AMF Configurations: Checking for Service Protection Using Heuristics

P. Salehi, F. Khendek, A. Hamou-Lhadj
Electrical and Computer Engineering Department
Concordia University
Montréal, Canada
{pe_saleh, khendek, abdelw}@ece.concordia.ca

M. Toeroe
Ericsson Inc.
Montréal, Canada
Maria.Toeroe@ericsson.com

***Abstract*— AMF (Availability Management Framework) is a middleware service that manages the availability of applications. AMF has been defined by the Service Availability Forum (SA Forum). An AMF configuration for an application running on top of AMF is a logical organization of hardware and software resources to provide and protect services. Resources, namely components, are grouped into logical entities such as service units and service groups and are set together at configuration time to provide and protect services represented as component service instances and service instances. The assignment of component service instances and service instances to components and service units, respectively, is performed at runtime by the AMF middleware. However, the configuration is valid if and only if it satisfies all AMF constraints, including the provisioning and protection of the services. Therefore, the problem is how to ensure at configuration time that the services will be protected at runtime. In a previous work, we tackled this problem and proved it to be NP-hard in general for most redundancy models. Here, we tackle the problem of AMF configuration validation further with heuristics using extended versions of heuristics developed for the bin-packing problem. We consider all the redundancy models for which the problem is NP-hard. In addition, we propose an approach which incrementally adds resources to a "likely" invalid configuration and transforms it into a valid one.**

Keywords- High Availability; Availability Management Framework; System Configurations; Service Protection; Complexity; Heuristics.

I. INTRODUCTION

High availability of a system is achieved when its services are accessible (or available) to the users 99.999% of the time [1]. The demand for Highly Available (HA) services is continuously growing in different domains including telecommunications, banking and air traffic monitoring. The Service Availability Forum (SA Forum) [2] is a consortium of telecommunications and computing companies that has defined several standard services/interfaces to support the development of HA systems. Such standards aim at reducing the application development time and cost by shifting the availability management from applications to a dedicated middleware, and hence enabling portability. Among the SA Forum standards, the Application Interface Specification (AIS) [3] supports the development of HA applications by abstracting hardware and software resources. AIS defines several services that provide common functionality needed in HA applications, among which the most important one is the Availability Management Framework (AMF) [4]. AMF manages the high availability of applications by coordinating their redundant entities. In order to provide and protect the services, AMF requires a configuration that specifies the characteristics of the entities composing the system, their types and the way they are organized. Entities, such as components, Service Units (SU) and Service Groups (SG), describe service providers; other entities, like Component Service Instances (CSI) and Service Instances (SI), describe the provided services. AMF defines five redundancy models [4], namely the No-redundancy, the 2N, the N+M, the N-Way-Active and the N-Way redundancy models, which vary depending on the number of SUs that can take active and/or standby assignments and how these assignments are distributed among the SUs.

It is necessary to provide the middleware with configurations which are not only syntactically correct and well-formed with respect to the AMF specification, but also semantically correct and capable of ensuring the required level of availability of the configured services. In our previous work [5], we explored and discussed this issue, referred to as the SI-Protection problem. We identified two redundancy models, namely 2N and No Redundancy, where the problem can be solved efficiently. We proved that in the case of N+M, N-Way and N-Way-Active redundancy models the problem is NP-hard in general. For these three redundancy models, we identified some specific situations where the problem can be simplified. In this paper, we tackle the problem further and propose a solution for the N+M, N-Way and N-Away-Active redundancy models that is based on heuristics. Our solution is based on extensions to the well-known problem of bin-packing [6]. We replace bins and objects with SUs and SIs, respectively. We consider different types of capacity, i.e. capacity vector, unlike the single type of capacity in the classical bin-packing problem.

Our ultimate goal is to design valid AMF configurations to provide and protect services. When a configuration is "likely" to be invalid because we cannot demonstrate its validity or invalidity with our heuristics based approach, we propose to transform it into a valid configuration by the incremental addition of resources. Despite the fact that this may result in over dimensioning the system, the resulting configuration is a valid one. We discuss this method in this paper as well. In

Section II, we introduce the necessary background on AMF configurations along with the review of related work. In Section III, we define the SI-Protection problem and introduce notations used throughout the paper. In Section IV, we propose our generic approach, i.e. independently from redundancy models, for solving the SI-Protection problem using heuristics. In Section V, we map our approach to each of the redundancy models: N+M, N-Way-Active, and N-Way. We present our approach for the incremental modification of AMF configurations in Section VI and we draw conclusions in Section VII.

II. BACKGROUND

A. AMF Configurations

An AMF configuration [4] for a given application is a logical organization of resources for providing and protecting services. It consists of components grouped into SUs, which are grouped into SGs.

TABLE I. DESCRIPTION OF AMF ENTITIES AND ACRONYMS

Entity	Description
Component	Represents a hardware or software resource that can provide a service.
Component Service Instance (CSI)	Represents a service workload assigned to a component. AMF assigns workload to a component at run-time.
Service Unit (SU)	Groups a set of components which collaborate to provide services.
Service Instance (SI)	Represents a service an SU can provide. It is an aggregation of CSIs.
Service Group (SG)	It consists of a set of SUs. It protects according to redundancy model the SIs assigned to these SUs.
Application	Represents a logical entity that contains one or more SGs and combines the individual functionalities of the constituent SGs to provide a higher-level service.
Node	Represents a computational resource for the deployment of artefacts.
Cluster	Represents the aggregation of the complete set of AMF nodes in an AMF configuration.

An application consists of one or several SGs that provide and protect services defined in terms of SIs composed of CSIs. At runtime, for each SI configured to be protected by a given SG, AMF assigns the active and standby states to the SUs of that SG, according to the redundancy models. As mentioned previously, the AMF specification defines five redundancy models [4], namely the No-redundancy, the 2N, the N+M, the N-Way-Active and the N-Way redundancy models. These redundancy models vary depending on the number of SUs that can be active and standby for the SIs and how these assignments are distributed among the SUs. In the 2N redundancy model one SU is active for all the SIs protected by the SG and one is standby for all the SIs. In the N+M model, N SUs handle the active assignments and M SUs handle the standby assignments. N+M allows at the most one active and one standby assignment for each SI. An SG with N-Way redundancy model contains N SUs. Each SU can have a combination of active and standby assignments. However, each SI can be assigned active to only one service unit, whereas it can be assigned standby to several service units. An SG with N-Way-Active redundancy model has N SUs, which are only assigned as active. It has no SU assigned as standby. Furthermore, each of the SIs protected by this SG can be assigned to more than one SU. Figure 1 summarizes the different redundancy models defined in the AMF specification.

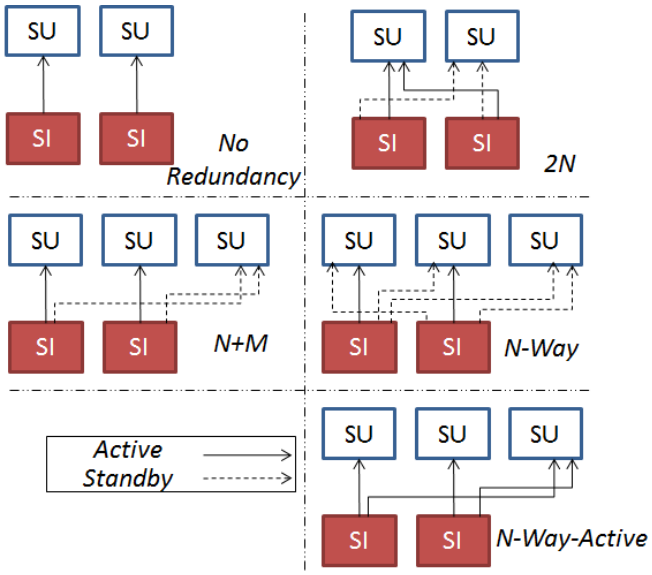

Figure 1. Redundancy models defined in the AMF specification

An AMF application may consist of different SGs with different redundancy models. Each SU is deployed on an AMF node and the set of all AMF nodes forms the AMF cluster. Table I summarizes the AMF entities and their descriptions. AMF entity types define the common characteristics among multiple instances of the previously defined logical entities. In AMF, all entities except the deployment entities (i.e. nodes and clusters) have a type. For instance, Component Service Type (CSType) is an entity type describing the common characteristics of its CSIs. Figure 2 shows an example of AMF configuration. In this example a cluster is composed of two nodes (Node1 and Node2). It hosts an application consisting of one SG protecting two SIs in a 2N redundancy model. The SG consists of two SUs, SU1 and SU2, each composed of two components. The distribution of the active and standby assignments is shown in the figure. However, it is not part of the configuration as defined by AMF, as this is decided by AMF at runtime. For the sake of simplicity, the entity types are not presented in this figure.

B. Bin-packing Problem

In the bin-packing problem [6], objects of different volumes must be packed into a finite number of bins of capacity *n* in a way that minimizes the number of bins used. The bin-packing problem has many applications, such as bandwidth allocation and creating file backup in removable media. Despite the fact that the problem is NP-hard, optimal solutions can be produced with sophisticated algorithms. In addition, many heuristics have been developed. For example, the first fit algorithm provides a fast but often non-optimal solution, which involves placing each item into the first bin in which it fits [8]. The algorithm can be made much more effective by first sorting the list of objects into decreasing order [9]. This, however, still

does not guarantee an optimal solution, and for longer lists it may increase the running time of the algorithm [10].

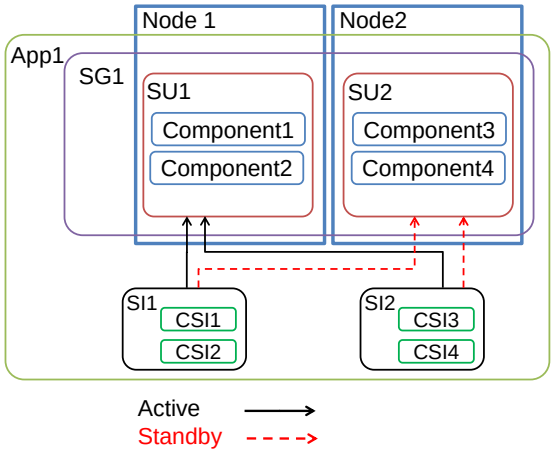

Figure 2. An example of AMF configuration

C. Related Work

An AMF middleware uses configurations to manage the resources under its control. AMF configurations need to be validated before they can be used by the AMF middleware. One important criterion in the validation of an AMF configuration is whether it provides the level of protection it is designed for according to the specified redundancy model. Indeed, a configuration is AMF compliant if and only if it meets all the AMF requirements including the protection of services as required. Validating the protection of services as requested may require the exploration of all possible SI combinations and SI-SU assignments. This is a combinatorial and complex problem. To the best of our knowledge, except for the traditional work on complexity [11] and our previous work [5], we are not aware of any related work dealing with SI protection in the context of AMF.

In our previous paper [5] we discussed the complexity of the SI-Protection decision problem for an AMF configuration. In the case of the 2N redundancy model and the No-redundancy model, we have identified necessary and sufficient conditions that can be checked in polynomial order. We also showed that the problem is NP-hard for the N+M, the N-Way-Active, and the N-Way redundancy models. To overcome this complexity, due mainly to the consideration of all possible combinations of SI-to-SU assignments, we have characterized special cases for special sets of SIs, where necessary and sufficient conditions have been defined and can be checked with simple algorithms.

The bin-packing problem has already been revisited and extended to vector bin-packing, see for instance [7, 12, 13]. Vector bin-packing is a variation of classical bin-packing in which the capacity of bins and objects is described in terms of a vector of capacities [7]. Several approximation algorithms have been proposed to optimize the number of bins. Recently, Patt-Shamir and Rawitz explored the vector bin-packing problem with bins of variable sizes and presented an approximation algorithm [12]. In [13] Rao et al. developed an approximation algorithm based on the near-optimal solution of linear programming relaxation of integer programming. These approximation algorithms introduce a boundary guaranteeing that their sub-optimal result will not exceed this boundary. This boundary is expressed as a factor of the optimal solution and the parameters (number of objects and the size of the vector) of the problem. Furthermore, the amount of computational and memory resources necessary for solving the problem will increase exponentially when the boundary becomes close to the optimal solution. For this reason, the efficiency of these approximation algorithms will rarely prove to be practical for large systems such as AMF configurations. Heuristics, however, target reasonably good solutions efficiently [14]. Moreover, the main concern in abovementioned papers is the approximation of the optimal number of bins, while in our case we are interested in finding a possible assignment of a given set of SIs to a given set of SUs. Therefore, based on the traditional bin-packing problem heuristics, we devised new heuristics for solving the SI-Protection problem taking into account the specificities of the domain in question, i.e. SUs, SIs, and redundancy models.

III. The SI Protection Problem

An AMF configuration can be seen as a set of n of SUs denoted by $SUList = \{SU_1, \dots SU_n\}$. Each SU combines a group of components capable of supporting different CSTypes (i.e. capable of providing the CSIs of those CSTypes). Let k denote the total number of CSTypes supported by the SUs of the *SUList*. Consequently, the provided capacity list for SU_i is defined as $\langle c_1^i, \dots, c_k^i \rangle$ in which $c_t^i \geq 0$ is an integer representing the capacity of the SU in supporting CSIs from the CSType t.

The n SUs in the *SUList* need to protect a given sequence of m SIs, denoted by $SIList = \{SI_1, \dots SI_m\}$. Similar to the provided capacity list of SUs, for each $SI_j \epsilon SIList$, the required capacity list can be defined by an ordered set $\langle r_1^j, \dots, r_k^j \rangle$ determining the required capacity of the SI_j for each CSType.

We can assign an SI (SI_j) to an SU (SU_i) if and only if the remaining capacity of SU_i for all CSTypes is not less than the capacity required by SI_j. It is important to note that SIs are units of assignment and are indivisible.

The problem of SI-protection can be defined as follows: is it possible to assign, according to the redundancy model, all the m SIs to the given set of n SUs?

As an illustration of the SI-Protection problem, let us consider the partial configuration in Figure 3, which displays only the active part of an SG that consists of three SUs and which is configured to protect two SIs with the N+M redundancy model. Note that in this example we only consider the capacity of the components for handling the active workload, the $SUList = \{SU_1, SU_2, SU_3\}$ with the provided capacity list of $\{\langle 4,2,1 \rangle, \langle 3,3,1 \rangle, \langle 3,4,1 \rangle\}$ and the $SIList = \{SI_1, SI_2\}$ with the required capacity list equal to $\{\langle 1,2,0 \rangle, \langle 3,2,1 \rangle\}$. Can the list of SUs with the provided capacities protect the list of SIs with the given required capacities?

IV. CHECKING FOR SI-PROTECTION USING HEURISTICS

In this section, we present a solution for checking the SI-Protection for the general case of the N+M, N-Way and N-Way-Active redundancy models using heuristics. The proposed approach is based on heuristics developed initially for the bin-packing problem [6].

In the case of the SI-Protection problem, the SUs can be considered as bins and the SIs, as the objects. The capacities of SUs/SIs are described in terms of the number of the CSIs of a certain CSType which can be provided by SUs and form the SIs.

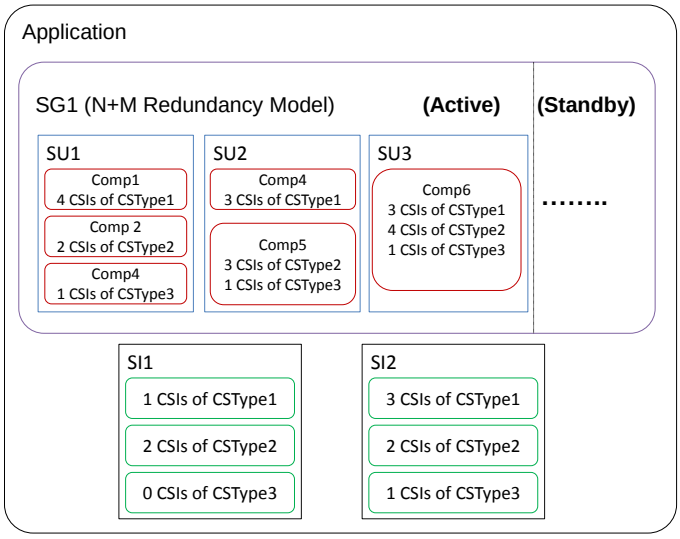

Figure 3. The active part of an example of SG with N+M redundancy model

We extend the three well-known heuristics for bin-packing. Each of these extensions takes the *SUList* and *SIList* as input and decides if there exists a way to assign all SIs of the *SIList* to the SUs of the *SUList*. If an algorithm succeeds in assigning all SIs to the SUs, the answer to the problem is 'Yes'. If it fails, the answer could be 'Yes' or 'No'. Since all these algorithms take a sequence of SIs and assign them one by one, an algorithm will answer 'No' if it fails to assign an SI at a certain point. This may be a False negative. When all SIs are successfully assigned to the SUs, the algorithm returns 'Yes' as result. Therefore, the signature of each algorithm can be represented as:

$$Boolean\ bin\_packing\_extension_X(SUList, SIList)$$

It is worth noting that these extensions are generic algorithms for deciding about the SI-Protection and do not consider any specific redundancy model. In Section V, we discuss the application of these algorithms to each of the redundancy models.

To achieve better results, our approach applies all proposed algorithms to the sequence and then determines the logical OR of the answers. Since these algorithms are different (and somehow based on opposite principles), the probability of a False negative result is reduced.

A. *First-Fit approach (FF)*

The first approach is the First-Fit (FF) approach, where we preserve a fixed order of SUs in the *SUList* during the whole processing. To assign a given SI to an SU, we simply take the first available SU in the *SUList* which can serve the SI.

Although the FF approach appears to be the easiest heuristic to the problem, it is known to be quite effective for $k = 1$ (classical bin-packing).

Complexity: The assignment of each SI to each SU can be achieved with k comparisons between the provided and required capacities of the SU and the SI. Moreover, the number of SUs that need to be checked before finding the appropriate one can reach n, at the most. Considering the number of assignments which equals the number of SIs (m), the complexity of this approach is $m \times n \times k$ in the worst case.

B. *Best-Fit approach (BF)*

This approach gives the best results in practice for the classical bin-packing problem [9].We keep the SUs sorted in an increasing order of remaining capacities, and find the first SU capable of handling the load of the SI. Therefore, a given SI is assigned to an SU which has the minimum remaining capacity among those which have enough capacity for the SI under consideration. Note that the list of SUs should be sorted after each assignment. Here, the goal is to exhaust an SU as much as possible before moving to the next. BF is occasionally referred to as *unbalanced assignment approach* [9]. Since there is no single value defined as the 'capacity' of each SU, the provided capacity being represented through a list of non-negative integers, it is necessary to come up with a single criterion for the capacity of each SU, and to sort the SUs in the *SUList* based on this criterion. In what follows, we introduce three different criteria to represent the capacity of a given SU.

Total Capacity: Given the remaining capacity list ($\langle c_1^i, \dots, c_k^i \rangle$) for a given SU ($SU_i$), the total capacity is the sum of the remaining capacities of all supported CSTypes in SU_i ($capacity\ of\ SU_i = \sum_{t=1}^{k} c_t^i$).

For instance, let us consider the example of Figure 3 where we have three SUs (SU_1, SU_2, SU_3) supporting three different CSTypes and let the remaining capacity list for these SUs be $\langle 4,2,1 \rangle, \langle 1,1,1 \rangle$, and $\langle 2,4,0 \rangle$, respectively. The total capacities for the SUs are 7, 3, and 6, resulting in the sorted list $\{SU_2, SU_3, SU_1\}$. On the service side, there are two unassigned SIs (SI_1, SI_2) with the required capacity list of $\langle 1,2,0 \rangle$ and $\langle 3,2,1 \rangle$, respectively. For assigning SI_1, SU_2 will be considered first, then SU_3 and finally SU_1. SU_1 does not have the required capacity of each CSType, however SU_3 does in fact have this capacity.

Complexity: Sorting the *SUList* can be achieved in $O(n \log n)$ and keeping it sorted is $O(\log n)$. For each SI, we need to examine at the most all the n SUs in the *SUList* in order to find the proper SU. This can be done in $n \times k$ comparisons. In addition, after the successful assignment of an SI, we need to keep the *SUList* sorted. As a result, the complexity of this approach is $m \times \big(n \times k + O(\log n)\big) + O(n \log n)$.

As a variation for this case, one may also consider the sorting of the SIs at the beginning of the process, according to the total required capacity and processing the SI with the smallest capacity first or last. However, sorting SUs or SIs according to total provided or required capacity, respectively, does not necessarily help as it does not look into CSType capacities which are important for the assignments.

Relative Capacity: Contrary to the total capacity criterion, the relative capacity is defined with respect to a specific SI and is based on the largest element of the required capacity list of the SI. As a result, for each SI, the sorted list of SUs may differ. For a given SI (SI_j) with the capacity list of $\langle r_1^j, \dots, r_k^j \rangle$, let the index of the largest member of the required capacity list be $t = argmax(\langle r_1^j, \dots, r_k^j \rangle)$. This means that, for SI_j, the number of CSIs of CSType_t is larger than the number of CSIs of the other CSTypes. Consequently, for SI_j we need to sort the *SUList* based on the c_t of each SU (e.g. c_t^i for SU_i).

Let us consider again the example in Figure 3. The largest required capacities of SI_1 and SI_2 are 2 and 3, respectively. Therefore, the relative capacity criterion for SI_1 is c_2 , which results in the sorted *SUList*, $\{SU_2, SU_1, SU_3\}$. Similarly, c_1 is the criterion for SI_2 and the sorted *SUList* is $\{SU_2, SU_3, SU_1\}$.

Complexity: The complexity of the approach is very similar to the case of total capacity. The only difference is that the sorted list of SUs is different for each SI and thus, we need to sort the *SUList* for each SI separately. Consequently, the complexity of this approach is $m \times \big(n \times k + O(n \log n)\big)$.

Critical Capacity: Similar to the relative capacity, this criterion is also defined with respect to each SI. Here our objective is to find the most critical CSType for each SI and then sort the list of SUs based on this criterion. The most critical CSType for each SI is the CSType which has the largest required capacity in the SI while having the smallest provided capacity among the SUs in the *SUList*. To this end, we first determine the total capacity per CSType of the SUs as $\langle tc_1, \dots, tc_k \rangle = \sum_{i=1}^{n} \langle c_1^i, \dots, c_k^i \rangle = \langle \sum_{i=1}^{n} c_1^i, \dots, \sum_{i=1}^{n} c_k^i \rangle$. Thereafter, for a given SI_j with the required capacity list of $\langle r_1^j, \dots, r_k^j \rangle$, the index of the most critical required capacity is:

$$T = \arg\max \left({r_1^j}/{tc_1}, \dots, {r_k^j}/{tc_k} \right)$$

Consequently, for SI_j we need to sort the *SUList* based on the c_T of each SU (e.g. c_T^i for SU_i).

Going back to the example in Figure 3, the total capacity per CSType of the *SUList* is $\langle 7,7,2 \rangle$. For SI_1 based on the calculation $(\langle \frac{1}{7}, \frac{2}{7}, \frac{0}{2} \rangle)$, the critical required service is r_2 and hence, the *SUList* should be sorted according to c_2, which results in the sorted list $\{SU_2, SU_1, SU_3\}$. With the same calculation, the *SUList* for SI_2 is sorted based on c_3 and results in $\{SU_3, SU_2, SU_1\}$.

Complexity: The complexity of the critical capacity is the same as for relative capacity, i.e. $m \times \big(n \times k + O(n \log n)\big)$.

C. Worst-Fit approach (WF)

While this algorithm is not preferred in practice to the BF approach, it is important as it uses a contrary approach, and occasionally gives positive answers when BF fails. The algorithm is more or less the same as for the BF approach the only difference being that the *SUList* is sorted in a decreasing order of capacities. In fact, the algorithm attempts to assign SIs to the SUs in a balanced way. To sort the *SUList*, we can use the exact same sorting criteria as described for the BF approach in Section IV.B.

V. Taking Into Account the Redundancy Models

In the previous section, we introduced three different approaches for checking the protection of the SIs. In addition, we have also defined three different criteria for sorting the list of SUs that can be used for both the BF and the WF approaches. Therefore, we presented seven different heuristic methods for solving the SI-Protection problem that can be applied in sequence to improve the accuracy of the solution. However, all presented approaches target the generic case of the SI-Protection without taking into account the features and the specific constraints of the redundancy models. In this section, we discuss how we map these general approaches for the different redundancy models, N+M, N-Way, and N-Way-Active.

A. The N+M Redundancy Model

In the N+M redundancy model, N SUs support the active assignments and M SUs support the standbys. This model allows at the most one active and one standby assignment for each SI. Assuming that the standby SUs are distinguished from active SUs, we apply our approach, the sequence of seven heuristic methods defined previously, for the N SUs configured to support the active assignment, considering their active capacity. Thereafter, we apply the approach for M SUs configured to support the standby assignment, considering their standby capacity. We are certain that the SG can protect the SIs if and only if the result of the method is 'Yes' for both N active SUs and M standby SUs. Please note that if a "No" answer results for either case, this may be a False negative.

B. The N-Way-Active Redundancy Model

An SG with the N-Way-Active redundancy model has N SUs which are assigned only as active and has no SU assigned as standby. Furthermore, each of the SIs protected by this SG can be assigned to more than one SU as specified in the *PreferredActiveAssignments* configuration attribute. In Section IV we discussed one assignment per SI only. In order to handle multiple assignments, whenever we consider an SI, we assign it to *PreferredActiveAssignments* different SUs before proceeding to the next SI. Every assignment is handled according to the methods in Section IV.

C. N-Way Redundancy Model

An SG with the N-Way redundancy model contains N SUs. Each SU can have a combination of active and standby assignments. However, each SI can be assigned active to only one SU while it can be assigned standby to several SUs (as specified in the *PreferredStandbyAssignments* attribute). The solution for this redundancy model is quite similar to the one

for N-Way-Active. For the single active assignment in N-Way redundancy model, we consider the active capacity of the SUs while, for multiple standby assignments, the standby capacity of the SUs is taken into account. The same SU cannot be reassigned to the same SI, neither as standby nor as active.

VI. INCREMENTAL DESIGN OF AMF CONFIGURATIONS

The validation technique specified in Section IV assigns the SIs to the SUs and returns 'No' if it fails to do so for any SI. In this case we propose to modify the invalid SG by adding resources, namely SUs incrementally, to increase the provided capacities.

At the point where the technique fails to assign an SI, we add SUs to the *SUList* and continue the assignment process. This process continues until all SIs are assigned or until it again fails to assign a certain SI and requires additional SUs. At the end of this incremental process, all SIs must be assigned to the SUs in the augmented *SUList*. The number of additional SUs to be added each time the algorithm fails in assigning a given SI depends on the redundancy model of the SG and in some cases on other configuration attributes.

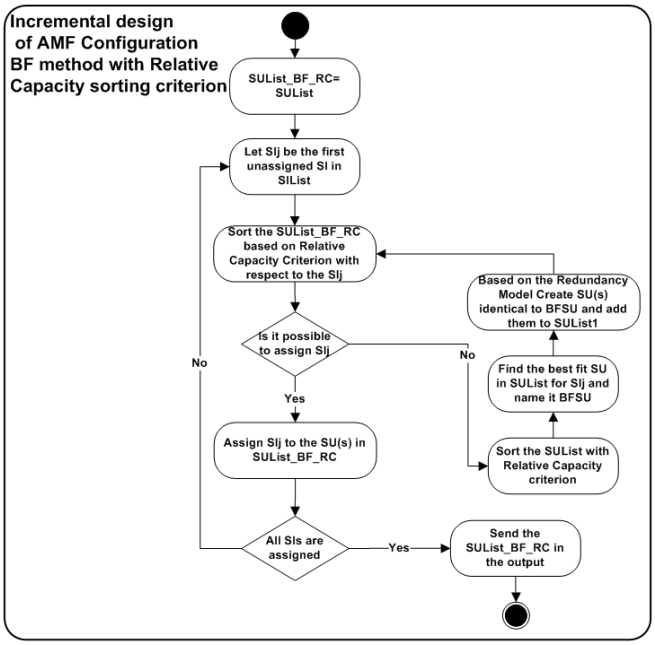

Figure 4. Incremental AMF configuration design using BF method with relative capacity sorting criterion

In the case of the active part of the N+M and N-Way redundancy models only one SU should be added. For the N-Way-Active redundancy model and the standby part of the N-Way, the number is equal to the number of remaining active/standby assignments of the SI in question i.e., if Q assignments of an SI have already taken place before the failing point, the number of additional SUs is equal to *PreferredActiveAssignments* – Q or *PreferredStandbyAssignments* – Q. More specifically, one SU for handling the standby assignment will be added in the case of N+M and *PreferredStandbyAssignments* SUs will be added in the case of the N-Way redundancy model.

The creation of the additional SU(s) varies depending on the applied heuristic method used. More specifically, in the BF method the extra SU(s) for a given SI is/are identical to the first SU in the sorted (increasing order) list of SUs in the *SUList*. However, for a given SI in the WF method, the additional SU(s) is/are identical to the first SU in the sorted (decreasing order) list of SUs in the *SUList*. In other words, the additional SU(s) for a given SI is/are identical to the best fit SU in the BF method and identical to the worst fit SU in the WF method. In order to sort the *SUList*, we use the same sorting criteria as used in the heuristic methods.

It is worth noting that, for the case of the FF approach, the extra SU is simply identical to the first SU in the *SUList* (i.e. the first fit SU). Figure 4 shows the activity diagram for the AMF configuration incremental design method using BF method with the relative capacity as sorting criterion.

In order to illustrate our incremental design approach, let us add three more SIs, $SI_3 = \langle 3,2,1 \rangle$, $SI_4 = \langle 2,1,0 \rangle$, and $SI_5 = \langle 0,1,0 \rangle$ to the example in Figure 3. The *SIList* becomes $\{SI_1, SI_2, SI_3, SI_4, SI_5\}$ with the required capacity list $\{\langle 1,2,0 \rangle, \langle 3,2,1 \rangle, \langle 3,2,1 \rangle, \langle 2,1,0 \rangle, \langle 0,1,0 \rangle\}$, while the *SUList* remains the same. In this example, we use the BF method and we apply the relative capacity criterion for sorting the *SUList*. Figure 5 shows the steps of the approach. As shown in part (2) of Figure 5, the *SUList* is sorted according to the relative capacity criterion of SI_1 (i.e. c_2) in an ascending order. Afterwards, the algorithm finds the first SU in the sorted *SUList* which has the adequate capacity to support SI_1, SU_1, in this case. After the successful assignment of SI_1, the algorithm proceeds to SI_2 by sorting the *SUList* according to the relative capacity of SI_2 and by finding the appropriate SU to support it (part (3) of Figure 5). As presented in part (4) of Figure 5, after sorting the *SUList*, the algorithm succeeds in assigning SI_3 to SU_3. For SI_4, after sorting the *SUList*, the algorithm fails to find an appropriate SU capable of supporting SI_4. This means that the SG cannot protect the set of SIs configured for it and thus the configuration is "likely" not valid. In this case, the algorithm proceeds by adding an extra SU in order to increase the capacity. To do so, the algorithm determines the best fit SU among the SUs of the original *SUList* (see part (1) of Figure 5) and creates an SU with the same capacity, adding it to the *SUList*. As presented in part (6) of Figure 5, SU_4 is created based on the SU_2 and is added to the *SUList* in order to support the load of SI_3. The remaining capacity of the *SUList* is sufficient to support the load of SI_5 and therefore it is assigned to SU_4 see part (8) of Figure 5).

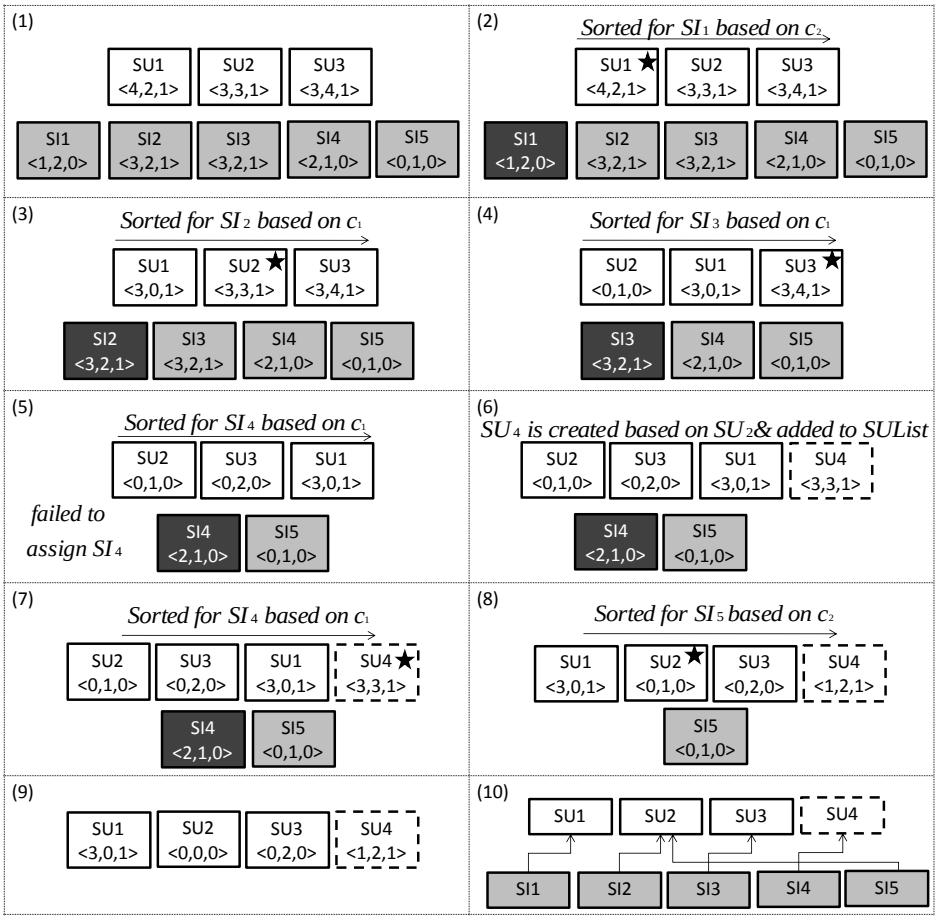

Figure 5. An example for the incremental design approach

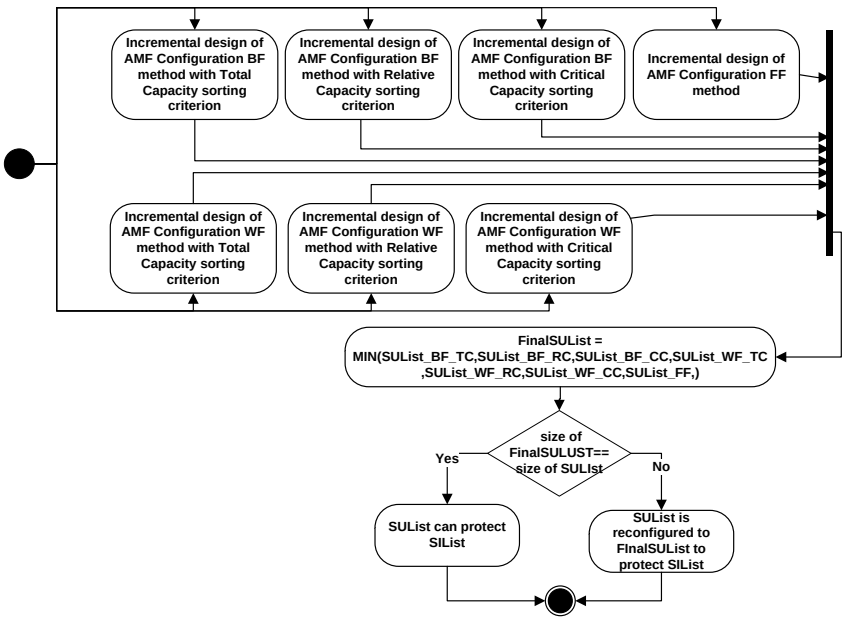

Figure 6. Overview of the incremental design approach

In the last row of Figure 5, part (9) represents the remaining capacity of the *SUList* after the successful assignment of the entire *SIList* and part (10) shows the order of the active assignment of each SI to one of the SUs of the augmented *SUList*.

In order to get the best result, we run seven different heuristics in parallel. Each one will end up with an *SUList*, and the final *SUList* will be the list with the least number of SUs. In other words, the final result will be the *SUList* with minimal additional SUs and therefore, the resources used for protecting services will be relatively minimized. In the case of equality between at least two lists, one may chose the list of SUs with minimal total capacity or the list with maximal total capacity, depending on the design criteria of minimizing resources further or on extendibility. However, comparing lists of SUs with different capacities is not straightforward and further investigations are required. Notice that having a smaller number of SUs lowers the chances of collocating them on the same node which better protects the SIs against node failure. This will facilitate the management of the availability of the applications by the AMF middleware, resulting in the increase of protection level given a fixed number of deployment nodes. Obtaining the original *SUList* as the final result indicates that the input SG is valid and can protect its SIs without any additional SUs. Figure 6 presents the overview of our approach for the incremental design of AMF configurations.

VII. CONCLUSION

Ensuring the protection, at configuration time, of the services as required and according to the specified redundancy model (SI-Protection problem) is one of the most important objectives and challenges in the validation of AMF configurations.

In this paper, we have presented a heuristics based approach to tackle the SI-Protection problem and the validation of AMF configurations, by extending the heuristics introduced for the well-known bin-packing problem. Our approach takes into account the specifics of the domain, i.e. SUs, SIs, and redundancy models.

The precision of the approach is enhanced by embedding seven different methods in order to obtain a better result. When the result is "Yes", we are certain the configuration is valid. However, when it is "No", we are not certain if it is invalid, as this may be a False negative. The precision of the approach depends on different criteria, such as the number of SIs and SUs, as well as the variation of SIs based on their required capacities. In terms of performance, we have tested our approach on a limited number of small scale configurations that were generated automatically by our method [15]. However, these configurations were not appropriate for the performance analysis of the validation approach. In order to analyze the performance of the approach, it is necessary to have a set of large scale configurations. This set also needs to include a variety of configurations in order to cover different criteria such as the variation of SIs or SUs based on the number of CSTypes they require/provide. Therefore, analysing the performance and the accuracy of the approach is a complex task which requires the implementation of a simulation framework for different scenarios. In addition, we will be able to introduce new heuristics focusing on the order of the SIs or alternative sorting criteria. As future work, we will investigate this simulation framework and perform the thorough analysis of the precision and performance of our approach, as well as the design of new heuristics.

As a corollary, we proposed a technique for the incremental modification of "likely" invalid configurations into valid ones. We believe that our technique may lead to over-dimensioned systems, though only by adding a minimal number of extra resources.

ACKNOWLEDGMENT

This work has been partially supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada and Ericsson Software Research.

REFERENCES

[1] C. Oggerino, High Availability Network Fundamentals: A Practical Guide to Predicting Network Availability. 2001: Cisco Press.

[2] Service Availability Forum™, URL: http://www.saforum.org

[3] Service Availability Forum, Service Availability Interface. Overview SAI-Overview-B.05.01.

[4] Service Availability Forum, Application Interface Specification. Availability Management Framework SAI-AIS-AMF-B.04.01.

[5] P. Salehi, et al. "Checking Service Instance Protection for AMF Configurations," in Proc. of the Third IEEE International Conference o Secure Software Integration and Reliability Improvement, Shanghai, China, 2009, pp. 269 - 274.

[6] E. G. Coffman, Jr. , M. R. Garey , D. S. Johnson, Approximation algorithms for bin packing: a survey, Approximation algorithms for NP-hard problems, PWS Publishing Co., Boston, MA, 1996

[7] J. Csirik, et al., "On the multidimensional vector bin packing" European Institute for Advanced Studies in Management, 1990.

[8] S. Albers , M. Mitzenmacher, Average-case analyses of first fit and random fit bin packing, Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms, p.290-299, January 25-27, 1998, San Francisco, California, United States

[9] C. Kenyon, Best-fit bin-packing with random order, Proceedings of the seventh annual ACM-SIAM symposium on Discrete algorithms, p.359-364, January 28-30, 1996, Atlanta, Georgia, United States

[10] G. Dósa, The tight bound of first fit decreasing bin-packing algorithm is FFD(I)≤11/9OPT(I)+6/9. In: Proc. of the 1st International Symposium on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies (ESCAPE2007), pp. 1–11 (2007)

[11] M. R. Garey, and D. S. Johnson, Computers and Intractability, A Guide to the Theory of NP-Completeness, W. H. Freeman and Co., San Francisco, Calif. (1979).

[12] B. Patt-Shamir, and D. Rawitz, "Vector Bin Packing with Multiple-Choice", in Proc. the 12th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT), Bergen, Norway, 2010, pp. 248-259.

[13] C.S. Rao, et al., "Improved Approximation Bounds for Vector Bin Packing". Arxiv preprint arXiv:1007.1345, 2010.

[14] J. Pearl, "Heuristics: Intelligent Search Strategies for Computer Problem Solving". New York, Addison-Wesley,1984.

[15] P. Salehi, et al., "A Model Driven Approach for AMF Configuration Generation", in Proc of the 6th Work. on System Analysis and Modelling, Oslo, Norway, 2010.