Handling QoS Dependencies in Distributed Cooperative Real-Time Systems

Luís Nogueira and Luís Miguel Pinho

Abstract Due to the growing complexity and adaptability requirements of real-time embedded systems, which often exhibit unrestricted 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. This paper proposes an iterative refinement approach for a service's QoS configuration taking into account services' inter-dependencies and quality constraints, and trading off the achieved solution's quality for the cost of computation. Extensive simulations demonstrate that the proposed anytime algorithm is able to quickly find a good initial solution and effectively optimises the rate at which the quality of the current solution improves as the algorithm is given more time to run. The added benefits of the proposed approach clearly surpass its reduced overhead.

1 Introduction

Most of today's embedded systems are required to work in highly dynamic 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 [1]. Nevertheless, response to events still have to be provided within precise timing constraints in order to guarantee a desired level of performance.

One promising solution is to support cooperation among nodes of a distributed system. A careful partitioning of the workload between a device and their remote

Luís Nogueira · Luís Miguel Pinho
IPP-Hurray Research Group, Polytechnic Institute of Porto
e-mail: luis, lpinho@dei.isep.ipp.pt

neighbours has been proved to achieve power and performance gains [2, 4]. Nevertheless, supporting the maximisation of each user's quality of service (QoS) requirements in such a distributed service execution is a key issue [7]. This need imposes a complexity that may prevent the possibility of computing an optimal QoS configuration within an useful and bounded time. It is therefore beneficial to build systems that can trade off the needed computation time for the quality of the achieved solution. In [9], an iterative refinement QoS optimisation that maximises the provided QoS of a set of independent tasks was proposed. The configuration process can be interrupted at any time and still provide a solution and a measure of its quality.

However, the problem is even more complex 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 proposes an anytime local QoS optimisation, assuming 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. To the best of our knowledge no other works propose an anytime approach for a distributed QoS configuration of resource intensive services in open real-time embedded services with the ability to handle tasks' inter-dependencies and maximise the satisfaction of each user's quality preferences.

2 System model

With tasks joining and leaving the system at any time both resource demands and availability can fluctuate rapidly and unpredictably. This may affect the ability to individually execute services with specific user-imposed QoS constraints and drive devices to group themselves in a coalition for a cooperative service execution [7].

A real-time service $S_i = \{w_{i1}, w_{i2}, \dots, w_{in}\}$ 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 embedded devices' specific constraints, nature of executing tasks and dynamically changing system conditions. Each work unit $w_{ij} = \tau_{i1}, \tau_{i2}, \dots, \tau_{in}$ 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}, with each graph node representing a task and the edges representing 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 [12].

Given the heterogeneity of services to be executed, users' 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 [7] and can be expressed in several QoS description languages [3]. This scheme is used to specify in the service's description, known at admission time, how a task's output quality depends on the quality of its inputs and on the amount of resources it uses to produce the output. These inter-dependency relations among the QoS parameters of a particular service S_i 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.

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, without having to understand the individual work units that make up the service. Preferences are defined in a qualitative way, imposing a relative decreasing order of importance on QoS dimensions, their attributes, and acceptable values. Given the spectrum of the user's acceptable QoS levels, each node formulates the best instantaneous service level agreement (SLA) it can offer. 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 [10]. A SLA also includes a stability period Δ_t, guaranteeing that during a specific time interval the promised QoS will be assured by the node's local QoS optimisation. Δ_t is dynamically determined in response to fluctuations in the tasks' traffic flow, relating observations of past and present system's conditions and extending users' influence also to the services' adaptation during execution [8].

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 [10, 11].

3 Optimising the QoS of a inter-dependent task set

The formation of a cooperative coalition should enable the selection of individual nodes that, based on their own resources and availability, will constitute the group that maximises the user's QoS requirements Q_i associated with service S_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 a work unit w_{ij} of the new service S_i.

This paper extends the anytime local[1] QoS optimisation introduced in [9] by allowing tasks to exhibit unrestricted inter-dependencies among them, only know at admission time. Service negotiation is based on a iteratively refinement of each node's local QoS level, maximising the provided QoS for the new service and minimising the quality degradation of previously accepted tasks. The proposed approach ensures 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. 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 work on the problem of finding a feasible set of service configurations 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 previously accepted tasks' QoS, the configuration that maximises the increase in the obtained reward of the new service is incrementally selected. On the other hand, when QoS degradation is needed to accommodate w_{ij}, the algorithm incrementally selects the configuration that minimises the decrease in the obtained reward for all services.

Rewards 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 and its value should increase with the distance to the user's preferred value for a particular quality attribute.

[1] Dependencies among tasks running on different nodes will be handled in future work

Algorithm 1 Service proposal formulation

```
Let $\tau^p$ be the set of previously accepted tasks
Let $\tau^e$ be the set of tasks whose stability period $\Delta_t$ 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 $\tau^p$, for all newly arrived tasks $\tau_a$ in $w_{ij}$
Keep the current QoS level of previously accepted tasks $\tau^p$
while the new set of local tasks $\tau^*$ is feasible do
  for each task $\tau_a \in w_{ij}$ do
    for each attribute without dependencies with $\tau^p$ receiving service at $Q_{kj}[m] > Q_{kj}[0]$ do
      Upgrade attribute $j$ to the next possible value $m-1$
      Follow dependencies of attribute $j$ in $w_{ij}$ and change values accordingly
      Determine the utility increase of this upgrade
    end for
  end for
  Find task $\tau_{max}$ whose reward increase is maximum and perform upgrade
end while

Step 2: Find the local minimal service degradation in $\tau^*$ to accommodate each $\tau_a \in w_{ij}$
while the new set of local tasks $\tau^*$ 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 $\tau^*$ and change values accordingly
      Determine the utility decrease of this downgrade
    end for
  end for
  Find task $\tau_{min}$ whose reward decrease is minimum and perform downgrade
end while
return new local QoS optimisation
```

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

By combining the rewards of all services' configurations, a measure of a node's global satisfaction with the proposed QoS for the new task set 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 local reward $R_{current}$ and the maximum theoretical local 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 the nodes' local rewards to select the nodes that are going to constitute the new cooperative coalition [7]. Selecting the node with a higher local reward for similar service proposals, 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.

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 service r_{S_i}, the impact on the provided QoS of previous existing tasks r_{S_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.

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

The algorithm can be interrupted at any time as a consequence of the dynamic nature of the environment [10], or finishes when it finds a feasible configuration whose quality cannot be further improved, or when it finds that even if all the node's tasks would be served at the lowest requested 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.

4 Behaviour and Evaluation

Since we are primarily interested in dynamic open real-time scenarios 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.

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.

At randomly generated times, one or more users generated new service requests at randomly selected nodes expressed 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. 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 [8].

Since the proposed algorithm clear splits the formulation of a new service proposal in two different scenarios according to resource availability, the evaluation of its behaviour was based on those two scenarios. 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. 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 [5]. The random values were generated by the Mersenne Twister algorithm [6].

The first study evaluated the algorithm's behaviour when approaching its optimal solution. Anytime algorithms correlate the output's quality with time in a performance profile [14], 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 of the proposed anytime algorithm was estimated by normalising the results of the conducted simulations with respect to the algorithm's completion time [13], 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.

When there are enough resources to improve the initial feasible solution without degrading the current QoS of the previous tasks (Figure 1 (a)), 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 proposed solution of serving the new arrived service with the minimal requested QoS level, also increases as the algorithm approaches its final solution. With limited resources (Figure 1 (b)), an upgrade of the new service's reward may result in an unfeasible set of tasks, which demands service degradation of some tasks. At each iteration, the configuration that minimises the decrease in the obtained reward for all services is selected.

From Figure 1, two important conclusions can be taken considering the desirable properties of an anytime algorithm [14]. 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

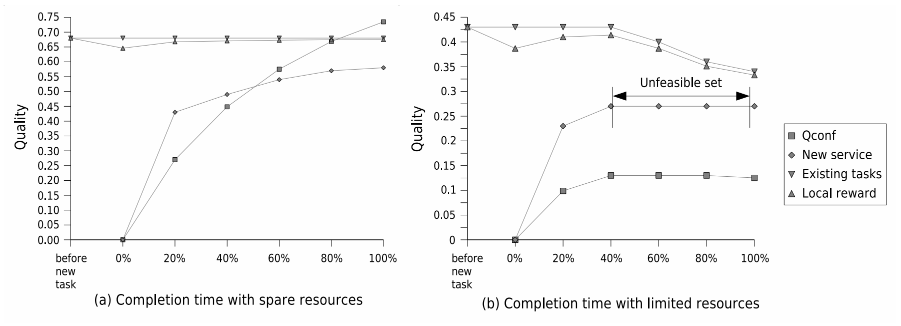

Fig. 1 Expected solution's quality in different scenarios

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 (a)), 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 1 (b)), the configuration for the new service achieves near 85% of its final quality at 20% of the needed computation time.

A second study compared the computational cost required by the anytime approach to reach its optimal solution against the traditional version of the algorithm. The traditional local QoS optimisation proposed in [7] 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.

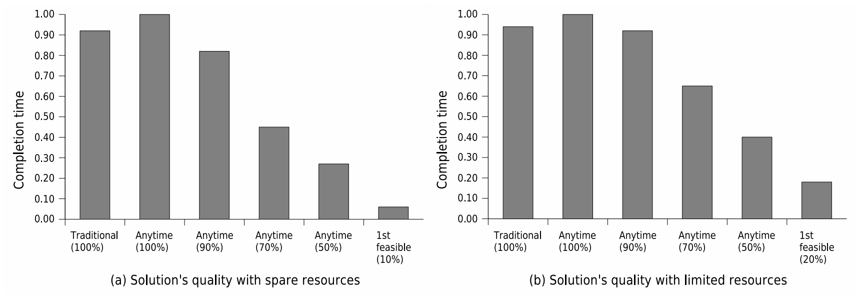

Fig. 2 Computational cost of both approaches

Figure 2 shows 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 optimal solution, the traditional version only has to resolve dependencies after finding the best configuration for the individual tasks in a second phase. Dependencies are 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 Figure 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.

5 Conclusions

It is not possible to predict in advance the characteristics of a dynamic open real-time system's computational load. Resource needs are usually data dependent and vary over time as tasks dynamically enter and leave the system. As such, nodes may need to cooperate with their neighbours in order to fulfil complex service requirements imposed by users. However, finding an optimal resource allocation that deals with both users' and nodes' constraints can be quite complex and may take a long time. The problem is even harder to solve within a useful time when tasks exhibit unrestricted inter-dependencies among them only known at admission time.

The proposed anytime approach is able to quickly find a sub-optimal solution at an early stage of the computation time. This service solution is then iteratively refined as the algorithm has more time to run. Such flexibility in the needed time to find a feasible service proposal allows a higher adaptation to the dynamically changing conditions of open real-time systems and, as the achieved results demonstrate, can be achieved with an overhead that can be considered negligible when compared against the introduced benefits.

Acknowledgements

This work was supported by FCT (CISTER Research Unit - FCT UI 608 and the CooperatES project - PTDC/EIA/71624/2006), and by the European Commission through the ARTIST2 NoE (IST-2001-34820).

References

1. Sourav Ghosh, Ragunathan (Raj) Rajkumar, Jeffery Hansen, and John 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.
2. Xiaohui Gu, Alan Messer, Ira Greenberg, Dejan Milojicic, and Klara Nahrstedt. Adaptive offloading for pervasive computing. *IEEE Pervasive Computing Magazine*, 3(3):66–73, 2004.
3. Jingwen Jin and Klara Nahrstedt. Qos specification languages for distributed multimedia applications: A survey and taxonomy. *IEEE MultiMedia*, 11(3):74–87, 2004.
4. Ulrich Kermer, Jamey Hicks, and James Rehg. A compilation framework for power and energy management on mobile computers. In *14th International Workshop on Parallel Computing*, pages 115–131, 2001.
5. Averill M. Law and W. David Kelton. *Simulation modeling and analysis*. McGraw-Hill, 3rd edition, 2000.
6. Makoto Matsumoto and Takuji 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.
7. Luís Nogueira and Luís Miguel Pinho. Dynamic qos-aware coalition formation. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, page 135, Denver, Colorado, April 2005.
8. Luís Nogueira and Luís Miguel Pinho. Dynamic adaptation of stability periods for service level agreements. In *Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 77–81, Sydney, Australia, August 2006.
9. Luís Nogueira and Luís Miguel 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.
10. Luís Nogueira and Luís Miguel 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.
11. Luís Nogueira and Luís Miguel 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.
12. Cheng Wang and Zhiyuan 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.
13. Shlomo Zilberstein. *Operational Rationality Through Compilation of Anytime Algorithms*. PhD thesis, Department of Computer Science, University of California at Berkeley, 1993.
14. Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *Artificial Inteligence Magazine*, 17(3):73–83, 1996.