Topology-Aware Replica Placement in Fault-Tolerant Embedded Networks

Thilo Streichert, Michael Glaß, Rolf Wanka, Christian Haubelt, Jürgen Teich
{streichert,glass,wanka,haubelt,teich}@cs.fau.de
Department of Computer Science 12
University of Erlangen-Nuremberg, Germany

Abstract. Application details uncertain at design time as well as tolerance against permanent resource defects demand flexibility and redundancy. In this context, we present a strategy for placing replicas in embedded point-to-point networks where link as well as node defects may occur at runtime. The proposed strategies for replica placement are based on the partitioning of the network into biconnected components. We are able to distinguish between different replication strategies, i.e., active and passive replication. Our experimental results show that the reliability improvement due to the proposed replica placement strategies is up to 23% compared to a randomized strategy.

1 Introduction

Many networked embedded systems such as sensor networks or networks in the field of industry automation need to be flexible and extensible towards applications which are unknown at design time. In particular, new network nodes need to be integrated and new tasks are to be placed onto the computational resources in the network. On the other hand, such networks have to meet demands concerning reliability/availability at a minimum of additional monetary costs. But how is it possible to combine flexibility with reliability at a minimum of extra monetary cost? Traditional approaches try to treat transient and permanent faults by introducing spatial or temporal redundancy at design time [10] or by applying coding techniques [11]. In this contribution, we will propose an online methodology that replicates the tasks executed by the network nodes in order to tolerate permanent fail-silent faults of nodes and links. These replicated tasks will be dynamically placed onto network nodes using underutilized computational reserves. That is, we need not explicitly extend the network with a node and a spare node providing the same functionality. Instead, the functionality of a new node or a new application will be distributed over the network making use of free computational reserves in the network, i.e., reducing the amount of extra cost. Moreover, it is possible to tolerate several subsequent permanent defects, whereas static systems might tolerate only as many defects as spares are available.

Using a two-phase online methodology (Fig. 1), we are able to integrate new tasks into the network and to treat resource defects. After a defect of a node or link in the network, the online methodology tries to activate replicas and reroute the communication paths in a *fast repair* phase. The second phase called *optimization phase* tries 1.) to optimize the binding of the tasks and 2.) to place replicas in order to tolerate further resource defects. In previous publications, we investigated methodologies and their performance for the *fast repair* phase [13] and the optimization of the tasks binding [20, 18, 19]. In this paper, we focus on the algorithmic aspects of placement of replicas in order to tolerate node or link defects.

As an introductory example, consider the networks and placement of tasks in Fig. 2. Two cases are shown which are completely different in their reliability. Both cases show a network with six nodes $n_1, \ldots, n_6$ and two communicating tasks t_1 and t_2, each having a unique replica t'_1 and t'_2, respectively. If in Fig. 2a) node n_2, node n_4 or the link (n_2, n_4) fails, the functionality fails as well because no connected component contains both tasks. In Fig. 2b) the placement of the replicas t'_1 and t'_2 differs such that each node and link defect, even n_2, n_4 as well as the link (n_2, n_4) may occur without loosing the functionality. Thus, the reliability of the entire network depends heavily on the placement strategy of replicated tasks and the underlying topology.

Therefore, we will study in this paper the impact of replica placement on the network reliability and in summary, the contributions are:

1. Novel heuristics for placing replicas are presented which are based on the partitioning of networks into so-called *biconnected components*.

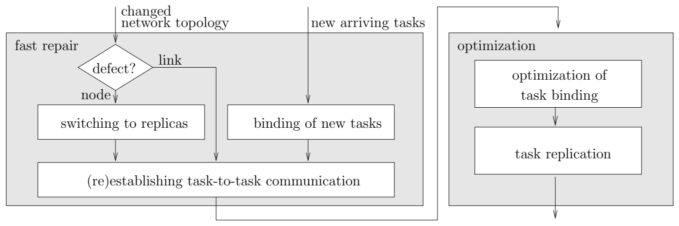


Fig. 1. In case of topology changes, the *fast repair* phase activates replicated tasks and the communication between any two tasks will be reestablished. If a new task arrives at one node in the network, the network decentrally tries to bind the task onto one of its network nodes. The optimization phase optimizes the binding of tasks and creates new replicas.

2. Differences for placing *passive replicas* [3] and *active replicas* [17] are analytically discussed and experimentally evaluated.
3. An experimental evaluation compares the presented heuristics with respect to resulting reliability properties.

The remaining paper is structured as follows: Sec. 2 presents the related work in the field of replica placement. Sec. 3 introduces our network model used throughout this paper. Sec. 4 focuses on the main contribution of this paper, i.e., on distributed replica placement techniques. The proposed replica placement approach is evaluated in Sec. 5.

2 Related Work

Replication has been a research topic in different areas, like distributed file servers, content delivery networks, and databases. In databases for instance, replication has been applied for performance reasons. In distributed systems, replication or redundancy, respectively, is usually introduced for fault-tolerance reasons and obtained by replicating tasks or services as well as components or devices.

Independent of the kind of replication, i.e., active replication or passive replication, different strategies have been investigated for the placement of replicated tasks. In the field of *content-delivery networks*, data objects are replicated on servers in order to minimize the latency in case of an access by a client. A famous approach that minimizes this latency is the *Hot Zone* algorithm [22] which is derived from the *Hot Spot* [16] algorithm. The Hot Spot algorithm attempts to place replicas close to the clients generating the highest amount of traffic. N potential sites are sorted according to this amount of traffic and on the M sites with the highest traffic replicas are placed. The Hot Zone algorithm partitions the network into regions consisting of nodes with a low latency to each other. In a second step, the Hot Zone algorithm places replicas into the most active regions.

A very good overview of different placement strategies, objectives and application areas is given in [12, 16]. The presented algorithms are executed in a centralized way with global knowledge. But interestingly, the authors of [12] see a strong relevance towards the research of distributed online-placement strategies which is the scope of this paper. They argue that distributed approaches overcome scalability problems of centralized approaches.

Heuristics which determine a placement of replicas in a distributed manner have been presented by Douceur and Wattenhofer. They studied placement strategies [5–7] based on hill-climbing algorithms which are applied in the distributed server-less file

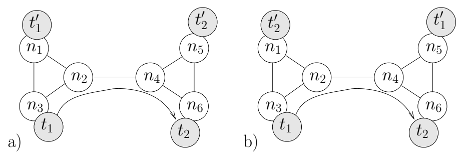


Fig. 2. In a) neither the nodes n_2, n_4 nor the link (n_2, n_4) must fail. Otherwise, the functionality provided by the communicating tasks t_1 and t_2 cannot operate any more. In b), the nodes n_2 and n_4 as well as the link (n_2, n_4) may fail without loosing functionality as the communication from task t_1 to t_2 may still be handled from replica t'_1 to t_2 or from t_1 to t'_2.

system Farsite [1]. The heuristics select replicas and swap their placement in order to iteratively increase their availability. In particular, one group of nodes contacts another group and each of it selects a file it manages. The groups then decide whether to swap the files or not. For the selection of files, three different strategies were applied: 1.) A *RandRand*-strategy swaps two randomly chosen files. 2.) A *MinRand*-strategy swaps between a minimum-availability file and any other file. 3.) A *MinMax*-strategy swaps between a minimum-availability file and a maximum-availability file. The swaps are only processed if they reduce the difference of the file availabilities.

In [4], a distributed algorithm with a loosely synchronized global database is presented which tries to take placement decisions of data objects into account by applying a randomized strategy. A more recent publication [15] proposes a so-called *sequential placement* strategy which determines a binding of a primary object with the help of a hash function calculated for that object. The idea of sequential placement is to place replicas close to each other such that failures can be detected and repaired very fast.

Unfortunately, all these heuristics fail in the context of embedded networks, since they consider only the availability of a single data object on its own. But, in the context of embedded networks an application consisting of distributed *communicating tasks* works only correctly if all tasks are available. Moreover, in the context of embedded systems based on point-to-point communication, there is a high risk that networks split up into two parts due to a defect. Therefore, in the following sections, we study novel heuristics for the placement and replication of communicating tasks.

3 Computational Model

The considered systems are specified by the *network model* which separates functionality from the network architecture. The network model consists of a *topology graph* and a *sensor-controller-actuator graph.* The third component of the network model is the set of mapping edges which denotes the binding possibilities of tasks onto network nodes. Hence, we model also heterogeneous nodes that cannot execute all tasks. The system model can be formally described as follows:

Definition 1 (Network Model). *The entire network model $M(G^{\mathrm{tg}}, G^{sca}, E_{\mathrm{m}})$ consists of a* topology graph G^{tg}*, a set of* sensor-controller-actuator chains G^{sca}*, and a set of* mapping edges E_{m}.

Definition 2 (Topology Graph). *The topology graph $G^{\mathrm{tg}}(N^{\mathrm{tg}}, E^{\mathrm{tg}})$ consists of network nodes $n_i \in N^{\mathrm{tg}}$ and edges $e_i^{\mathrm{tg}} \in E^{\mathrm{tg}} \subseteq N^{\mathrm{tg}} \cup N^{\mathrm{tg}}$. Edges e_i^{tg} between the network nodes are undirected.*

For modeling the functionality, we define a *sensor-controller-actuator graph.*

Definition 3 (Sensor-Controller-Actuator Graph). *The sensor-controller-actuator graph $G^{sca}(T^{\mathrm{sca}}, E^{\mathrm{sca}})$ consists of tasks $t_i \in T^{\mathrm{sca}}$ and edges $e_i^{sca} \in E^{sca} \subseteq T^{\mathrm{sca}} \times T^{\mathrm{sca}}$ represent the data dependencies between the tasks.*

Definition 4 (Mapping Edges). *The set of mapping edges $E_{\mathrm{m}} \subseteq T^s \times N^{\mathrm{tg}}$ relate vertices of the sensor-controller-actuator chains with the vertices of the topology graph. A mapping edge $e_{\mathrm{m}} \in E_{\mathrm{m}}$ starts at a vertex of the sensor-controller-actuator graph and ends at a vertex of the topology graph.*

A mapping edge $e_{\mathrm{m}} \in E_{\mathrm{m}}$ indicates the possible implementation of a task onto the corresponding resource. If a mapping edge $(t_i, n_i) = e_{\mathrm{m}}$ is selected, the task t_i of the sensor-controller-actuator graph is executed at the corresponding node n_i. This will be also denoted with $t_i \mapsto n_i$ in the following.

In the following, we assume that at most two mapping edges e_{m} are selected, one for the task t_i and one for the replica t'_i. The replicas can be either active or passive and throughout the reminder of the paper, we define the property of active and passive replication as follows.

Definition 5 (Active Replica). *Active replicas ${t'}_i$ are executed on a node in the same way as primary tasks t_i. Thus, the active replicas produce the same computational load c_i as the corresponding primary tasks t_i.*

Definition 6 (Passive Replica). *Passive replicas ${t'}_i$ are not executed on a node, but stored in the memory of that node. In order to work like a primary task t_i, replicated tasks need to be activated. Thus, the passive replicas produce no computational load c_i unless they are activated.*

For a precise description on the functioning of active and passive replicas, we refer to [3] and [17].

Furthermore, the replica placement methodology presented here is based on the following *failure model*:

Definition 7 (Failure Model). *Only one node or communication link may fail simultaneously and another subsequent defect of such a resource occurs in a minimum arrival time bigger than after the methodology from Fig. 1 has been processed.*

Obviously, such a defect might result in a situation where more than one resource is inaccessible due to the decomposition of a network into two or more parts. But for the presented replica placement algorithms, this assumption is very important and typically, the execution time of the online methodology is much shorter than the time between two resource defects. Thus, this assumption does not reduce the applicability of the presented approach.

4 Topology-Aware Replica Placement

According to the overall methodology presented in Fig. 1, the placement of tasks and replicas is performed in two steps with competing objectives. The placement of tasks is executed with the objective to improve the performance of an application, i.e., reduce the traffic in the network and balance the load on the network nodes such that the task response times and the overhead due to context switches are reduced. Due to the fact that the run-time behavior is of major interest, the task placement is prioritized and executed at first. Afterwards, the replica placement phase is supposed to place replicas onto the remaining computational resources in the network. That is, the replica placement phase assumes a fixed binding of tasks onto the network nodes and each task already consumes a certain part of the computational capacity of its host network node. With the objective to increase the reliability of the entire functionality executed by the network, the placement strategy binds replicated tasks onto network nodes.

In embedded networks tasks might be inoperable because the communication partners are in separated connected components of the network after a resource defect. Therefore, a novel strategy is proposed which 1.) identifies network partitions that will under no circumstances split up into disjoint components under the above mentioned conditions and 2.) places replicas t'_i with respect to these partitions. These two parts are explained in the following subsections.

4.1 Identifying Network Partitions

With the help of our failure model, it is possible to identify network regions that will under no circumstance decompose. Such components are called *biconnected components* (e.g., see [2, Sec. 5.3]):

Definition 8 (Biconnected Component). *Let $G(V, E)$ be an undirected graph where V is the set of vertices and $E \subseteq V \times V$ is the set of edges e_i. Two edges e_i, e_j are said to be* biconnected *if $e_i = e_j$ or there exists a* simple cycle *containing both e_i and e_j. A simple cycle is a path consisting of vertices $v_k, v_l, \ldots, v_k$ where no vertex except v_k occurs twice. Each two distinct edges e_i and e_j are in the same set $E_s \subseteq E$ iff they are biconnected. All vertices $v_j \in V$ incident to the edges $e \in E_s$ belong to the set $V_s \subseteq V$. A maximal subgraph $G_s = (V_s, E_s)$ is called a* biconnected component.

For the following examinations, we define a set of nodes BCC_i containing only the nodes of a biconnected component G_s: $BCC_i = V_s$. In networks with several biconnected components, the following elements might occur which are critical with respect to the reliability of the network:

Definition 9 (Articulation Point). *An* articulation point *is a vertex whose removal disconnects the graph.*

Definition 10 (Bridge). *A* bridge *is an edge whose removal disconnects the graph.*

In Fig. 3 a network with six nodes $n_1, \ldots, n_6$ is shown. The biconnected components in this network are given by the sets $V_{s1} = \{n_1, n_2, n_3\}$, $V_{s2} = \{n_4, n_5, n_6\}$, and $V_{s3} = \{n_2, n_4\}$ with their corresponding edges. Biconnected components with more than two nodes have the important property that each node is connected to any other node in the same biconnected component via at least two disjoint paths. Thus, a node defect

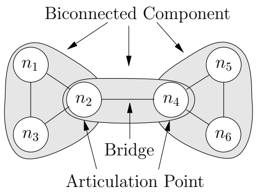


Fig. 3. Shown is a network with six nodes $n_1, \ldots, n_6$, with two articulation points n_2, n_4, one bridge (n_2, n_4), and three biconnected components given through their sets $V_{s1} = \{n_1, n_2, n_3\}$, $V_{s2} = \{n_4, n_5, n_6\}$, and $V_{s3} = \{n_2, n_4\}$ and their corresponding edges.

will result in a failure of the functionality of the defect node but the functionality of the other nodes within this biconnected component will not be inaccessible. In case of a link defect, no task bound onto nodes within the biconnected component will be inaccessible. Note that biconnected components with two elements are somehow an exception because a link between the two nodes might decompose the two nodes. On the other hand, a link defect in such a small biconnected component can be interpreted as a node defect. Hence, the biconnected component will shrink from a node's point of view but will not decompose. However, the articulation points in Fig. 3 are n_2 and n_4. If one of these nodes is faulty, the network is divided into two parts. The same holds true for the bridge between the nodes n_2 and n_4. If the link has a defect, the network is divided, too.

Algorithms for determining biconnected components have been widely studied and applied, e.g., for CAD/CAM applications or efficient search strategies in logic programming [23]. Another interesting application is presented in [21], where causal message ordering is achieved with the help of biconnected components. In this paper, we present distributed algorithms for maintaining biconnected components in a dynamically changing network for placing replicas onto network nodes.

The first approach for finding biconnected components in undirected graphs is based on depth-first search. Such a sequential algorithm which solves the problem in $\mathcal{O}(n+m)$ time where n is the number of vertices in the graph and m is the number of edges has been presented in [9][2, Sec. 5.3]. If each node in the network knows about the topology and changes to the topology are immediately announced to each node in the network, this algorithm is applicable in our replica placement strategy. A very efficient approach has been presented in [23] which allows for adapting biconnected components after an edge insertion. Unfortunately, this approach is not applicable to our case because we consider link defects, i.e., edges are deleted but not inserted. The algorithms presented in [8] and [21] are distributed algorithms that solve the problem of finding biconnected components concurrently in dynamic graphs.

4.2 Replica Placement Algorithms

After the biconnected components are identified in the network, each node hosting tasks t_i searches for other host nodes for placing the replica t'_i. This search can be simply implemented with a sort of depth-first search. This search sends a message *msg* to one neighboring node. The message *msg* consists of two ordered lists *msg.visited* containing all visited nodes and *msg.backtracking* with nodes which will be visited again if a search in a certain network direction was not successful. Additional task parameters are stored in *msg.constraints* and contain the following information: 1.) required resources like computational capacity or dedicated I/O components and 2.) the current quality of the replica placement. Based on the information in *msg.constraints*, the receiver of the message *msg* can decide whether the replica will be accepted or not. If the receiver of *msg* accepts the new replica, it sends a message back to the former host node such that the task binaries can be transferred to the new host. In Alg. 1 the search strategy for nodes is presented which is locally executed at each node in the network.

In algorithm Alg. 1 line 10, the replicas are accepted by a receiving node iff certain constraints stored in *msg.constraints* are fulfilled. It has been mentioned that these constraints cover resource constraints like required computational capacity, necessary I/O components, etc. But it is also important that these constraints contain information

Algorithm 1: In order to search for nodes which may host a replica, this depth-first search algorithm processes messages *msg*.

```
if $myNodeID$ in $msg.visited$ then
    forall $neighbors$ do
        if $neighborID$ not in $msg.visited$ then
            send $msg$ to $neighborID$;
            return;
    delete $myNodeID$ from $msg.backtracking$;
    send $msg$ to last element of $msg.backtracking$;
else
    if $msg.constraints$ are fulfilled then
        allocate resources for the replica;
        notify former replica host
        delete $msg$;
    else
        push back $myNodeID$ to $msg.visited$;
        push back $myNodeID$ to $msg.backtracking$;
        forall $neighbors$ do
            if $neighborID$ not in $msg.visited$ then
                send $msg$ to $neighborID$;
                return;
        delete $myNodeID$ from $msg.backtracking$;
        send $msg$ to last element of $msg.backtracking$;
```

about the current placement of the replicas. Therefore, criteria are explained in the following that allow for deciding whether the new replica placement is better than the current placement.

The first group of algorithms requires reliability values of computational resources in the network and respects these values during the placement process: **Max Rel:** The *Max Rel* strategy tries to place tasks onto nodes with highest reliability as long as enough computational capacity is available and other resource constraints are not violated. The host node of the replica t'_i and its task t_i must not be the same. This strategy resembles the idea from [5] explained in Sec. 2. This strategy is considered as a reference here, and the next three approaches will be compared to it.
BCC Max1 Max Rel: The *BCC Max1 Max Rel* places a replica in a biconnected component where the most adjacent tasks are located, i.e., tasks the replica communicates with after activation of the replica. Within this biconnected component, the replica is placed onto the most reliable node. As before, the host node of the replica t'_i and its task t_i must not be the same.
BCC Max2 Max Rel: The *BCC Max2 Max Rel* places a replica in a biconnected component where the most tasks are located. In this case, data dependencies between tasks are not respected. Within this biconnected component, the replica is placed onto the most reliable node. Again, the host node of the replica t'_i and its task t_i must not be the same.
BCC Task Max Rel: The *BCC Task Max Rel* places a replica t'_i in a biconnected component where the corresponding task t_i must not be located in. The idea is to distribute the tasks and replicas over the biconnected components such that one component with the entire functionality survives. Within the biconnected component the replica is placed onto the most reliable node. If another biconnected component without the task t_i and a node with higher reliability is found the replica t'_i will be placed onto the new node. If only one biconnected component exists, the replica is placed onto the node with the highest reliability in this component.

In many cases, a network designer does not precisely know about the reliability of the applied communication or computational resources. Therefore, the following algorithms are investigated which do not require any reliability values, e.g., failure

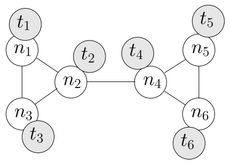


Fig. 4. Each node n_i executes a task t_j and each task t_j causes the load of $c_j = 0.5$.

rates, of the resources:
Random: The *Random* strategy places a replica t'_i onto a randomly selected node which does not execute the task t_i. This strategy is just considered for comparison of our approaches.
BCC Max1: The *BCC Max1* strategy resembles the *BCC Max1 Max Rel* strategy, but instead of selecting the node with highest reliability an arbitrary node of the biconnected component is chosen.
BCC Max2: The *BCC Max2* strategy is also similar to the *BCC Max2 Max Rel* strategy, but neglects the reliability of the nodes.
BCC Task: The *BCC Task* strategy places the replica t'_i in the biconnected component next to the biconnected component hosting the task t_i. If only one component exists the replica t'_i is placed onto an arbitrary node.

The entire replica placement approach is a greedy strategy which places a replica onto a network node if it improves the current binding. Moreover, the approach runs asynchronously and concurrently in the network.

4.3 Computational Load of Active and Passive Replicas

Typically, the computational capacity in embedded networks is restricted and it might not be possible to execute the tasks and their replicas simultaneously (active replication). Instead, it is possible to keep replicas passively in the memory and activate them if the node executing the corresponding tasks will fail. In this case, it has to be assured that the computational capacity of each node is not exceeded after the replicas are activated. Therefore, we will consider two load scenarios here: 1.) tasks are actively replicated such that each replica requires memory as well as computational resources and 2.) tasks are passively replicated such that only memory but no computational resources are required (unless a node fails and the replica is activated).

For example, consider the network and the task binding shown in Fig. 4. Each node n_i executes a task t_j with the computational load of $c_j = 0.5$, i.e., 50% of the capacity of a node are required by the task t_j. We can now illustrate the differences for active and passive replication:
If active replicas t'_j are placed onto the nodes n_i, exactly one active replica may be bound onto each node in the network. Otherwise, the computational load of the replicas and the task bound onto a node n_i will exceed the capacity of that node.
If passive replication is chosen, each node may host several replicas, but it has to be assured that only one replica will be active after a node defect. Thus, considering the results from the discussion on biconnected components the following binding possibilities for replicas exist on node n_1: $\{t'_2, t'_3\}$, $\{t'_3, t'_4\}$, $\{t'_3, t'_5\}$, $\{t'_3, t'_6\}$. Each set of the form $\{t'_r, \ldots, t'_s\}$ contains a maximal number of replicas which may but might not necessarily be bound onto a node n_1 in addition to a task t_1. For example, the node n_1 can host task t_1 and the replicas t'_2 and t'_3. Alternatively, n_1 can host t_1 and $\{t'_3, t'_4\}$, or t_1 and $\{t'_3, t'_5\}$ or t_1 and $\{t'_3, t'_6\}$. Compared to the active replication strategy, it can be clearly seen that the sets are larger. Note that n_1 cannot host t_1 and $\{t'_2, t'_4\}$ because a defect of n_2 leads to a the activation of both replicas t'_2 and t'_4. Compared to the active replication strategy, it can be clearly seen that there are more potential nodes for placing replicas.

In the following, we will formally describe the differences between active and passive replication in the context of computational load. For simplicity, we assume that each task can be executed by each node, i.e., other resource constraints than the computational power do not exist.
Definition 11 (Normalized Load Constraint). *Let the computational capacity of a node n_i be 1. Let $c_j \in \mathbb{R}^+$ be the real-valued fraction of the capacity of a task t_j*

consumed if executed by node n_i. All computational loads c_j of tasks executed by n_i must not exceed the capacity: $\sum_{t_j : t_j \mapsto n_i} c_j \leq 1$ $t_j \mapsto n_i$ means that task t_j is executed by node n_i.

Definition 12 (Active Replication Constraint). *The sum of computational loads c_s of the tasks t_s or active replicas t'_s executed by node n_i must not exceed the capacity of n_i: $\sum_{t_s : t_s \mapsto n_i} c_s + \sum_{t'_{\tilde{s}} : t'_{\tilde{s}} \mapsto n_i} c_{\tilde{s}} \leq 1$*

By Def. 5, it is trivial that Def. 12 obeys Def. 11.

For the following considerations, we additionally assume that each node n_i knows whether a task t_x is executed by a node n_j in the same biconnected component ($n_i, n_j \in BCC_r$) or in a different biconnected component ($n_i \in BCC_r, n_j \in BCC_s$ with $BCC_r \neq BCC_s$). If the nodes n_i, n_j are in different biconnected components, the unique articulation point $n_k^{AP} \in BCC_r$ required for accessing BCC_s is also known by $n_i \in BCC_r$.

Definition 13 (Passive Replication Constraint). *Let c_x be again the normalized computational load of a task t_x and its passive replica t'_x after activation. Then, for a node n_i belonging to a biconnected component BCC_r ($n_i \in BCC_r$), the following equation must hold:*

$$\max\left(LR_{n_i}^{\mathrm{BCC}}, LR_{n_i}^{\mathrm{AP}}\right) + LT_{n_i} \leq 1 \tag{1}$$

$LR_{n_i}^{\mathrm{BCC}}$ denotes the load caused by the replicas placed onto node n_i. The replicas stem from the tasks of the nodes n_k within the same biconnected component than n_i where n_k is not an articulation point ($n_i, n_k \in BCC_r, n_k \notin BCC_s$ with $BCC_r \neq BCC_s$):

$$LR_{n_i}^{\mathrm{BCC}} = \max_{\substack{n_k \in BCC_r \\ \wedge n_k \neq n_i \\ \wedge n_k \notin BCC_s}} \Bigg(\sum_{\substack{t'_x : t'_x \mapsto n_i \\ \wedge t_x \mapsto n_k}} c_x \Bigg) \tag{2}$$

$LR_{n_i}^{\mathrm{AP}}$ denotes the load caused by the replicas placed onto node n_i. This time the replicas stem from the tasks of the nodes n_k which are not in the same biconnected component than n_i ($n_i \in BCC_r, n_k \notin BCC_r$):

$$LR_{n_i}^{\mathrm{AP}} = \max_{\substack{n_j^{AP} \in BCC_r \\ \wedge n_j^{AP} \neq n_i}} \Bigg(\sum_{\substack{n_k \in BCC_s \\ \wedge n_j^{AP} \in PATH(n_i, n_k) \\ \wedge BCC_r \neq BCC_s}} \sum_{\substack{t'_x : t'_x \mapsto n_i \\ \wedge t_x \mapsto n_k}} c_x \Bigg) \tag{3}$$

$PATH(n_i, n_k)$ denotes a set of nodes connecting $n_i \in BCC_r$ and $n_k \in BCC_s$. n_j^{AP} is an articulation point as defined in Def. 9.

$LT_{n_i}^{\mathrm{AP}}$ denotes the load caused by the tasks placed onto node n_i: $LT_{n_i} = \sum_{t_x : t_x \mapsto n_i} c_x$

Theorem 1. *If passive replication is applied then the passive replication constraint satisfies the normalized load constraint.*

Proof. The overall computational load of a node n_i is composed of three different loads: 1.) the load of replicas $t'_x \mapsto n_i \in BCC_r$ with $t_x \mapsto n_j \in BCC_r$ and $n_j \notin BCC_s$, 2. the load of replicas $t'_x \mapsto n_i \in BCC_r$ with $t_x \mapsto n_j \in BCC_s$ ($BCC_r \neq BCC_s$), and 3.) the load of tasks $t_x \mapsto n_i$.

1. Given are two different nodes n_k and n_i belonging to the same biconnected component $n_i, n_k \in BCC_r$ and n_k belongs to no other biconnected component BCC_s, i.e., n_k is not an articulation point. The tasks t_x of node n_k have replicas t'_x at node n_i which do not cause any computational load unless they are activated (see Def. 6). After a single defect of n_k, the other nodes n_j belonging to the same biconnected component $n_j \in BCC_r$ are still connected to n_i (see Def. 8). Thus, only the replicas t'_x of the tasks t_x of node n_k will be activated. Since each node n_k might fail and may have replicas t'_x of its task at n_i, the maximum computational load that will be added to the load of n_i after a defect of a node n_k has to be considered. This is $LR_{n_i}^{\mathrm{BCC}}$.

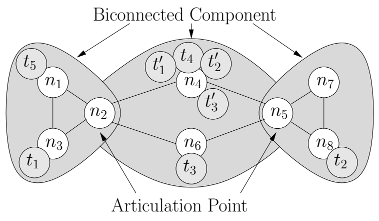


Fig. 5. The figure acts as an example for Eq. (3). From the point of view of node n_4, a single node defect might lead to failure of the functionality hosted in either the right biconnected component, the left biconnected component, or node n_6 but not all together.

2. Given are two different nodes n_k and n_i belonging to different biconnected component $n_i \in BCC_r$, $n_k \in BCC_s$ and $BCC_r \neq BCC_s$. BCC_r has one or several articulation points $n_j^{AP} \in BCC_r$. Since different biconnected components are connected via one unique articulation point n_j^{AP}, a single resource defect leads to a situation where all tasks t_x on n_j^{AP} and behind this articulation point are inaccessible. Thus, a failure of $n_j^{AP} \in BCC_r$ leads to a failure of all biconnected components BCC_s which are accessible via $n_j^{AP} \in BCC_r$. Since one resource may fail simultaneously, the network region accessible via another $n_l^{AP} \in BCC_r$ is still accessible in case of the a failure of $n_j^{AP} \in BCC_r$. Thus, network regions accessible via different articulation points $n_l^{AP} \in BCC_r$ fail exclusively from the point of view of a node $n_i \in BCC_r$. The maximal computational load at node n_i coming from a network region if $n_j^{AP} \in BCC_r$ fails is $LR_{n_i}^{\mathrm{AP}}$.

The load caused at a node $n_i \in BCC_r$ due to a defect of $n_k \in BCC_r$ is either $LR_{n_i}^{\mathrm{BCC}}$ or $LR_{n_i}^{\mathrm{AP}}$. This depends on whether n_k is an articulation point or not. Thus, the maximum of $LR_{n_i}^{\mathrm{BCC}}$ and $LR_{n_i}^{\mathrm{AP}}$ will be added to the load of n_i.

3. Since all tasks t_x executed by n_i cause a computational load, the overall load caused by the tasks is the sum of the corresponding c_x.

As an example for the case covered by Eq. (1), consider the network in Fig. 5. The network consists of eight nodes partitioned into three biconnected components which are connected by two articulation points (n_2^{AP}, n_5^{AP}). The task binding is given and some of the replicas are already placed in the network Assume that node n_4 is asked whether it can host a passive replica of task t_5 by the algorithm shown in Alg. 1. The node n_4 need to know that they access nodes outside their own biconnected component via n_2^{AP} or n_5^{AP}. In detail, the nodes n_1 and n_3 can be accessed via a path containing articulation point n_2^{AP} whereas the nodes n_7 and n_8 can be accessed via a path containing articulation point n_5^{AP}. Thus, a defect of on single resource , e.g., n_2^{AP} or n_5^{AP}, can split the network such that either the left or the right biconnected component is not accessible from n_4. But, it is not possible that the left and right biconnected component will become unaccessible from n_4 at the same time. If node n_6 fails no task hosted by other nodes will be inaccessible. Thus, node n_4 has to calculate:

$$LR_{n_4}^{\mathrm{BCC}} = \max(c_3), \quad LR_{n_4}^{\mathrm{AP}} = \max(c_5 + c_1, c_2), \quad LT_{n_4} = c_4 \tag{4}$$

If the following inequality holds true the replica of t_5 can be placed:

$$\max\left(LR_{n_4}^{\mathrm{BCC}}, LR_{n_4}^{\mathrm{AP}}\right) + LT_{n_4} \leq 1 \tag{5}$$

5 Experimental Evaluation

With the help of extensive Monte-Carlo simulations, the different strategies have been evaluated. For this purpose, we implemented a behavioral model of the network and each node as well as each link in the network has been annotated with a constant failure rate λ. Afterwards, the simulator randomly determines the defect time of each node and link using the inverse function of the reliability function $R(t) = e^{-\lambda t}$ [14]: $t = \frac{\ln R(t)}{-\lambda}$ Note that the random values for $R(t)$ are uniformly distributed. The resources are ordered according to their defect time t. Starting with the smallest defect time, the simulator removes at each defect time the corresponding resource from the network topology. After each removal of a resource, the online methodology (see Fig. 1) is processed and the simulator checks if the entire functionality is still operable. If this is not the case, the network has failed to operate. The defect time t of the last removed resource which led to a network failure is called the *time to failure* TTF. In order to determine the *mean time to failure* $MTTF$, the simulation is repeatedly executed and the average of all obtained TTF values is calculated.

For the experimental evaluation of our methodology, we generated different network system models and varied 1.) the number of mapping edges for each task and 2.) the failure rate of the links ($\lambda - Link$) and the failure rate of the nodes ($\lambda - Node$).

In the following figures, a relation of the failure rates is presented. A relation of the failure rates $\lambda - Node/\lambda - Link = 0.1$ denotes a case where $\lambda - Node$ varies between 0.0001 and 0.0009 and $\lambda - Link$ ranges from 0.001 to 0.009. The number mapping edges is normalized with the number of tasks, i.e., if $\#MappingEdges/Task = 0.3$, each task can be executed by 30% of the network nodes. If $\#MappingEdges/Task = 1$ each task can be executed everywhere in the network unless the nodes do not have enough computational resources. The computational power of all network node is normalized to one and the load of a task varies between 0.1 and 0.5.

At first, we consider the placement strategies where active replicas are placed in the network. We normalized the MTTF values obtained with the proposed strategies with the MTTF values of the reference strategies *Random* or $MaxRel$, respectively. Fig. 6a shows the normalized MTTF for active replication over the number of mapping edges per task where each node knows its failure rate. The same case is presented in Fig. 6b) but here, the nodes do not know about their failure rate.

Fig. 6c),d) present the same results than Fig. 6a),b), but instead of placing active replicas, passive replicas are placed within the network.

As mentioned above, we also varied the failure rate of nodes and links. Therefore, the same results as in Fig. 6 and Fig. 6 are presented over the failure rate relation $\lambda - Node/\lambda - Link$ in Fig. 7. Fig. 7a),b) show the results for active replication while Fig. 7c),d) show results for passive replication.

Our test-cases show that the *BCC Task (Max Rel)* strategy outperforms the other distributed approaches to replica placement. Thus, a diverse placement of tasks and replicas in the network is better than cumulating tasks in certain biconnected components in the network. Comparing Fig. 6c) with Fig. 6c) and Fig. **??**d) with Fig. 6d), respectively shows the impact of passive replication to tolerate permanent resource defects. In particular, passive replication together with the *BCC Task (Max Rel)* strategy leads to an improvement of the MTTF which is up to 10% higher than active replication. Of course, active replication has advantages concerning transient faults and fault reaction times, but due to the limited computational power of network nodes, passive replication is able to improve the availability of the entire network functionality.

Fig. 7 shows that the benefit of our topology-aware replica placement strategies decreases if network nodes are much more likely to fail than links between these nodes. This behavior is due to the fact that the network decomposes very fast, but no connected component is able to host all communicating tasks and the network functionality cannot be provided any more. In this case, the potential for increasing the MTTF compared to strategies without topology information is low. On the other hand, if the probability of a link defect is higher than the probability of a node defect, the network topology might become like a chain of biconnected node groups. In such a network topology, our strategies are aware of the node groups and are able to place tasks such that the MTTF is improved. Note that this latter case where the failure rate of a link is higher than the failure rate of a node is more realistic. Hence, our approach leads to a better improvement of the normalized MTTF for more realistic cases.

6 Conclusions

In this paper, we presented novel heuristics for placing replicated tasks in embedded networks. With the goal to increase the mean time to failure and hence the reliability

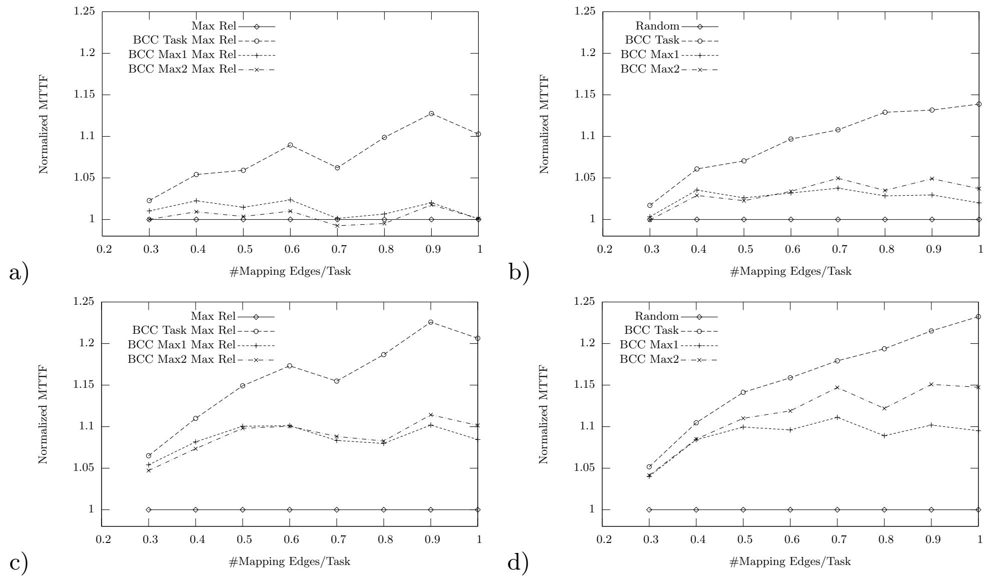
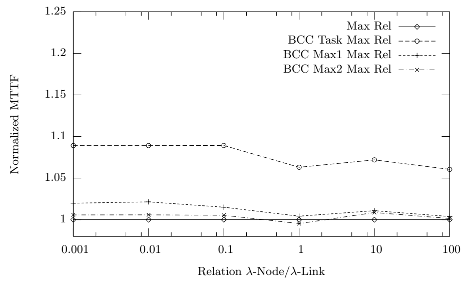


Fig. 6. Depending on the number of mapping edges per task, the reliability improvements a) compared to the reliability of the Max Rel strategy and b) compared to the Random strategy for active replication as well as c) compared to the reliability of the Max Rel strategy and d) compared to the Random strategy for passive replication are presented.

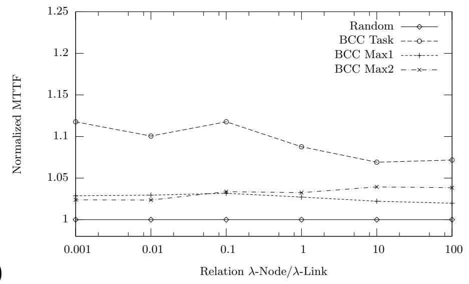
of the network and to delay the failure of the network's functionality, different heuristics are proposed which are based on the partitioning of the network into biconnected components. Moreover, we discussed differences of active and passive replication and experimentally illustrated the impact of the passive replication on the mean time to failure. Note that differences due to active and passive replication can only be considered if our heuristics are applied. Other randomized strategies ignoring topology information might not be able to respect differences in active and passive replication. All in all, replica placement strategies based on biconnected components seem to be quite attractive, 1.) because they run in a distributed manner, 2.) they improve the mean time to failure, and 3.) they allow for distinguishing between active and passive replication.

References

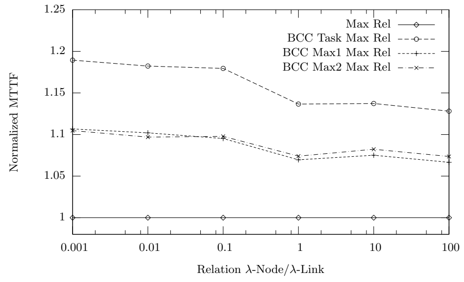
1. A. Adya, W. Bolosky, and M. Castro. FARSITE: Federate, Available, and Reliable Storage for an Inclomplete Trusted Environment. In *Proceedings of the OSDI*, 2002.
2. A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
3. N. Budhiraja, K. Marzullo, F. B. Schneider, and S. Toueg. *The Primary-Backup Approach*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1993.
4. F. M. Cuenca-Acuna, R. P. Martin, and T. D. Nguyen. Autonomous Replication for High Availability in Unstructured P2P Systems. In *22nd IEEE International Symposium on Reliable Distributed Systems*, 2003.
5. J. R. Douceur and R. Wattenhofer. Competitive Hill-Climbing Strategies for Replica Placement in a Distributed File System. In *DISC '01: Proc. of Int. Conference on Distributed Computing*, pages 48–62, London, UK, 2001. Springer-Verlag.
6. J. R. Douceur and R. Wattenhofer. Modeling Replica Placement in a Distributed File System: Narrowing the Gap between Analysis and Simulation. In *ESA '01: Proc. of European Symposium on Algorithms*, pages 356–367, London, UK, 2001.
7. J. R. Douceur and R. P. Wattenhofer. Optimizing the File Availability in a Server-less Distributed File System. In *Proceedings of the Symposium on Reliable Distributed Systems*, pages 4–13, 2001.
8. W. Hohberg. How to find biconnected components in distributed networks. *J. Parallel Distrib. Comput.*, 9(4):374–386, 1990.



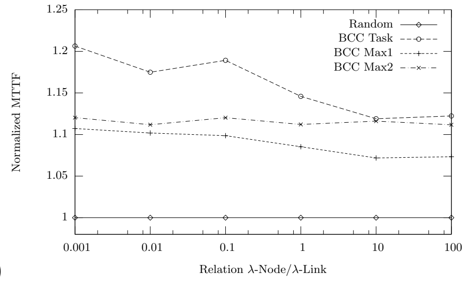
a)



b)



c)



d)

Fig. 7. Depending on the failure rate relation $\lambda - Node/\lambda - Link$, the reliability improvements a) compared to the reliability of the Max Rel strategy and b) compared to the Random strategy for active replication as well as c) compared to the reliability of the Max Rel strategy and d) compared to the Random strategy for passive replication are presented.

9. J. Hopcroft and R. Tarjan. Algorithm 447: efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, 1973.
10. V. Izosimov, P. Pop, P. Eles, and Z. Peng. Design Optimization of Time- and Cost-Constrained Fault-Tolerant Distributed Embedded Systems. In *Proceedings of Design, Automation and Test in Europe*, Munich, Germany, Mar. 2005.
11. P. Jalote. *Fault Tolerance in Distributed Computing.* Prentice Hall, 2002.
12. M. Karlsson, C. Karamanolis, and M. Mahalingam. A Framework for Evaluating Replica Placement Algorithms. Technical report, HP Labs, HPL-2002-219, 2002.
13. D. Koch, T. Streichert, S. Dittrich, C. Strengert, C. Haubelt, and J. Teich. An Operating System Infrastructure for Fault-Tolerant Reconfigurable Networks. In *Proceedings of the 19th International Conference on Architecture of Computing Systems (ARCS 2006), Frankfurt / Main, Germany*, pages 202–216, Frankfurt, Germany, Mar. 2006.
14. J. C. Laprie. *Dependability: Basic Concepts and Terminology - In English, French, German, and Japanese.* Springer-Verlag, 1992.
15. Q. Lian, W. Chen, and Z. Zhang. On the Impact of Replica Placement to the Reliability of Distributed Brick Storage Systems. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pages 187–196, Washington, DC, USA, 2005. IEEE Computer Society.
16. L. Qiu, V. N. Padmanabhan, and G. M. Voelker. On the Placement of Web Server Replicas. In *Proc. of the IEEE INFOCOM conference*, pages 1587–1596, Apr. 2001.
17. F. Schneider. Implementing Fault-tolerant Services using the State Machine Approach: A Tutorial. *ACM Computing Surveys*, 7(5):37–53, Jan. 1990.
18. T. Streichert, C. Haubelt, and J. Teich. Distributed HW/SW-Partitioning for Embedded Reconfigurable Systems. In *Proceedings of Design, Automation and Test in Europe*, pages 894–895, Munich, Germany, Mar. 2005.
19. T. Streichert, C. Haubelt, and J. Teich. Online Hardware/Software Partitioning in Networked Embedded Systems. In *Proceedings of Asia and South Pacific Design, Automation and Test Conference*, pages 982–985, Shanghai, China, Jan. 2005.
20. T. Streichert, C. Strengert, C. Haubelt, and J. Teich. Dynamic Task Binding for Hardware/Software Reconfigurable Networks. In *SBCCI'06: Proceedings of the 19th annual symposium on Integrated circuits and systems design*, pages 38–43, Aug. 2006.
21. B. Swaminathan and K. Goldman. An Incremental Distributed Algorithm for Computing Biconnected Components in Dynamic Graphs. *Algorith.*, 22(3):305–329, 1998.
22. M. Szymaniak, G. Pierre, and M. van Steen. Latency-Driven Replica Placement. In *Proc. of the Symp. on Applications and the Internet (SAINT'05)*, pages 399–405, 2005.
23. J. Westbrok and R. E. Tarjan. Maintaining Bridge-Connected and Biconnected Components On-Line. *Algorithmica*, 7(1):433–464, Dec. 1992.