposals with similar evaluation values, those nodes that achieve higher local rewards, promoting load balancing.

One can conclude that the proposed heuristic search for a better feasible solution, optimises the rate at which the quality of the current solution improves in both scenarios. With spare resources (Figure 4), at only 20% of the computation time, the solution's quality for the new arrived task is near 70% $\pm$ 5% of the achieved quality at completion time, with a confidence degree of

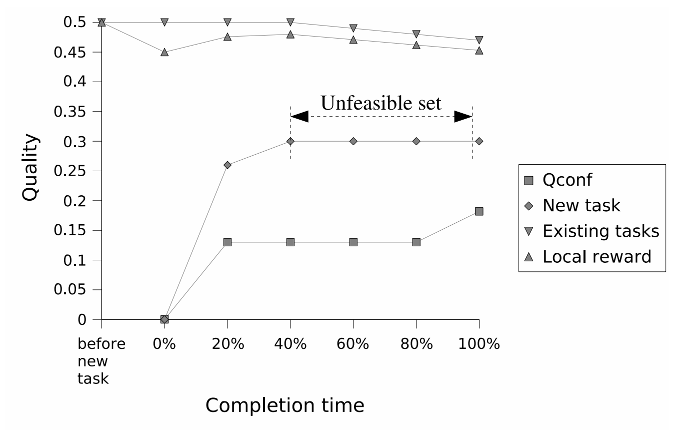

Fig. 5. Expected quality improvement with limited resources

99,9%. When QoS degradation is needed (Figure 5), the service proposal for the new task achieves 87% ± 4% of its final quality at 20% of computation time. The quality of the node's global service solution, identified by $Qconf$ in both figures, also quickly approaches its maximum value at an early stage of the computation. As such, the diminishing returns property is also verified in the proposal formulation anytime algorithm.

Figure 4 and 5 also show that the quality measure of a service proposal 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.

Both algorithms if interrupted before their completion can still provide a solution for cooperative service execution and a measure of its quality. This 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 the cooperative service configuration to provide a solution at any time results in a significant improvement of the framework's behaviour.

8.2 Comparison against the traditional versions of the algorithms

Throughout the paper the idea that complex scenarios may prevent the possibility of computing optimal resource allocations before a cooperative execution among a set of nodes was stated and anytime algorithms that can tradeoff deliberation time for quality of results were proposed. The goal was to introduce flexibility in the execution time of the coalition formation and service proposal formulation algorithms in order to provide adaptation to changing conditions in dynamic heterogeneous environments.

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 [15]. The results discussed in the next paragraphs were normalised with respect to the completion time of the longest solution.

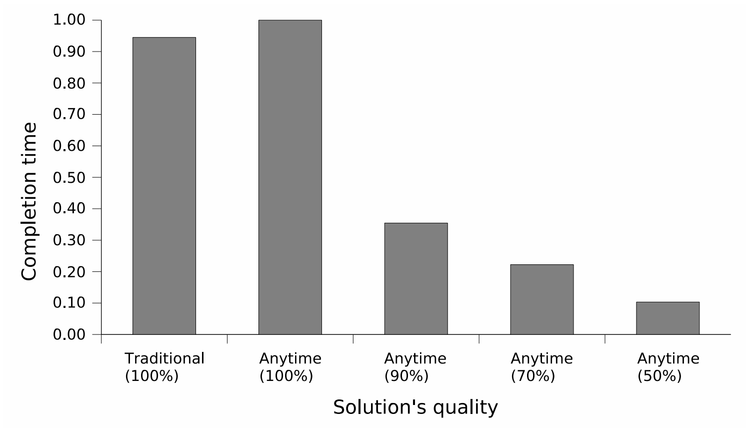

Fig. 6. Coalition formation: Anytime vs Traditional

The achieved results for both the anytime and traditional versions of the coalition formation algorithm, considering the needed time to complete their computations are described in Figure 6. The traditional version of the algorithm, which performs a sequential proposal evaluation, reaches its end within nearly 95% ± 2% of the needed time of the anytime approach, with a 99,9% confidence degree. As such, the heuristic selection of the next service proposal to evaluate, based on each service provider's local reward, has an associated cost. However, and in consonance with the performance profile presented in the previous section, the anytime version needs as little as near 10% of its completion time to achieve a solution's quality of 50% ± 6% of its optimal solution, and below 40% to achieve 90% ± 5% of its optimal solution. As discussed in the previous section, the same confidence cannot be achieved with an evaluation that relies in the order of proposals' reception.

The comparison between the computational cost of the anytime and traditional versions of the service proposal formulation algorithm is resumed in Figure 7 and Figure 8 using the same scenarios of the previous section. Figure 7 presents the scenario where the average amount of resources per node is greater than the average amount of resources necessary for each service execution. Figure 8 presents the scenario where the average amount of resources per node is smaller than the average amount of resources necessary for each service execution.

The discrepancy in the needed computation time to achieve its optimal solution of both versions is explained by the different approach of each version of the algorithm. The anytime version tries to quickly find a feasible solution. Its initial solution considers the worst requested values for all QoS dimensions

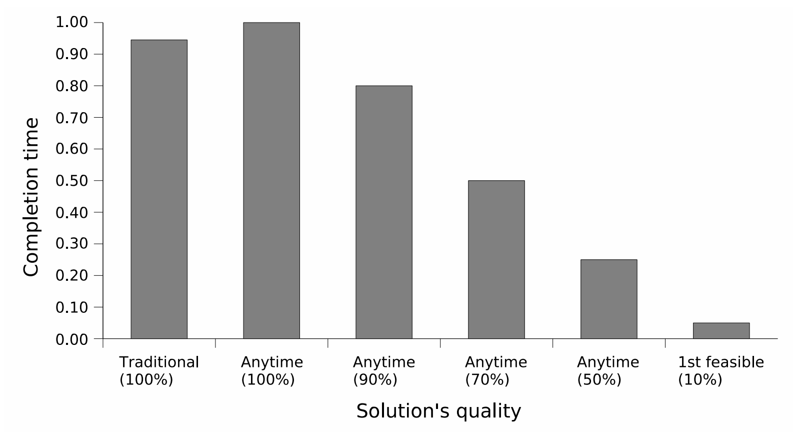

Fig. 7. Proposal formulation: Anytime vs Traditional with spare resources

for the new arrived task as opposed to the traditional version that starts by maximising the requested level of service for the new task. The traditional version stops at its first solution, as this is its optimal one. On the other hand, the anytime algorithm continues to improve achieved feasible solutions until the optimal one is reached.

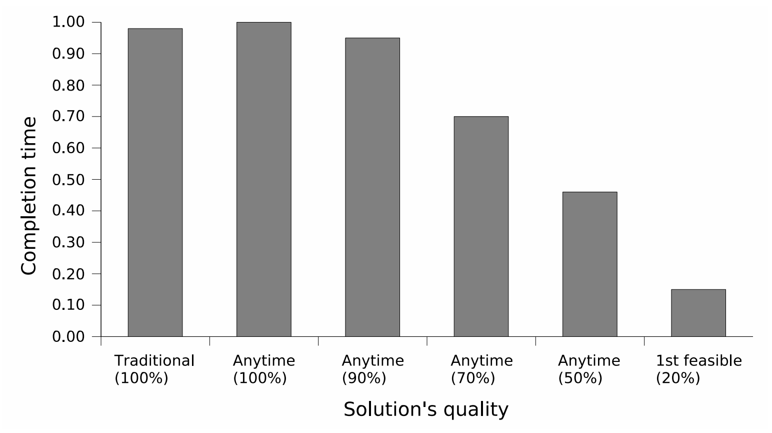

Fig. 8. Proposal formulation: Anytime vs Traditional with limited resources

By analysing Figures 7 and 8 one can conclude that when there are spare resources the traditional version is slightly quicker to find the optimal solution. With limited resources both versions need almost the same time to conclude their computations. However, in both scenarios the anytime version is by far quicker to find a 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 the 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.

9 Conclusions

This paper considers an heterogeneous distributed system, where computing devices may range from small, resource limited mobile devices to stationary powerful servers. The paper addresses a cooperative execution environment, where a resource limited device with a set of tasks that it cannot execute (or handles them inefficiently) may use additional resources offered by neighbour nodes, opportunistically taking advantage of global distributed resources and processing power.

Each of the system's users may have its own QoS preferences as well as each service provider different capacities for service provisioning, that can be expressed by a QoS description scheme that guarantees information consistency and compatibility in a heterogeneous network. The scheme includes dimensions, attributes and values of a particular domain, as well as relations that map dimensions to attributes and attributes to values and their dependencies. Having a QoS characterisation of a particular application domain, users and service providers are able to define service requirements and proposals in order to reach an agreement about service provisioning. To avoid the need to express absolute values for every quality tradeoff, a preference order is imposed in users' service requests.

Finding an optimal distributed service provisioning that deals with both users' and service providers' constraints can be extremely complex and take a long time. For large and complex problems, a sub-optimal solution that can be found quickly may be more useful. This paper proposes an anytime approach that has the ability to tradeoff deliberation time for the quality of the solution, upgrading our cooperative QoS-aware execution framework for a successful operation in highly dynamic real-time environments. The search for a feasible solution is adapted to the available time that is dynamically imposed as a result of emerging environmental conditions. Furthermore, at each iteration of the algorithms, the search of a better solution is guided by heuristic evaluation functions that optimise the rate at which the quality of the current solution improves overtime.

The conformity of the proposed algorithms for coalition formation and service proposal formulation with the desired properties of anytime algorithms was proved and the design decisions of our approach validated through extensive simulations in different highly dynamic scenarios. The achieved results demonstrate that the proposed 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. The computation overhead of the proposed anytime approach to reach its optimal solution when compared against the traditional versions of the algorithms can be considered negligible.

References

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

[2] 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.

[3] 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, Berkeley, CA, USA, 2003, pp. 21–32.

[4] S. Schmidt, T. Legler, D. Schaller, W. Lehner, Real-time scheduling for data stream management systems, in: 17th Euromicro Conference on Real-Time Systems, IEEE Computer Society, Palma de Mallorca, Spain, 2005, pp. 167–176.

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

[6] 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.

[7] 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, Washington, DC, USA, 2004, pp. 119–130.

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

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

[10] 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, Florida, USA, 2002, p. 79.

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

[12] 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, Atlanta, Georgia, USA, 2001, pp. 238–246.

[13] M. Othman, S. Hailes, Power conservation strategy for mobile comput-

ers using load sharing, SIGMOBILE Mobile Computing Communications Review 2 (1) (1998) 44–51.

[14] 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.

[15] L. Nogueira, L. M. Pinho, Dynamic qos-aware coalition formation, in: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, IEEE Computer Society, Denver, Colorado, 2005.

[16] 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, Lecture Notes in Computer Science, Springer, York, UK, 2005, pp. 25–38.

[17] 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.

[18] 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.

[19] T. Dean, M. Boddy, An analysis of time-dependent planning, in: Proceedings of the 7th National Conference on Artificial Intelligence, MIT Press, St. Paul, MN,USA, 1988, pp. 49–54.

[20] E. J. Horvitz, Reasoning under varying and uncertain resource constraints, in: Proceedings of the 7th National Conference on Artificial Intelligence, MIT Press, St. Paul, MN,USA, 1988, pp. 111–116.

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

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

[23] 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, IEEE Computer Society Press, Boston,MA,USA, 1994, pp. 90–99.

[24] 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, Orcas Island,Washington,USA, 1995, p. 12.

[25] 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.

[26] 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 49 (11) (2000) 1170–1183.

[27] 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.

[28] 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.

[29] 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, San Francisco, California, USA, 1997, p. 298.

[30] C. Lee, J. Lehoczky, D. Siewiorek, R. Rajkumar, J. Hansen, A scalable solution to the multi-resource qos problem, in: 20th IEEE Real-Time Systems Symposium, IEEE Computer Society Press, Phoenix, AZ, USA, 1999, pp. 315–326.

[31] C. Lee, J. Lehoezky, R. Rajkumar, D. Siewiorek, On quality of service optimization with discrete qos options, in: Fifth IEEE Real-Time Technology and Applications Symposium, 1999, pp. 276–286.

[32] 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, IEEE Computer Society, 2001.

[33] 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.

[34] 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.

[35] 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.

[36] 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, Cape Town, South Africa, 1999, p. 197.

[37] 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, Santa Fe, New Mexico, 2005, pp. 555–562.

[38] R. Barbosa, L. M. Pinho, Mechanisms for reflection-based monitoring of real-time systems, in: Work-In-Progress Session of the 16th Euromicro Conference on Real-Time Systems, IEEE Computer Society, Cata-

nia,Sicily,Italy, 2004.

[39] B. Landfeldt, A. Seneviratne, C. Diot, User services assistant: An end-to-end reactive qos architecture, in: Proceedings of the 6th International Workshop on Quality of Service, Napa,California, USA, 1998.

[40] F. van Harmelen, A. ten Teije, Describing problem solving methods using anytime performance profiles, in: Proceedings of the 14th European Conference on Artificial Intelligence, Berlin,Germany, 2000, pp. 181–186.

[41] 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 PARTES'04 Workshop, Pisa, Italy, 2004.

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

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

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