Optimising and Adapting the QoS of a Dynamic Set of Inter-dependent Tasks

Luís Nogueira, Luís Miguel Pinho
IPP Hurray Research Group
School of Engineering, Polytechnic Institute of Porto, Portugal
{luis,lpinho}@dei.isep.ipp.pt

Abstract

Due to the growing complexity and adaptability requirements of real-time systems, which often exhibit unrestricted Quality of Service (QoS) inter-dependencies among supported services and user-imposed quality constraints, it is increasingly difficult to optimise the level of service of a dynamic task set within an useful and bounded time. This is even more difficult when intending to benefit from the full potential of an open distributed cooperating environment, where service characteristics are not known beforehand and tasks may be inter-dependent.

This paper focuses on optimising a dynamic local set of inter-dependent tasks that can be executed at varying levels of QoS to achieve an efficient resource usage that is constantly adapted to the specific constraints of devices and users, nature of executing tasks and dynamically changing system conditions. 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 as the algorithms are given more time to run, with a minimum overhead when compared against their traditional versions.

1 Introduction

Real-time systems evolved to dynamic open environments where the characteristics of the computational load cannot always be predicted in advance and resource needs are usually data dependent and vary over time as tasks enter and leave the system [6]. Nevertheless, response to events still has to be provided within precise timing constraints in order to guarantee a desired level of performance.

Furthermore, these environments are expected to be heterogeneous, with computing devices ranging from small, resource limited mobile devices to stationary powerful servers. As such, for some of those nodes there may be a constraint on the type and size of applications they can execute with specific user's acceptable quality of service (QoS).

To address these increasingly complex demands on resources and performance, one promising solution is to support a cooperative execution of services among nodes. The possibility to partition resource

intensive services and distribute them across a set of nodes [7, 27, 13, 14] can benefit a wide range of application domains. Consider, for example, the real-time stream processing systems described in [16, 3, 25]. 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 a coordinated content analysis of the 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 [25].

In a previous paper [19] we have proposed a novel framework that facilitates the cooperation among resource constrained devices and their more powerful (or less congestioned) neighbours in order to handle stringent quality constraints imposed by users, opportunistically taking advantage of the global resources and processing power. This is achieved via the formation of a temporary group of individual nodes, which, due to its higher flexibility and agility, is capable of effectively respond to new, challenging, requirements. We call these groups *coalitions*.

We base our approach on a general form of QoS contract between users and service providers, achieved by an iterative negotiation process and dictated by each user's preferences [20]. Since different users may have different quality preferences, supporting the maximisation of each user's QoS requirements in a distributed service execution is a key issue. A coalition of nodes allocates resources to each new service, achieving the best possible instantaneous QoS and establishing an initial Service Level Agreement (SLA) that maximises the user's expressed quality preferences.

Nevertheless, short term dynamic environmental changes impose that the promised SLA can never be more than an expectation of a best-effort service quality during long term periods [2]. After admission, since the goal is to allocate resources to services in such a way that optimises the system's global utility, individual SLAs may be downgraded in a controlled fashion to a lower QoS level in order to accommodate new service requests with a higher utility. On the other hand, SLAs that were previously downgraded are restored to their original values when the needed resources become available.

A QoS manager must then react to the dynamic changes in the environment, adjusting the level of provided service and reallocating resources efficiently. Several efforts have already appeared in the context of real-time operating systems research [26, 18, 10, 23]. At the middleware level, [12] presented an optimal solution for satisfying multiple QoS dimensions in a resource-constrained environment. System resources are distributed proportionally 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 [8] and to handle non-monotonic dimensions [5]. Further improvement techniques to reduce the computation complexity of the initial proposal are described in [4].

We follow a different approach to deal with a large number of dynamic tasks, multiple resources, and real-time operation constraints in open dynamic real-time systems. We believe that frequent QoS reconfigurations may be undesirable for some users or applications. For example, in some video applications a constant frame rate may be better than a frequent variation whose average is higher than the initial contracted level of service. It is therefore desirable to propose a generic mechanism that adapts the services' execution to the dynamically changing system's conditions under the control of each user.

At the same time, 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. Therefore, it is beneficial to build systems

that can trade the quality of results against the cost of computation [29, 20]. This is particularly useful when tasks exhibit QoS dependency relations among them. Such dependency relations specify that a task offers a certain level of QoS under the condition that some specified QoS will be offered by the environment or by other tasks. In this case, the negotiation process has to ensure that a source task provides a QoS which is acceptable to all consumer tasks and lies within the QoS range supported by the source task.

This paper explores these ideas and proposes a different and novel anytime approach for the local QoS optimisation and adaptation mechanisms that react to the observed system changes. It is assumed that services share resources and their execution behaviour and input/output qualities are interdependent, i.e., a constraint on one quality or resource parameter can constrain other system's parameters. To guarantee that a valid solution is available at any time, QoS dependencies are tracked and the performed changes are propagated to all the affected attributes at each iteration of the algorithm. The proposed algorithms can be interrupted at any time and provide both a solution and a measure of its quality, which is expected to improve as the run time of the algorithms increases. A higher adaptation to changing conditions in dynamic environments is then introduced by allowing flexibility in the execution times of the algorithms. To the best of our knowledge, no other approaches exist to provide this adaptation, considering inter-dependent task sets.

2 System model

We are primarily interested in open dynamic scenarios where tasks can arrive and leave the system while others are being executed, the processing of those tasks has associated real-time execution constraints only known at admission time, and service execution can be performed by a coalition of neighbour nodes.

A real-time service $S_i = \{w_{i1}, w_{i2}, \ldots, w_{in}\}$ is a collection of one or more work units w_{ij} that can be executed at varying levels of QoS to achieve an efficient resource usage that constantly adapts to the devices' specific constraints, nature of executing tasks and dynamically changing system conditions. Each work unit $w_{ij} = \tau_{i1}, \tau_{i2}, \ldots, \tau_{in}$ is a set of one or more tasks τ_{ij} that must be executed in the same node due to local dependencies. Dependencies are modelled as a directed graph G_{ij}, where each graph node represents a task and the edges represent the data flow between the tasks. Correct decisions on service partitioning are made at run time when sufficient information about the workload and communication requirements become available [27].

Given the heterogeneity of services to be executed, users' quality preferences, underlying operating systems, networks, devices, and the dynamics of their resource usages, QoS specification becomes an important issue in the context of a distributed QoS-aware cooperative service execution framework. Nodes must either have a common understanding of how QoS should be specified, or be able to map their individual specifications into a common one. A sufficiently expressive scheme for defining the QoS dimensions subject to negotiation, their attributes and the quality constraints in terms of possible values for each attribute, as well as inter-dependency relations between some of those QoS parameters was proposed in [19] and can be expressed in several QoS description languages [9].

A key element of a particular domain's QoS information is how the quality of a task's output depends on the quality of its inputs and on the amount of resources it uses. There may exist a set $Deps_i$ of inter-dependency relations among the QoS parameters of a particular service S_i. Such QoS dependencies can be present (i) among two or more QoS attributes of a single task τ_i; (ii) among two or more tasks within

a work unit w_{ij}; or (iii) among two or more work units that may be executed in the same or in different nodes. Therefore, in addition to a service's functional behaviour and internal structure, information on how the quality of its output depends on the quality of its input must be present in the service's description at admission time.

Based on a domain's QoS characterisation, users provide a single specification of their own range of QoS preferences Q_i for a complete service S_i, ranging from a desired QoS level $L_{desired}$ and the maximum tolerable service degradation, specified by a minimum acceptable QoS level $L_{minimum}$, without having to understand the individual work units that make up the service. The relative decreasing order of importance imposed in dimensions, attributes and values expresses his preferences in a qualitative way (elements identified by lower indexes are more important than elements identified by higher indexes), eliminating the need to specify an utility value for every quality choice.

For example, suppose that in a remote surveillance system video is more important to a particular user than audio. Assuming that for that user a grey scale, low frame rate is fine for video, his request could be as follows (the following list of possible QoS dimensions and attributes is not intended to be exhaustive):

```
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 this qualitative ordering, 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 in the audio quality dimension. For each of these attributes, a preference order for the QoS values is as well expressed. Note that it is possible to define multiple intervals in decreasing preference order for the user's acceptable values of a particular attribute. The evaluation of the user's acceptability of each service proposal can deal with this type of definition [19, 20].

Given the spectrum of the user's acceptable QoS levels, nodes respond to the cooperation request and formulate the best instantaneous service level agreement (SLA) they can offer for each work unit $w_{ij} \in S_i$. At each node, the local QoS optimisation recomputes the set of QoS levels for the new set of tasks, as it tries to find a feasible set of service configurations that maximises users' satisfaction with the provided service as well as minimises the impact on the current QoS of the previously accepted tasks. At each iteration, the search of a better solution is guided by a heuristic evaluation function that optimises the rate at which the quality of the current solution improves overtime. The time to find a feasible service solution is dynamically imposed as a result of emerging environmental conditions [21].

Currently provided SLAs can be downgraded to accommodate new services with a higher utility or restored to their original values when the needed resources become available. To maximise system's stability, it is ensured that during a specific time interval Δ_t the promised QoS level will not be changed by the node's local QoS optimisation and adaptation mechanisms. Forecasted stability periods are periodically updated in response to fluctuations in the tasks' traffic flow, relating observations of past and present system's conditions.

With several independently developed applications with different timing requirements coexisting in the same node, it is important to guarantee a predictable performance under specified load and failure

conditions, and ensure a 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, providing service guarantees to critical applications [21, 22].

3 Formulating an initial SLA

QoS-aware applications are usually structured to provide different quality levels, which have associated estimations of the needed resources. From this perspective, service negotiation can be based on a iteratively optimisation of each node's local QoS level, maximising the provided QoS for the new service and minimising the quality degradation of previously accepted tasks.

Each new service S_i to be executed has associated a set of user-imposed QoS preferences Q_i, expressed in decreasing preference order. Each $Q^i_{kj} = \{Q^i_{kj}[0], \ldots, Q^i_{kj}[n]\}$ is a finite set of n quality choices for the j^{th} attribute of the k^{th} QoS dimension associated with service S_i.

Since the tasks in $w_{ij} \in S_i$ can exhibit unrestricted QoS inter-dependencies among them or with currently executing tasks, the negotiation process must ensure that a source task provides a QoS which is acceptable to all consumer tasks and lies within the QoS range supported by the source task. Based on the new service's data flow graph G_i and on its set of inter-dependency relations $Deps_i$, Algorithm 1 tracks QoS dependencies and propagates the performed changes in one attribute to all local affected attributes at each iteration. If, by following the chain of dependencies, the algorithm finds a task that is already in its list of resolved dependencies, a deadlock is detected and the service proposal formulation is aborted.

In order to be useful in practice, an anytime approach must try to quickly find a sufficiently good initial proposal and gradually improve it if time permits, conducting the search for a better feasible solution in a way that maximises the expected improvement in the solution's quality [29]. Algorithm 1 starts by keeping the QoS levels of previously guaranteed tasks and by selecting the lowest requested QoS level for the new tasks in w_{ij} that complies with any eventual QoS dependency with currently executing tasks. Note that this is the service configuration with the highest probability of being feasible without degrading the current level of service of previously accepted tasks.

The algorithm iteratively works on the problem of finding a feasible set of service configurations with a higher utility and produces results that improve in quality over time. At each iteration, the search of a better feasible solution is guided by the maximisation of the users' expected satisfaction with the provided service. When w_{ij} can be accommodated without degrading the QoS of previously accepted tasks, the configuration that maximises the reward's increase of w_{ij} is incrementally selected (Step 1). On the other hand, when QoS degradation is needed to accommodate w_{ij}, the algorithm incrementally selects the configuration that minimises the reward's decrease for all local tasks (Step 2).

Rewards obtained by each selected SLA are determined by considering the proximity of a service proposal with respect to the weighted user's QoS preferences expressed in decreasing order (Equation 1). The *penalty* parameter can be fine tuned by the system's administrator and its value should increase with the distance to the user's preferred value for a particular quality attribute. The expressed relative order of preference determines the weight β of each dimension.

$$reward(S_i) = 1 - \sum_{j=0}^{\forall Q_{jk} < Q_{best_j}} \beta_j * penalty_j \quad (1)$$

Algorithm 1 Service proposal formulation

Let τ^p be the set of previously accepted tasks
Let τ^e be the set of tasks whose stability period Δ_t has expired
Let $\tau^* = \tau^p \cup w_{ij}$ be the new set of tasks

Step 1: Improve the QoS level of each task $\tau_a \in w_{ij}$
Select $Q_{kj}[n]$, the lowest requested level of service for all k QoS dimensions, considering the dependencies with the previously accepted tasks τ^p, for all newly arrived tasks τ_a in w_{ij}
Keep the current QoS level of previously accepted tasks τ^p
while the new set of local tasks τ^* *is* feasible **do**
  for each task $\tau_a \in w_{ij}$ **do**
    for each attribute without dependencies with τ^p receiving service at $Q_{kj}[m] > Q_{kj}[0]$ **do**
      Upgrade attribute to the next possible value $m - 1$
      Follow attribute's dependencies in w_{ij} and change values accordingly
      Determine the utility increase of this upgrade
    end for
  end for
  Find task τ_{max} whose reward's increase is maximum and perform upgrade
end while

Step 2: Find the local minimal service degradation in τ^* **to accommodate each** $\tau_a \in w_{ij}$
while the new set of local tasks τ^* *is not* feasible **do**
  for each task $\tau_i \in \tau^e \cup w_{ij}$ receiving service at $Q_{kj}[m] > Q_{kj}[n]$ **do**
    for all QoS attributes **do**
      Degrade attribute j to the previous possible value $m + 1$
      Follow dependencies of attribute j in all local tasks τ^* and change values accordingly
      Determine the utility decrease of this downgrade
    end for
  end for
  Find task τ_{min} whose reward's decrease is minimum and perform downgrade
end while
return new local QoS optimisation

By combining the rewards of all QoS configurations, a measure of a node's global reward can be obtained (Equation 2).

$$R = \frac{\sum_{i=1}^{n} reward(S_i)}{n} \tag{2}$$

Note that unless all services are executed at their highest requested QoS level, there is a difference between the current node's global reward $R_{current}$ and the maximum theoretical global reward R_{max}. This difference can be caused by either resource limitations, which is unavoidable, or poor load balancing. The later can be improved by using global rewards in the nodes' selection for the new cooperative

coalition [19]. Selecting the node with a higher global reward for service proposals with a similar QoS level, not only maximises a particular user's satisfaction with the provided service, but also maximises the global system's utility, since a higher local reward clearly indicates that the node's previous set of tasks had to suffer less QoS degradation in order to accommodate the new tasks.

Equally important for the usefulness of an anytime approach is the fact that Algorithm 1 always improves or maintains the current solution's quality 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 service configuration for the new task set. However, each intermediate configuration, even if not feasible, is used to calculate the next solution, minimising the search effort.

Instead of a binary notion of the solution's correctness, the algorithm returns a proposal and a measure of its quality. Equation 3 considers the reward achieved by the new arriving work unit $r_{w_{ij}}$, the impact on the provided QoS of previous existing tasks r_{τ^p} and the value of the previous generated feasible configuration Q'_{conf}. Initially, Q'_{conf} is set to zero and its value is only updated if the achieved solution is feasible.

$$Q_{conf} = \left(r_{w_{ij}} * \frac{\sum_{i=0}^{n} r_{\tau^p}}{n} \right)^{(1-Q'_{conf})} \tag{3}$$

The algorithm can be interrupted at any time as a consequence of the dynamic nature of the environment [21, 22], or finishes when it finds a feasible set of QoS configurations whose quality cannot be further improved, or when it finds that even if all the tasks would be served at the lowest admissible QoS level it is not possible to accommodate the new requesting tasks in w_{ij}. In this later case, the service request is rejected and the previously accepted tasks continue to be served at their current QoS levels.

The group of nodes that proposes service closer to the user's quality preferences for each of the work units $w_{ij} \in S_i$ will be selected to form the new temporary coalition for a cooperative execution of S_i [20].

4 Supporting QoS adaptation and stability

Making an effective use of the system's resources in a dynamic system is a complex and difficult task [8]. Any service provider's resource allocation policy is subject to environmental uncertainties, and for that reason, the promised SLA can never be more than an expectation of a quality level during longer term periods [2].

Since the goal is to maximise the node's global reward achieved by the selected SLAs, it makes sense to upgrade the currently provided SLAs when the needed resources to perform such upgrade become available. However, an undesirable high reconfiguration rate may be achieved by reconfiguring the offered SLAs on every service departure in highly dynamic systems. As such, the QoS reconfiguration should be based on a specific threshold of desired system utilisation. Also, since different users or applications may have different service requirements, services' QoS reconfiguration should be done under the control of the user.

We propose that each user should be able to add a *QoS stability* personal choice to his service request, extending his influence to the QoS adaptation behaviour during the service's execution. Possible

attributes for this stability dimension can be a minimum granted stability period Δ_{min} and a minimum increment in the service's reward U_{min} to perform the upgrade. These measures can be interpreted as "do not change to a better quality state unless this gives me at least a reward's increment of U_{min} over a Δ_{min} period".

From the service provider's side, each proposed SLA includes a stability period Δ_t, indicating that during a specific time interval the promised level of service will be assured, by both the local QoS optimisation policy, on the arrival of a new service request, and the local QoS reconfiguration policy, executed when an underutilisation of the service provider is detected. To adapt the system's behaviour to the observed environmental changes, proposed stability periods are periodically updated in response to variations in the tasks' traffic flow and correspondent resource usage.

The degree of the user's acceptability of the proposed level of service and stability period is determined by measuring the distance between the user's preferred values and the values proposed by the service provider [20].

4.1 Promised stability periods

Service stability could be achieved by using a fixed, large enough value for Δ_t, but this would then result in lack of responsiveness in adaptability to environmental changes. Furthermore, fixed values only make sense when there is some knowledge about the tasks' traffic model, which is not the case in open real-time systems.

Time series analysis comprises methods that attempt to understand a sequence of data points, typically measured at successive times, spaced at (often uniform) time intervals to make forecasts. The simple exponential smoothing (SES) model [1] has become very popular as a forecasting method for a wide variety of time series data as it is both robust and easy to apply. In fact, empirical research by Makridakis et al. [15] has shown SES to be the best choice for one-period-ahead forecasting, from among 24 other time series methods and using a variety of accuracy measures. Thus, regardless of the theoretical model for the process underlying the observed time series, simple exponential smoothing will often produce quite accurate forecasts.

Intuitively, past data should be discounted in a more gradual fashion, putting relatively more weight on the most recent observations. SES accomplishes exactly such weighting, whit exponentially smaller weights being assigned to older observations. We use the SES model to forecast the length of the next period without QoS degradation for each of the system's resources r_i. Equation 4 is used recursively to update (forecast) the smoothed series as new observations are recorded. The observed minimum stability period for resource r_i during the period of observation t is denoted by x_t and $\Delta_t^{r_i}$ may be regarded as the best estimate of what the next value of x will be.

$$\Delta_t^{r_i} = \alpha x_t + (1 - \alpha)\Delta_{t-1}^{r_i} \tag{4}$$

Each new forecast is then based on the previous forecast plus a percentage of the difference between that forecast and the actual value of x_t at that point. The percentage $0 \leq \alpha \leq 1$ is known as the smoothing factor. Values of α close to 1 have less of a smoothing effect and give a greater weight to recent changes in the data, while values of α closer to 0 have a greater smoothing effect and are less responsive to recent changes. α can then be adjusted by the system's designer to create a more reactive or conservative response to recent changes in the tasks' traffic flow. Alternatively, a statistical technique may be used to optimise the value of α, minimising the difference between the predicted and observed

values. For example, the method of least squares may be used to determine α's value for which the sum of the quantities $(\Delta_{t-1}^{r_i} - x_t)^2$ is minimised [1].

Having the stability forecasts for each of the system's resources, the promised stability period for service S_i must be based on a coherent summary of the forecasts for the resources it uses. Combination of several dynamical variables using AND (multiplication) and OR (addition) have already been discussed to provide more expressive policies for SLAs [24]. Equation 5 determines promised stability periods by aggregating the forecasted values for each resource r_i used by S_i.

$$\Delta_t = \Delta_{r_1}\ AND\ \Delta_{r_2}\ AND\ \ldots\ AND\ \Delta_{r_n} \tag{5}$$

Fuzzy logical reasoning provides a suitable interpretation of the stability periods of each resource r_i in order to determine the promised stability period for a particular service S_i. The use of the fuzzy AND operator (the min function) leads to a correct system behaviour and it is also computationally simple to evaluate, allowing a quick evaluation of stability periods for each service, using Equation 6.

$$\Delta_t = min(\Delta_t^{r_i}) \tag{6}$$

The system ensures that during the forecasted Δ_t no change in the promised QoS for service S_i will be made. However, services whose stability period has already expired can be downgraded to a lower quality level to accommodate new requesting tasks.

5 Dynamic QoS reconfiguration

The goal of a dynamic QoS reconfiguration is to reallocate the system's resources to optimise the total utility achieved by all the services being locally executed, upgrading the current SLAs when the needed resources become available. However, rather than trying to upgrade on every service departure, QoS reconfiguration should be based on a specific threshold of desired system utilisation, avoiding high reconfiguration rates in dynamic systems.

Let L_t be the desired threshold to activate the dynamic QoS reconfiguration of previously degraded tasks whose stability period Δ_t has already expired. Let L be the current level of the system's load demanded by the n offered SLAs. Intuitively, $L < L_t$ indicates an underutilisation and the dynamic QoS reconfiguration will take place.

Our basic viewpoint is that service stability can be more important for some users than some momentary maximum quality. Although the framework ensures a fixed quality level during a dynamically determined period of time, an upgrade of the current quality level when the needed resources become available should be done under the control of each user.

Possible upgrades for the previously downgraded SLAs are determined by Algorithm 2. The anytime algorithm iteratively maximises the users' expected satisfaction with the new SLA by selecting, at each iteration, the new configuration that achieves the greatest reward's increase until the initially granted SLA is reached. The goal is to provide the maximum desirable stability, by always trying to restore previously degraded services to their original SLA.

The achieved solution's quality at each iteration is measured by Equation 7, considering the reward achieved by the new SLAs of previously degraded tasks r_{τ^d} and the value of the previous generated feasible configuration Q'_{conf}. Initially, Q'_{conf} is set to zero and its value is only updated if the achieved solution is feasible.

Algorithm 2 Dynamic QoS reconfiguration

Let τ^d be the set of previously downgraded tasks whose Δ_t has expired
Let τ^o be the set of all other tasks
Let $Q_{kj}[init]$ be the initial quality level for attribute j of the k_{th} QoS dimension for task $\tau_i \in \tau^d$
Keep the current QoS level for all tasks in τ^o
while the new set of configurations *is* feasible **do**
  for each task $\tau_i \in \tau^d$ **do**
    for each attribute without dependencies with τ^o receiving service at $Q_{kj}[m] > Q_{kj}[init]$ **do**
      Upgrade attribute to the next possible value $m - 1$
      Follow attribute's dependencies and change values accordingly
      Determine the utility increase of this upgrade
    end for
  end for
  Find task τ_{max} whose reward's increase is maximum and perform upgrade
end while

$$Q_{conf} = \left(\frac{\sum_{i=0}^{n} r_{\tau^d}}{n} \right)^{(1-Q'_{conf})} \tag{7}$$

The algorithm can be interrupted at any time or finishes when it finds a feasible set of QoS configurations whose quality cannot be further improved due to resource limitations or all the initially granted SLAs are reached. If a new feasible set of configurations is found, changes to a "better quality" are controlled by each user's stability requirements, namely a minimum increment in the SLA's utility and a minimum promised stability period. As these constraints are stringent, it is harder to move to better quality states. Users' influence is thus extended to the services' adaptation behaviour during execution. Simulation results reported in Section 6 show that such influence can be achieved.

6 Behaviour and Evaluation

In order to evaluate the behaviour of the proposed anytime algorithms in dynamic open real-time scenarios we have conducted extensive simulations. A special attention was devoted to introduce a high variability in the characteristics of the conducted simulations.

The number of simultaneous nodes in the network varied from 10 to 50, with resources' capacities being randomly partitioned among all 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. At randomly generated times, one or more users generated new service requests at randomly selected nodes, expressing the spectrum of acceptable QoS levels in a qualitative way, ranging from a randomly generated desired QoS level to the randomly generated maximum tolerable service degradation. The relative decreasing order of importance imposed in dimensions, attributes and values was also randomly generated. The QoS domain used to generate the requests was composed

by 4 quality dimensions, each with 5 attributes and 10 possible values for each attribute.

An application that captures, compresses and transmits frames of real-time data to end users using a diversity of users' QoS preferences and inter-dependency relations among tasks was used as a scenario. The application was composed by a source unit to collect the data, a compression unit to gather and compress the data that may come from multiple sources, a transmission unit to transmit the data over the network, a decompression unit to convert the data into the user's specified format, and an user unit to display the data in the user's end device.

Promised stability periods were determined by taking into consideration the observed variations in the tasks' traffic flow and correspondent resource usage, adapting the system to the observed environmental changes. The value of the smoothing factor α was optimised using the method of least squares.

The reported results were observed from multiple and independent simulation runs, with initial conditions and parameters, but different seeds for the random values used to drive the simulations, obtaining independent and identically distributed variables, with a reasonably good statistical performance [11]. The random values were generated by the Mersenne Twister algorithm [17].

Anytime algorithms correlate the output's quality with time in a performance profile [29], a function that maps the time given to an anytime algorithm (and in some cases also input quality) to the quality of the algorithm's produced solution. The performance profile was estimated by normalising the results of the conducted simulations with respect to the algorithm's completion time [28], which is the minimal time when the expected quality is maximal, rather than measuring the algorithm's absolute execution time on every run of the simulation.

In the first study, the evaluation of Algorithm 1's behaviour was based on the two possible scenarios it considers, according to resource availability. In the first one, the average amount of resources per node was greater than the average amount of resources necessary to execute a new service, while in the second one, average service requirements were greater than the average amount of available resources per node, demanding QoS degradation of previously accepted services.

When there are enough resources to improve the initial feasible solution without degrading the current QoS of the previously accepted tasks (Figure 1), the solution's quality Q_{conf} is incrementally improved by increasing the new service's reward. Consequently, the node's local reward that is affected by the initial solution of serving the new service with the minimum requested QoS level, also increases as the algorithm approaches its final solution. With limited resources (Figure 2), an upgrade of the new service's reward may result in an unfeasible set of tasks, which demands QoS degradation of some tasks. When downgrading, Algorithm 1 selects, at each iteration, the configuration that minimises the reward's decrease for all local services.

From Figures 1 and 2, two important conclusions can be taken considering the desirable properties of an anytime algorithm [29]. First, the solution's quality measure is a non-decreasing function of time, since the current feasible configuration is only updated if, and only if, another feasible solution with a higher quality for the user's request under negotiation is found. Second, at an early stage of the computation the quality of the proposed solution is expected to be sufficiently close to its final value at completion time. With spare resources (Figure 1), at only 20% of the computation time, the solution's quality for the new service is near 74% of the achieved quality at completion time. When QoS degradation is needed (Figure 2), the configuration for the new service achieves near 85% of its final quality at 20% of the needed computation time.

During the conducted simulations the behaviour of the anytime QoS reconfiguration algorithm was also evaluated. The average quality increase of the previously downgraded SLAs whose stability period

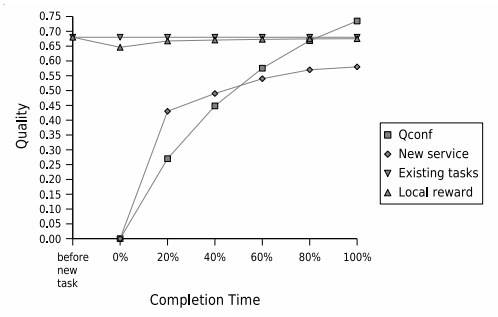

Figure 1. Initial SLA with spare resources

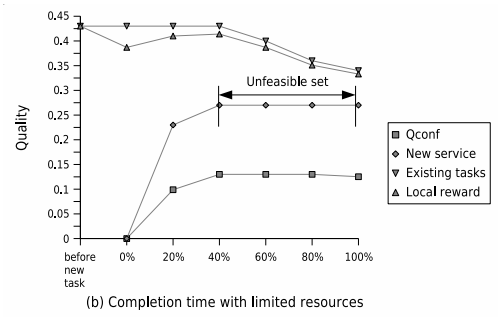

Figure 2. Initial SLA with limited resources

had already expired was measured on every dynamic reconfiguration that took place when a system's utilisation below 60% was detected. The results are plotted in Figure 3 and similar conclusions can be taken for both algorithms, regarding the desirable properties of anytime algorithms.

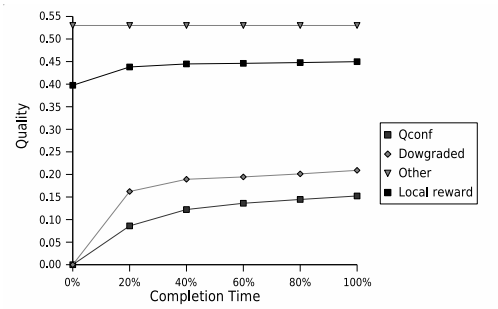

Figure 3. Dynamic QoS reconfiguration

A third study evaluated the users' influence on the services' reconfiguration behaviour. Three permanent service requests were added to the dynamic traffic that was randomly generated at each simulation run. The same randomly generated spectrum of acceptable QoS values in the same decreasing preference order was used in the three requests. They only differed on the users' QoS stability con-

straints for the minimum utility increase and stability period, $User_1 = \{0, 0s\}$, $User_2 = \{0.2, 10s\}$, $User_3 = \{0.3, 30s\}$, respectively.

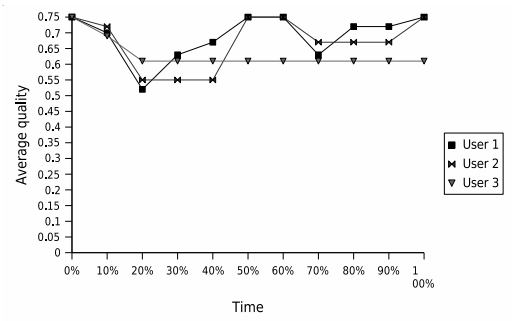

Figure 4. QoS reconfiguration rate

The influence of personal constraints on the system's adaptation behaviour is clearly observable in Figure 4 when comparing the three requests against each other. As the user's constraints for a service upgrade are harder to achieve there is less probability to change and stay in a better quality level.

A fourth study compared the computational cost of the proposed anytime approach when compared against the traditional version of the algorithms to reach their optimal solutions. The results obtained by the local QoS optimisation are reported in the next paragraphs. Similar results were obtained for the proposed QoS adaptation mechanism.

The traditional local QoS optimisation proposed in [19] was extended to resolve any QoS dependencies present in its optimal solution and used in this comparison. It starts by selecting the user's preferred QoS level for the new service and stops when it finds a feasible solution that minimises the impact on the provided global level of service caused by the new service's arrival. The comparison's results were normalised with respect to the completion time of the longest solution and plotted in Figures 5 and 6.

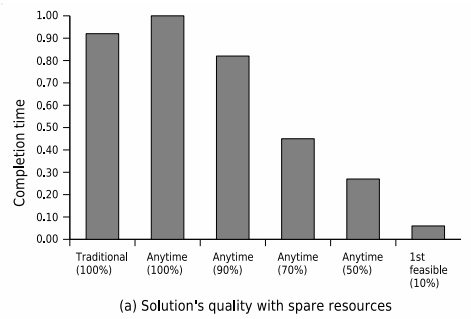

Figure 5. Needed time with spare resources

Both figures show that the anytime version can take more time to achieve the same optimal solution in both scenarios. Two main reasons explain this difference. First, the anytime version resolves QoS inter-dependencies at each iteration. Recall that the goal is to be able to interrupt the algorithm at any time and still be able to return a valid solution. Without any restriction on the needed time to compute its

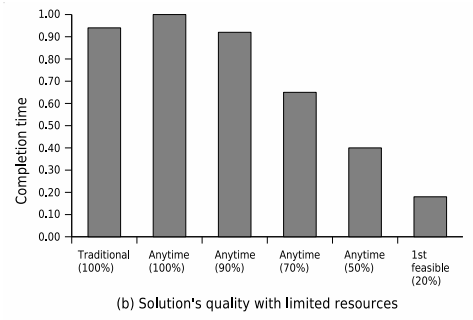

Figure 6. Needed time with limited resources

optimal solution, the traditional version only has to resolve any eventual dependencies after finding the best configuration for the individual tasks. Dependencies were resolved by relaxing the optimal values of some of the individual tasks to the maximum value constrained by the existing inter-dependency relations.

Second, the different approaches to achieve an optimal solution can have an impact on the number of needed iterations, particularly with spare resources. Since the anytime version tries to quickly find a feasible solution, it starts by considering the worst requested QoS values for the new service and iteratively improves that solution until the optimal one is found. On the other hand, the traditional version starts by trying to provide the best requested level of service for the new tasks and iteratively degrades all tasks, stopping when it finds a feasible, optimal solution.

Nevertheless, in both scenarios the anytime version is by far quicker to find a feasible solution. With spare resources, the first feasible solution with a quality near 10% of its optimal value is almost immediately found, and at near 20% of the running session the solution's quality is already around 50% of its optimal value. With limited resources, the anytime version takes about 20% of its computation time to reach a feasible solution with 20% of its optimal solution's quality, and at near 40% of the running session it achieves 50% of its optimal value. These results are in consonance with the performance profiles plotted in Figures 1 and 1, which further validate the ability of the proposed algorithm to quickly find a feasible solution and maximise the improvement in the expected solution's quality at each iteration.

7 Conclusions

A cooperative execution of resource intensive services among nodes of a heterogeneous open real-time system is a promising solution to address the increasingly demanding requirements on resources and performance. Furthermore, since different users may have different quality and stability preferences for their services, supporting the maximisation of each user's requirements is a key issue when allocating resources.

However, finding an optimal resource allocation that deals with both users' and nodes' constraints and constantly adapts to the observed changes in the environment can be quite complex and very difficult to achieve within an useful and bounded time. The problem is even harder to solve when tasks exhibit unrestricted inter-dependencies among them. The proposed anytime local QoS optimisation and adaptation algorithms are able to quickly find a sub-optimal solution at an early stage of the computation. These initial solutions are then iteratively refined as the algorithm are given more time to run. Such flexibility

allows a higher adaptation to the dynamically changing conditions of open real-time systems and, as the achieved results demonstrate, can be obtained with an overhead that can be considered negligible when compared against the introduced benefits.

References

[1] R. G. Brown. *Smoothing, forecasting and prediction of discrete time series*. Prentice-Hall, Englewood Cliffs, NJ, 1963.

[2] M. Burgess. On the theory of system administration. *Science of Computer Programming*, 49:1, 2003.

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

[4] S. Ghosh, J. Hansen, R. R. Rajkumar, and 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, August 2004.

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

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

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

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

[9] J. Jin and K. Nahrstedt. Qos specification languages for distributed multimedia applications: A survey and taxonomy. *IEEE MultiMedia*, 11(3):74–87, 2004.

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

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

[12] C. Lee, J. Lehoczky, D. Siewiorek, R. Rajkumar, and J. Hansen. A scalable solution to the multi-resource qos problem. In *20th IEEE Real-Time Systems Symposium*, pages 315–326, 1999.

[13] Z. Li, C. Wang, and 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*, pages 238–246. ACM Press, 2001.

[14] Z. Li, C. Wang, and 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*, page 79. IEE Computer Society, 2002.

[15] S. Makridakis, A. Andersen, R. Carbone, R. Fildes, M. Hibon, R. Lewandowski, J. Newton, E. Parzen, and R. Winkler. The accuracy of extrapolation (time series) methods: Results of a forecasting competition. *Journal of Forecasting*, 1:111–153, 1982.

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

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

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

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

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

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

[22] L. Nogueira and L. M. Pinho. Shared resources and precedence constraints with capacity sharing and stealing. In *Proceedings of the 22th IEEE International Parallel and Distributed Processing Symposium (to appear)*, Miami,Florida,USA, April 2008.

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

[24] G. Rodosek. Quality aspects in it service management. In *Proceedings of the 13th IFIP/IEEE Internation Workshop on Distributed Systems: Operations and Management*, pages 82–93, Montereal, Canada, October 2002.

[25] S. Schmidt, T. Legler, D. Schaller, and W. Lehner. Real-time scheduling for data stream management systems. In *Proceedings of the 17th Euromicro Conference on Real-Time Systems*, pages 167–176, 2005.

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

[27] C. Wang and Z. Li. Parametric analysis for adaptive computation offloading. In *Proceedings of the ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation*, pages 119–130. ACM Press, 2004.

[28] S. Zilberstein. *Operational Rationality Through Compilation of Anytime Algorithms*. PhD thesis, Department of Computer Science, University of California at Berkeley, 1993.

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