Technical Report

Towards a Flexible and Dynamic Replication Control for Distributed Real-Time Embedded Systems with QoS Interdependencies

Luis Miguel Nogueira

Luis Miguel Pinho

Jorge Coelho

HURRAY-TR-100202

Version:

Date: 02-16-2010

Towards a Flexible and Dynamic Replication Control for Distributed Real-Time Embedded Systems with QoS Interdependencies

Luis Miguel Nogueira, Luis Miguel Pinho, Jorge Coelho

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail:

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

Abstract

Replication is a proven concept for increasing the availability of distributed systems. However, actively replicating every software component in distributed embedded systems may not be a feasible approach. Not only the available resources are often limited, but also the imposed overhead could significantly degrade the system's performance. The paper proposes heuristics to dynamically determine which components to replicate based on their significance to the system as a whole, its consequent number of passive replicas, and where to place those replicas in the network. The results show that the proposed heuristics achieve a reasonably higher system's availability than static offline decisions when lower replication ratios are imposed due to resource or cost limitations.

The paper introduces a novel approach to coordinate the activation of passive replicas in interdependent distributed environments. The proposed distributed coordination model reduces the complexity of the needed interactions among nodes and is faster to converge to a globally acceptable solution than a traditional centralised approach.

Towards a Flexible and Dynamic Replication Control for Distributed Real-Time Embedded Systems with QoS Interdependencies

Luís Nogueira, Luís Miguel Pinho, Jorge Coelho
CISTER Research Centre
School of Engineering, Polytechnic Institute of Porto, Portugal
{*lmn,lmp,jmn*}*@isep.ipp.pt*

February 16, 2010

Abstract

Replication is a proven concept for increasing the availability of distributed systems. However, actively replicating every software component in distributed embedded systems may not be a feasible approach. Not only the available resources are often limited, but also the imposed overhead could significantly degrade the system's performance. The paper proposes heuristics to dynamically determine which components to replicate based on their significance to the system as a whole, its consequent number of passive replicas, and where to place those replicas in the network. The results show that the proposed heuristics achieve a reasonably higher system's availability than static offline decisions when lower replication ratios are imposed due to resource or cost limitations.

The paper introduces a novel approach to coordinate the activation of passive replicas in interdependent distributed environments. The proposed distributed coordination model reduces the complexity of the needed interactions among nodes and is faster to converge to a globally acceptable solution than a traditional centralised approach.

1 Introduction

The highly dynamic and unpredictable nature of open distributed real-time embedded systems can lead to a highly volatile environment where QoS provision needs to adapt seamlessly to changing resource levels [2].Some of the difficulties arise from the fact that the mix of independently developed applications and their aggregate resource and timing requirements are unknown until runtime, but, still, a timely answer to events must be provided in order to guarantee a desired level of performance.

Our previous work [23] applied concepts of cooperative QoS-aware computing to address such challenges, emerging as a promising distributed computing paradigm to face the stringent demands on resources and performance of new embedded real-time

systems. Service-based approaches provide the needed flexibility, supporting dynamic service composition, online QoS management, and load balancing. Available software components can be shared among different coalitions of nodes and can be adapted at runtime to varying operational conditions, enhancing the efficiency in the use of the available resources.

Nevertheless, it is imperative to accept that failures can and will occur, even in meticulously designed systems, and design proper measures to counter those failures [17]. As discussed in [16], software replication in distributed environments has some advantages over other fault-tolerance solutions, providing the shortest recovery delays, it is less intrusive with respect to execution time, it scales much better, and is relatively generic and transparent to the application domain.

Traditionally, replication is decided offline and applied statically. This approach is suitable for systems where the importance of the components is well defined and remains stable during execution. However, the open and dynamic nature of service-based distributed embedded systems makes it very difficult to identify in advance the most critical software components. As a consequence, early design decisions on where and how to apply fault-tolerance techniques may turn out inadequate. At the same time, actively replicating all software components independently of their significance to the overall system may be infeasible in embedded systems due to the scale of their timing, cost, and resource constraints [6].

This paper is then motivated by the need to develop a flexible and cost-effective fault-tolerance solution with a significant lower overhead compared to a strict active redundancy-based approach. The term cost-effective implies that we want to achieve high error coverage with the minimum amount of redundancy. The paper proposes low runtime complexity heuristics to (i) dynamically determine which components to replicate based on their significance to the system as a whole; (ii) determine a number of replicas proportional to the components' significance degree; and (iii) select the location of those replicas based on collected information about the nodes' availability as the system progresses. To quantitatively study the effectiveness of the proposed approach an extensive number of simulation runs was analysed. The results show that even simple heuristics with low runtime complexity can achieve a reasonably higher system's availability than static offline decisions when lower replication ratios are imposed due to resource or cost limitations.

This paper also tackles the challenging problem of activating backup replicas in distributed interdependent environments. Consider the case where the quality of the produced output of a particular component depends no only on the amount and type of used resources but also on the quality of the inputs being sent by other components in the system [32]. If a primary replica is found to be faulty, a new primary must be elected from the set of passive backup ones and the execution restarted from the last saved state. However, it is not guaranteed that the new primary will be able to obtain the needed resources to output the same QoS level that was being produced by the old primary. In such cases, the need of coordination arises in order to preserve the correct functionality of the distributed execution [1, 12]. This paper proposes a distributed coordination protocol that rapidly converges to a new globally consistent service solution by (i) reducing the needed interactions among nodes; and (ii) compensating for a decrease in input quality by an increase in the amount of used resources in key components in

interdependency graphs.

2 System model

We understand a service $S = \{c_1, c_2, \ldots, c_n\}$ as a set of software components c_i being cooperatively executed by a coalition of nodes. Each component c_i is an entity that is defined by its functionality, is able to send and receive messages, is available at a certain point of the network, and has a set of QoS parameters that can be changed in order to adapt service provisioning to a dynamically changing environment.

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

There may exist QoS interdependencies among two or more of the multiple QoS dimensions of a service S, both within a component and among components. Given two QoS dimensions, Q_a and Q_b, a QoS dimension, Q_a, is said to be dependent on another dimension Q_b if a change along the dimension Q_b will increase the needed resource demand to achieve the quality level previously achieved along Q_a [28]. Furthermore, we consider the existence of feasible QoS regions [32]. A region of output quality $[q(o)_1, q(o)_2]$ is defined as the QoS level that can be provided by a component when provided with sufficient input quality and resources. Within a QoS region, it may be possible to keep the current output quality by compensating for a decrease in input quality by an increase in the amount of used resources or vice versa.

Users provide a single specification of their own range of QoS preferences Q for a complete service S, ranging from a desired QoS level $L_{desired}$ to the maximum tolerable service degradation, specified by a minimum acceptable QoS level $L_{minimum}$, without having to understand the individual components that make up the service. Nodes dynamically group themselves into a new coalition, cooperatively allocating resources to each new service and establishing an initial Service Level Agreement (SLA) that maximises the satisfaction of the user's QoS constraints associated with the new service while minimises the impact on the global system's QoS caused by the new service's arrival [23]. Within a coalition, each component $c_i \in S$ will then be executed at a QoS level $L_{minimum} \leq Q^i_{val} \leq L_{desired}$ at a node n_i. This relation is represented by a triple (n_i, c_i, Q^i_{val}).

The set of QoS interdependencies among components $c_i \in S$ is represented as a connected graph $\mathcal{G}_\mathcal{S} = (\mathcal{V}_S, \mathcal{E}_S)$, on top of the service's distribution graph, where each vertex $v_i \in \mathcal{V}_\mathcal{S}$ represents a component c_i and a directed edge $e_i \in \mathcal{E}_\mathcal{S}$ from c_j to c_k indicates that c_k is functionally dependent on c_j. Within $\mathcal{G}_\mathcal{S} = (\mathcal{V}_S, \mathcal{E}_S)$, we call *cut-vertex* to a component $c_i \in \mathcal{V}_\mathcal{S}$, if the removal of that component divides $\mathcal{G}_S$ in two separate connected graphs.

Each component c_i is only aware of the set of inputs $I_{c_i} = \{(c_j, Q^j_{val}), \ldots, (c_k, Q^k_{val})\}$, describing the quality of all of its inputs coming from precedent components in $\mathcal{G}_\mathcal{W}$ and the set of outputs $O_{c_i} = \{(c_l, Q^l_{val}), \ldots, (c_p, Q^p_{val})\}$, describing the quality of all of its outputs sent to its successor components in $\mathcal{G}_\mathcal{W}$. As such, no global knowledge is

required.

3 Towards a flexible and adaptive replication control

The possibility of partial failures is a fundamental characteristic of distributed applications, even more so in open environments. A sub-domain of reliability, fault-tolerance aims at allowing a system to survive in spite of faults, *i.e.* after a fault has occurred, by means of redundancy. In this paper, we consider a failure to be when a component stops producing output.

Replication is an effective way to achieve fault tolerance for such type of failure [27]. In fault-tolerant real-time systems, using active replication schemes, where several replicas run simultaneously, has been common [25]. Even if errors are detected in some of the replicas, the non-erroneous replicas will still be able to produce results within the deadlines. On the negative side, running several replicas simultaneously is costly and can be infeasible in distributed embedded systems [6]. On the other hand, passive replication [5] minimises resource consumption by only activating redundant replicas in case of failures, as typically providing and applying state updates is less resource demanding than requesting execution. As such, passive replication is appealing for soft real-time systems that cannot afford the cost of maintaining active replicas and tolerate an increased recovery time [3]. Nevertheless, it may still be possible to tolerate faults within deadlines, thus improving the system's reliability without using a more resource consuming fault-tolerance mechanism [34].

However, most of the existing solutions for fault-tolerance are usually designed and configured at design time, explicitly and statically identifying the most critical components and their number of replicas, lacking the needed flexibility to handle the runtime dynamics of open distributed real-time embedded systems [30]. Distributed real-time embedded systems often consist of several independently developed components, shared across applications and whose criticality may evolve dynamically during the course of computation. As such, offline decisions on the number and allocation of replicas may be inadequate after the system has been executing for some time. Moreover, the available resources are often limited, which means that simultaneous replication of all the components may not be feasible or desirable due to the excessive overhead.

Consequently, the problem consists in finding a replication scheme which minimises the probability of failure of the most important components without replicating every software component. This involves the study of mechanisms to determine which components should be replicated, the quantity of replicas to be made, and where to deploy such replicas [19]. The benefits of replication in open, dynamic, resource-constrained environments are a complex function of the number of replicas, the placement of those replicas, the selected replica consistency protocol, and the availability and performance characteristics of the nodes and networks composing the system. Since replica consistency protocols are relatively well understood [18, 8, 30], we will not consider them in the remainder of this paper.

Assuming that a mechanism exists for keeping passive replicas consistent, how can we make use of passive replication for increasing the reliability of distributed resource-

constrained embedded systems where it may not be possible to replicate every available component? Our approach is based on the concept of significance, a value associated to each component which reflects the effects of its failure on the overall system. Intuitively, the more a component $c_i \in S$ has other components depending on it, the more it is significant to the system as a whole. Thus, the significance degree w_i of a component c_i at a given time t is periodically computed as the aggregation of the interdependencies of other components on it, determining the usefulness of its outputs to all the components which depend on it to perform their tasks.

More formally, given $S_{\mathcal{G}} = \{\mathcal{G}_1, \ldots, \mathcal{G}_n\}$, the set of connected graphs of interdependencies between components for a given system, and $\mathcal{O}_{\mathcal{G}_j}(c_i)$, the out-degree of a node $c_i \in \mathcal{G}_j$, the significance of c_i is given by Equation 1.

$$w_i = \sum_{k=1}^{n} \mathcal{O}_{\mathcal{G}_k}(c_i) \tag{1}$$

Once the significance of each component to the system has been estimated, the decision on which components to replicate and the correspondent number of passive replicas must be taken. We propose to compute, through Equation 2, the number of replicas that should be generated for a component c_i, which is directly proportional to the component's significance degree w_i and to the maximum number of possible replicas max_{c_i} and inversely proportional to the sum of the significance degree of all components in the system W. max_{c_i} is given by the number of nodes in a heterogeneous environment which have the needed type of resources to execute the component c_i.

$$n^{c_i} = \left| \frac{w_i * max_{c_i}}{W} \right| \tag{2}$$

Having determined the number of replicas for each component, a strategy for placing them in the network is needed. Consider the effects of placing replicas on unreliable nodes. The resulting unreliability of those replicas will usually require replica consistency protocols to work harder [30], increasing network traffic and processing overheads. Thus, not only will the system's performance suffer but its availability may actually decrease, despite the increased number of components [18]. Consequently, several strategies for replicas' placement have been investigated, independently of the followed replication approach. In the context of distributed embedded systems, the impact of different allocation heuristics has been studied in [33] and a quantitative survey on a QoS-aware replica placement can be found in [13]. Nevertheless, we believe that a dynamic allocation of replicas based on collected information about the nodes' behaviour as the system progresses and evolves will achieve a better performance than would be possible with static allocation approaches.

Two gross measures of the reliability of a node are its Mean Time To Failure (MTTF) and its Mean Time To Recovery (MTTR) [19]. We propose to use those measures to allocate the set of replicas of a component c_i based on the expected availability of nodes in the system. The utility $0 \leq u_k^{r_j^i} \leq 1$ of allocating a passive replica r_j^i of a component c_i to a node n_k is then defined by the probability of its availability during

the system's execution, given by Equation 3. Utilities range from zero, the value of a completely unavailable node, to one, the value of a totally available node.

$$u_k^{r_j^i} = \frac{MTTF_k}{MTTF_k + MMTR_k} \tag{3}$$

Having the utility of each possible allocation, the probability of failure of a given set of replicas $R_i = r_1^i, r_2^i, \ldots, r_{n^{c_i}}^i$ is determined by Equation 4.

$$F(R_i) = (1 - u_1^i) * (1 - u_2^i) * \ldots * (1 - u_{n^{c_i}}^i) \tag{4}$$

The system will then allocate the set of replicas $R_i = r_1^i, r_2^i, \ldots, r_{n^{c_i}}^i$ such that its probability of failure $F(R_i)$ is minimal among all the possible allocation sets. In order to keep this allocation as up-to-date as possible, nodes have to be monitored as the system runs. If reliability of a replica set strays outside a predefined tolerance value a reconfiguration of the set is required.

4 Coordinated activation of passive replicas

One of the advantages of passive replication is that it can be implemented without the use of complex replica consistency protocols [30, 8]. Since only the primary replica processes any requests, it propagates any state changes to all alive backups, trivially ensuring order through message numbering [8]. In our system, it means that whenever the primary replica of a component c_i updates its QoS level in response to dynamical environmental changes [22], such state changes are propagated to all backup replicas in the system.

Nevertheless, one of the disadvantages of passive replication is the overhead taken to elect a new primary among the set of backups after a failure. This is even more challenging when activating replicas in interdependent cooperative coalitions where the output produced by a component may depend not only on the amount and type of used resources but also on the quality of the received inputs [22]. Ideally, when a primary fails (a failure detector [7] is assumed) a backup which is able to obtain the needed resources to output the same QoS level that was being produced by the old primary replica is selected as the new primary. However, due to the heterogeneity and dynamically varying workloads of nodes in the system, is not guaranteed that at least one of the backups will be able to output such quality level. Such feasibility is determined by the anytime local QoS optimisation algorithm of [23], which aims to minimise the impact of the activation of a new component on the currently provided QoS level of other components at a particular node.

Whenever the required QoS level cannot be assured by the new primary replica there is a need to ensure that individual substitutions of a component will produce a globally acceptable solution for the entire distributed service [14]. While there has been a great deal of research in several aspects of runtime coordination in embedded real-time systems [10, 15, 9, 4, 11], to the best of our knowledge we are the first to address the specific problem of coordinating the activation of passive replicas in interdepedent distributed environments with real-time constraints. Here, the term *coordi-*

nated activation refers to the ability of a distributed system to invoke adaptive actions on multiple nodes in a coordinated manner so as to achieve a common goal.

With the increasing size and complexity of open embedded systems the ability to build self-managed distributed systems using centralised coordination models is reaching its limits [21], as solutions they produce require too much global knowledge. Without a central coordination entity, the collective adaptation behaviour must emerge from local interactions among components. This is typically accomplished through the exchange of multiple messages to ensure that all involved components make the same decision about whether and how to adapt. One main challenge is controlling this exchange of information in order to achieve a convergence to a globally consistent solution without overflowing components with messages. Furthermore, with some decentralised coordination models it becomes difficult to predict the exact behaviour of the system taken as a whole because of the large number of possible non-deterministic ways in which the system can behave [31].

Whenever real-time decision making is in order, a timely answer to events suggests that after some finite and bounded time the global adaptation process converges to a consistent solution. We propose to achieve a time-bounded convergence to a global solution through a regulated decentralised coordination protocol defined by the following phases:

1. **New primary selection.** Let Q^i_{val} be the QoS level that was being outputted by the primary replica of component c_i that has failed. If no passive replica of c_i is able to output the same QoS level, select the one which is able to output the QoS level $Q^i_{val'} < Q^i_{val}$ closer to Q^i_{val}. A coordination message is sent to affected partners in the coalition.

2. **Local adaptation.** Affected partners, executing any interdependent component $c_j \in S$, become aware of the new output values $Q^i_{val'}$ of c_i and recompute their local set of SLAs using the anytime QoS optmisation approach of [23]. We assume that coalition partners are willing to collaborate in order to achieve a global coalition's consistency, even if this might reduce the utility of their local optimisations.

3. **Coordinated adaptation.** Affected partners by the decrease to $Q^i_{val'}$ in the path to the next cut-vertex c_c may be able to continue to output their current QoS level despite the downgraded input by compensating with an increased resource usage while others may not. If the next cut-vertex c_c is unable maintain its current QoS level then all the precedent components c_j which are compensating their downgraded inputs with an increased resource usage can downgrade to $Q^j_{val'}$ since their effort is useless.

Note that, if a component c_j, despite the change in the current quality of some or all of its inputs, is able to maintain its current QoS level there is no need to further propagate the required coordination along the dependency graph $\mathcal{G}_{\mathcal{W}}$. Thus, a *cut-vertex* is a key component in our approach.

4.1 Properties of the coordination model

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

Definition 4.1 *Given a connect graph $\mathcal{G}_S = (\mathcal{V}_S, \mathcal{E}_S)$ and given two components $c_i, c_j \in \mathcal{V}_S$, we obtain all the components in the possible paths between c_i and c_j as the result of the function:*

$$
\begin{array}{ll}
m\_paths(c_i, c_j) = & flatten(m\_paths(c_i, c_j, \emptyset)) \\
 & \\
m\_paths(c_i, c_j, T) = & \emptyset, \textit{ if } c_i = c_j \\
m\_paths(c_i, c_j, T) = & \{\{c_i, c_{k_1}\} \\
 & \cup\, m\_paths(c_{k_1}, w_j, T \cup \{c_{k_1}\}), \\
 & \ldots \\
 & \{c_i, c_{k_n}\} \\
 & \cup\, m\_paths(c_{k_n}, c_j, T \cup \{c_{k_n}\})\}, \\
 & \forall c_{k_m} \in \mathcal{V}_S, \textit{ such that} \\
 & (c_i, c_{k_m}) \in \mathcal{E}_S \textit{ and } c_{k_m} \notin T \\
m\_paths(c_i, c_j, T) = & \bot
\end{array}
$$

Definition 4.2 *Given a set A containing other sets, the function* flatten(A) *is defined as:*

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

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

Proposition 4.1 *Given a connected graph $\mathcal{G}_S = (\mathcal{V}_S, \mathcal{E}_S)$ and two components $c_i, c_j \in \mathcal{V}_S$, $m\_paths(c_i, c_j, \emptyset)$ terminates and returns all the components in the possible paths between c_i and c_j, $\emptyset$ in case $c_i = c_j$, or $\bot$ in case there is no path between $c_i, c_j \in \mathcal{V}_S$.*

Definition 4.3 *Given a node n_i, a component c_i , the set of local SLAs $\sigma = \{SLA_{w_0}, \ldots, SLA_{w_p}\}$ for the p locally executed components, $Q_{val'}$ as the new imposed QoS level for c_i, and $I_{c_i} = \{(n_j, c_j, Q^j_{val}), \ldots, (n_k, c_k, Q^k_{val})\}$ as the set of QoS levels given as input to c_i, then the value of $test\_feasibility(n_i, w_i, Q^i_{val'}, I_{w_i})$ is the return value of QoS optimisation of [23] applied to node n_i.*

Lemma 4.1 (Correctness of the feasibility test) *The function $test\_feasibility$ always terminates and returns true if the new required set of SLAs for outputting the QoS level Q'_{val} at work unit w_i is feasible or false otherwise.*

Proof 4.1 *Termination comes from the finite number of tasks τ_i being executed in node n_i and from the finite number of the k QoS dimensions and j attributes being tested. The number of QoS attributes being manipulated decreases whenever a task τ_i is configured to be served at its lowest admissible QoS level $Q_{kj}[n]$, thus leading to termination.*

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

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

Definition 4.4 *Given a connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the component c_i and $I = \{(c_j, Q^j_{val}), \ldots, (c_k, Q^k_{val})\}$ as the current set of QoS inputs for a component c_i, and given T as the set of changed QoS inputs in response to the coordination request, the function $update(I, T)$ updates I with the elements from T:*

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

Proposition 4.2 *Given two sets I and T, both with elements of the form (c_i, Q^i_{val}),* update(I,T) *terminates and returns a new set with the elements of I such that whenever $(c_i, Q^i_{current}) \in I$ and $(c_i, Q^i_{new}) \in T$ the pair stored in the returned set is (c_i, Q^i_{new}).*

Definition 4.5 *Given a component c_i, we define the function $get\_input\_qos(c_i)$ as returning the set of elements (c_j, Q^j_{val}), where each of these elements represents a component with an output QoS level of Q^j_{val} used as an input of the component c_i.*

Definition 4.6 *Given a node n and a component c_i and QoS level Q^i_{val}, we define the function $set\_qos\_level(n, c_i, Q_{val})$ as setting the QoS level currently being used to process the compoenent c_i by node n to Q^i_{val}.*

Given these, the next section details how the proposed coordination model operates on updates of the currently supplied QoS level after an activation of a passive replica of an interdependent component $c_i \in S$.

4.2 Coordination of replica activation

Given the connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with a set of cut-vertices $\mathcal{C}$ and an end-user component c_u receiving the final outcome of the coalition's processing of service S, whenever a component $c_i \in \mathcal{V}$ is replaced by a replica r_k^i which needs to decrease the quality of the output from its current QoS level of Q_{val} to a lower QoS level Q'_{val} due to limitations of the selected replica, the other nodes in the coalition respond to this request according to Algorithm 1. Note that the set $\mathcal{C}'$ is the set of cut-vertices between c_i and c_u.

Algorithm 1 Coordinating Replica Activation

1: $temp := r_k^i$
2: **for each** $c_c \in \mathcal{C}' \cup \{c_u\}$ **do**
3: **if** $service\_stabilization(temp, c_c, \mathcal{G}, Q'_{val}) =$ **FALSE then**
4: $temp := c_c$
5: **else**
6: Replica activation keeps the previous global output Q_{val}
7: **return**
8: **end if**
9: **end for**

Definition 4.7 *Given the connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with a set of cut-vertices $\mathcal{C}$ and the subgraph that connects component c_i to next cut-vertex $c_c \in \mathcal{C}$, the function $service\_stabilization(c_i, c_c, \mathcal{G}, Q'_{val})$ is defined by:*

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

Lemma 4.2 *Given the connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ such that $c_i \in \mathcal{V}$ and $c_j \in \mathcal{V}$ and c_j currently outputs a QoS level Q_{val}, the call to $service\_stabilization(c_i, c_j, \mathcal{G}, Q'_{val})$ terminates and returns true if c_j is able to keep its current output level Q_{val} or false otherwise.*

Proof 4.2 *Since $\mathcal{V}$ is a finite set and since, by Proposition 4.1, m_paths terminates and by Proposition 4.2 $update$ terminates, the number of iterations is finite due to the finite number of elements in the paths. Thus, $service\_stabilization$ terminates.*

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

□

Theorem 4.1 (Correctness of Coordinating Replica Activation) *Given the connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ representing the QoS inter-dependencies of a service S being executed by a coalition of components such that $c_u \in \mathcal{V}$ is the end-user node receiving S at the QoS level Q_{val}, whenever a node c_i is replaced by a replica r^i_k which forces the decrease of the quality of the output from its current QoS level of Q_{val} to a degraded QoS level Q'_{val}, Algorithm 1 changes the set of SLAs at components in $\mathcal{G}$ such that c_u continues to receive S at its current QoS level Q_{val} or sets all nodes to a degraded QoS level of Q'_{val}.*

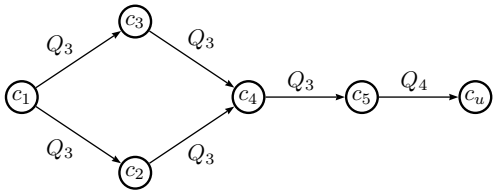
Proof 4.3 *Termination comes from the finite number of elements in $\mathcal{C} \cup \{c_u\}$ and from Lemma 4.2.*

The correctness trivially follows by the correctness of Lemma 4.2 and by induction on the number of elements in $\mathcal{C} \cup \{c_u\}$.

4.3 Example

Let's consider a simple coalition represented by a graph, where each component is labelled with letter C and the edges with a pair containing the outputted QoS level by that component. The list of properties of each node and graph follows:

Component	Output	Input
c_1	Q_3	$\emptyset$
c_2	Q_3	$\{(c_1, Q_3)\}$
c_3	Q_3	$\{(c_1, Q_3)\}$
c_4	Q_3	$\{(c_2, Q_3), (c_3, Q_3)\}$
c_5	Q_4	$\{(c_4, Q_3)\}$



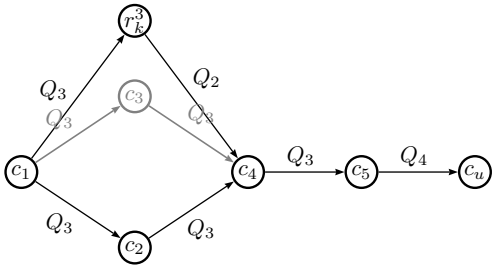
The component receiving the output here is component c_u. Now, suppose that component c_3 becomes offline. This results in c_1 sending its output to a selected replica $r_k^3 \in R_3$ where R_3 is the set of available replicas of c_3. Two different scenarios may occur:

1. r_k^3 is able to output the same QoS level and thus the graph stays the same as before with c_3 replaced by r_k^3.

2. r_k^3 is unable to output the same QoS level and now the coordination of replica activation takes place in order to maintain the QoS level to its maximum output.

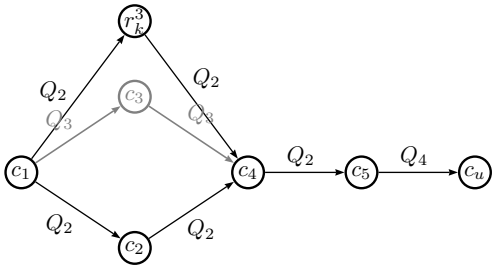
   The first cut-vertex found is c_4. And now, if c_4 is able to output the previous QoS value of Q_3 even with a degraded input, the replica activation coordination stops and the graph becomes:

Component	Output	Input
c_1	Q_3	$\emptyset$
c_2	Q_3	$\{(c_1, Q_3)\}$
r_k^3	Q_2	$\{(c_1, Q_3)\}$
c_4	Q_3	$\{(c_2, Q_3), (r_k^3, Q_2)\}$
c_5	Q_4	$\{(c_4, Q_4)\}$



   If on the other hand, c_4 is unable to keep the same output, then all the previous nodes decrease their output quality since it is unnecessary to keep the same QoS level with c_4 acting like a bottleneck. The next step is seeing if the next cut-vertex, in this case c_5, is able to maintain the same QoS with the degraded input. Suppose it can, the graph becomes:

Component	Output	Input
c_1	Q_2	$\emptyset$
c_2	Q_2	$\{(c_1, Q_2)\}$
r_k^3	Q_2	$\{(c_1, Q_2)\}$
c_4	Q_2	$\{(c_2, Q_2), (r_k^3, Q_2)\}$
c_5	Q_4	$\{(c_4, Q_2)\}$



5 Evaluation

An application that captures, compresses and transmits frames of video to end users, which may use a diversity of end devices and have different sets of QoS preferences, was used to evaluate the efficiency of the proposed passive replication mechanism, with a special attention being devoted to introduce a high variability in the characteristics of the considered scenarios. The application is composed by a set of components to collect the data, a set of compression components to gather and compress the data sent from multiple sources, a set of transmission components to transmit the data over the network, a set of decompression components to convert the data into the user's specified format, and a set of components to display the data in the end device [23].

The number of simultaneous nodes in the system randomly varied, in each simulation run, from 10 to 100. For each node, the type and amount of available resources, creating a distributed heterogeneous environment. Nodes failed and recovered according to their MMTF and MTTR reliability values, which were randomly assigned when the nodes were created (it was ensured that each node had an availability between 60% and 99%).

Each node was running a prototype implementation of the CooperatES framework [26], with a fixed set of mappings between requested QoS levels and resource requirements. At randomly selected nodes, new service requests from 5 to 20 simultaneous users were randomly generated, dynamically generating different amounts of load and resource availability. Based on each user's service request, coalitions of 4 to 20 components were formed [23] and a randomly percentage of the connections among those components was selected as a QoS interdependency.

In order to assess the efficiency of the proposed dynamic replication control as opposed to an offline static replication in dynamic resource-constrained environments, we considered the number of coalitions which where able to recover from failures and conclude their cooperative executions as a function of the used replication ratio. The

reported results were observed from multiple and independent simulation runs, with initial conditions and parameters, but different seeds for the random values[1] used to drive the simulations, obtaining independent and identically distributed variables. The mean values of all generated samples were used to produce the charts.

In the first study, we evaluated the achieved system's availability with the proposed dynamic replication control based on components' significance and with a static offline approach in which the components to replicate and their number of replicas is fixed by the system's designer at a coalition's initialisation phase [19]. At each simulation run, if the primary replica of a component c_i failed during operation, a new primary was selected among the set of passive backups. If this was not possible, all the coalitions depending on c_i were aborted. In this study, replicas were also randomly allocated among eligible nodes with the dynamic replication control policy.

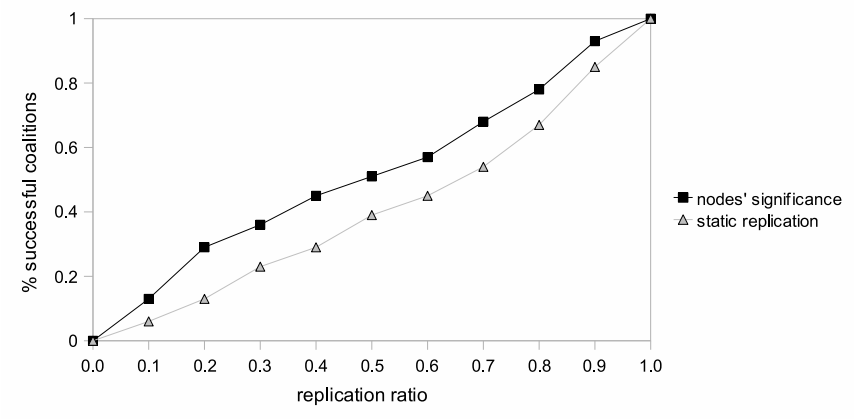


Figure 1: Impact of the chosen replication control strategy on the system's availability

Figure 1 clearly shows that our strategy is more accurate to determine and replicate the most significant components than a static offline one, particularly with lower replication ratios. Thus, when lower replication ratios are imposed due to resource or cost limitations, a higher availability can be achieved if the selection of which components to replicate and their number of replicas depends on their significance to the system as a whole. In open and dynamic environments, such significance can be determined online as the aggregation of all the other components that depend on a particular component to perform their tasks.

A second study evaluated the impact of the selected replicas' placement strategy on the achieved system's availability for a given replication ratio. The study compared the performance of the proposed allocation heuristic based on collected information about the nodes' availability as the system evolves with a random policy in which the placement of the generated replicas is fixed offline [24]. The decision on which components to replicate and their number of replicas followed the same dynamic and

[1]The random values were generated by the Mersenne Twister algorithm [20].

static approaches of the first study.

For the dynamic allocation strategy, a tolerance value for the availability of each replica set was randomly generated at each simulation run. If this tolerance was surpassed, a reassignment of replicas was performed. Figure 2 shows that an offline policy always achieves a poorer performance than a dynamic allocation that takes into account the nodes' reliability along time.

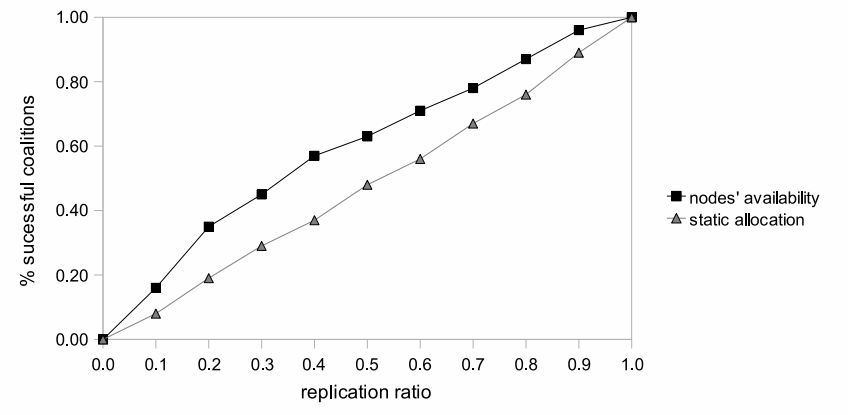


Figure 2: Impact of the chosen replica allocation strategy on the system's availability

It is then possible to conclude that the location of replicas is a relevant factor for the system's availability as a whole. A comparison of Figures 1 and 2 shows that even though an improvement in availability can be achieved by increasing the replication ratio, the impact of replicas' placement is quite significant.

A third study evaluated the efficiency of the proposed coordinated activation of interdependent passive replicas in comparison to a typical centralised coordination approach [29] in which a system-wide controller coordinates resource allocations among multiple nodes. The average results of all simulation runs for the different coalition sizes and percentages of interdependencies among components are plotted in Figure 3. As expected, both coordination approaches need more time as the complexity of the service's topology increases. Nevertheless, the proposed decentralised coordination model is faster to determine the overall coordination result in all the evaluated services' topologies, needing approximately 75% of the time spent by the centralised near-optimal model.

6 Conclusions

The availability and performance of open distributed embedded system is significantly affected by the choice ofthe replication control strategy and placement of the generated replicas. The proposed heuristics based on the components' significance to the overall system and on nodes' reliability history have a low runtime complexity and achieve a

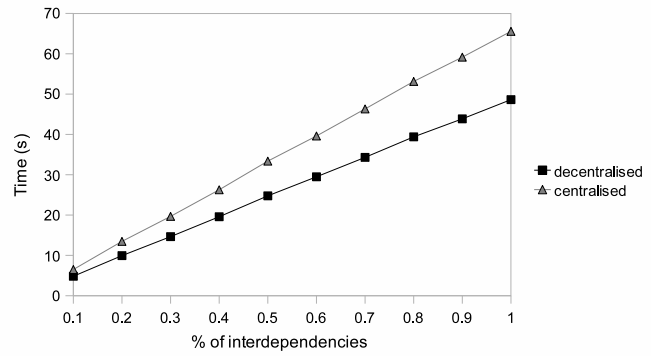


Figure 3: Time for a coordinated replica activation

reasonably higher system's availability than static offline decisions, particularly when lower replication ratios are imposed due to resource or cost limitations.

Since QoS interdependencies may exist among components of a distributed system, activating passive replicas when a primary component is found to be faulty may demand coordination. The proposed distributed coordination model enables a faster convergence to a global service solution than a typical centralised approach.

References

[1] Gabrielle Allen, Thomas Dramlitsch, Ian Foster, Nicholas T. Karonis, Matei Ripeanu, Edward Seidel, and Brian Toonen. Supporting efficient execution in heterogeneous distributed computing environments with cactus and globus. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing*, pages 52–52, November 2001.

[2] Rachid Anane. Autonomic behaviour in qos management. In *Proceedings of the Third International Conference on Autonomic and Autonomous Systems*, page 57, Athens, Greece, June 2007.

[3] Jaiganesh Balasubramanian, Sumant Tambe, Chenyang Lu, Aniruddha Gokhale, Christopher Gill, and Douglas C. Schmidt. Adaptive failover for real-time middleware with passive replication. In *Proceedings of the 15th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 118–127. IEEE Computer Society, April 2009.

[4] Craig Boutilier, Rajarshi Das, Jeffrey O. Kephart, Gerald Tesauro, and William E. Walsh. Cooperative negotiation in autonomic systems using incremental utility elicitation. In *In Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence*, pages 89–97, Acapulco,Mexico, August 2003.

[5] Navin Budhiraja, Keith Marzullo, Fred B. Schneider, and Sam Toueg. The primary-backup approach. *Distributed systems (2nd Ed.)*, pages 199–216, 1993.

[6] Zhongtang Cai, Vibhore Kumar, Brian F. Cooper, Greg Eisenhauer, Karsten Schwan, and Robert E. Strom. Utility-driven proactive management of availability in enterprise-scale information flows. In *Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware*, pages 382–403. Springer-Verlag, 2006.

[7] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.

[8] Ruben de Juan-Marin, Hendrik Decker, and Francesc D. Munoz-Esco. Revisiting hot passive replication. In *Proceedings of the 2nd International Conference on Availability, Reliability and Security*, pages 93–102, April 2007.

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

[10] Marco Dorigo and Gianni Di Caro. The ant colony optimization meta-heuristic. *New ideas in optimization*, pages 11–32, 1999.

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

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

[13] Wei Fu, Nong Xiao, and Xicheng Lu. A quantitative survey on qos-aware replica placement. In *Proceedings of the 7th International Conference on Grid and Cooperative Computing*, pages 281–286, Shenzhen, China, October 2008.

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

[15] Sven Graupner, Artur Andrzejak, Vadim Kotov, and Holger Trinks. Adaptive control overlay for service management. In *First Workshop on the Design of Self-Managing Systems*, San Francisco, USA, June 2003.

[16] Rachid Guerraoui and Andre Schiper. Software-based replication for fault tolerance. *IEE Computer*, 30:68–74, 1997.

[17] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer, 1997.

[18] M. C. Little. *Object Replication in a Distributed System*. PhD thesis, Department of Computing Science, Newcastle University, September 1991.

[19] M. C. Little and D.L. McCue. The replica management system: a scheme for flexible and dynamic replication. In *Proceedings of the 2nd International Workshop on Configurable Distributed Systems*, pages 46–57, April 1994.

[20] 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.

[21] Alberto Montresor, Hein Meling, and Özalp Babaoglu. Toward self-organizing, self-repairing and resilient distributed systems. In *Future Directions in Distributed Computing*, pages 119–126, 2003.

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

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

[24] Giwon On, Jens Schmitt, and Ralf Steinmetz. Quality of availability : replica placement for widely distributed systems. In *Proceedings of the 11th International Workshop on Quality of Service*, pages 325–342, Monterey, CA, June 2003.

[25] Luís Miguel Pinho, Francisco Vasques, and Andy Wellings. Replication management in reliable real-time systems. *Real-Time Systems*, 26(3):261–296, 2004.

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

[27] David Powell, editor. *A generic fault-tolerant architecture for real-time dependable systems*. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

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

[29] Kurt Rohloff, Richard Schantz, and Yarom Gabay. High-level dynamic resource management for distributed, real-time embedded systems. In *Proceedings of the 2007 Summer Computer Simulation Conference*, pages 749–756, July 2007.

[30] Paul Rubel, Matthew Gillen, Joseph Loyall, Richard Schantz, Aniruddha Gokhale, Jaiganesh Balasubramanian, Aaron Paulos, and Priya Narasimhan. Fault tolerant approaches for distributed real-time and embedded systems. In *Proceedings of the 2007 Military Communications Conference*, pages 1–8, Orlando, Florida, USA, October 2007.

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

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

[33] Thilo Streichert, Michael Glaß, Rolf Wanka, Christian Haubelt, and Jürgen Teich. Topology-aware replica placement in fault-tolerant embedded networks. In *Proceedings of the 21st International Conference on Architecture of Computing Systems*, pages 23–37, Dresden, Germany, February 2008.

[34] Asmund Tjora and Amund Skavhaug. A general mathematical model for runtime distributions in a passively replicated fault tolerant system. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, pages 295–301, Porto,Portugal, July 2003.