Fault-Tolerant Application Placement in Heterogeneous Cloud Environments

Bart Spinnewyn, Bart Braem, Steven Latré
University of Antwerp - iMinds,
Department of Mathematics & Computer Science,
Middelheimlaan 1, 2020 Antwerp, Belgium
{bart.spinnewyn, bart.braem, steven.latre}@uantwerpen.be

***Abstract*—The Internet of Things (IoT) has inspired a myriad of real-time applications, such as robotics and human-machine interaction. Many IoT applications have significant computational requirements, while at the same time they demand very low latencies. The cloud can provide the needed resources on-demand, however often fails to meet these timing requirements. Low response time can only be realized by having computational infrastructure in close vicinity. Therefore we investigate to what extent the cloud can be extended in the direct wireless surroundings of the IoT devices. This environment is highly heterogeneous as it comprises a wide variety of devices, connected using a plethora of technologies (both wired and wireless). A direct implication is that, compared to traditional cloud infrastructure, many of those nodes and links are likely to fail. We propose an application placement that can overcome failure-related challenges. We demonstrate that availability-awareness can increase the number of applications that can be hosted simultaneously by 132%. Furthermore we find that an additional increase of 54% can be realized through redundant provisioning of resources.**

I. Introduction

In recent years we have seen the uptake of many connected objects, generally referred to as the Internet of Things (IoT). While IoT applications typically have a significant computational aspect, IoT devices generally lack the computational power to execute Central Processing Unit (CPU) intensive tasks locally. Additionally these devices are limited in connectivity, as they connect using low-power wireless technology, which is unreliable by nature [1].

To address the computational requirements of IoT applications, tasks are often delegated to a cloud environment. This frees the enterprise and the end-user from the specification of many details, but brings an increase in latency. Latency-sensitive applications however require nodes in the vicinity to meet their delay requirements.

Recent efforts have been made to reduce latency, by geographical distribution of the cloud closer to the end-user. In this light we have seen the conception of edge-clouds and fog computing [2]. Edge-clouds push the location of computing, applications, data, and services away from centralized nodes towards the edge of the network [3].

Fog computing places computation nodes as close as the field area network. However, even when placed in the (wired) access network, the introduced latency may be too high for certain applications such as robotics which require ultra small response times. In other cases (e.g., in the case of a disaster) the physical infrastructure may be non-existing. Therefore, we investigate to what extent the cloud computing paradigm can be extended to the wireless network that surrounds the IoT devices. This results in a very heterogeneous cloud environment both in terms of node capability and channel performance as all IoT devices (e.g., sensors, wearables, laptops) could be used as cloud nodes.

When node-failure is considered for high-availability applications, it usually suffices to slightly over-provision Virtual Machines (VMs) and host them at multiple physical locations. This so-called *Active/Active* configuration has both primary and backup virtual resources on-line at the same time. Another approach is *Active/Passive*, which relies on having inactive backups. Failure of primary resources is handled by switch-over. Backup resources can be shared amongst multiple applications. This is motivated by the reliable nature of infrastructure, as the probability of multiple applications failing simultaneously is virtually zero.

In contrast to a traditional cloud, infrastructure in a wireless environment can be highly *unreliable*. Those devices can be battery-powered, connected over wireless links and physically move throughout the environment, which implicates that devices can enter and leave the network at any time. Rather than assuming one component or more components to fail, one must consider the entire distribution of failures. According to Herker et al. this topic remains unexplored [4].

For an extension of the cloud into the direct IoT-surroundings to be viable, cloud management algorithms must render the unreliable nature of local infrastructure transparent to the cloud user. To the best of our knowledge no such algorithms have been developed yet.

This paper addresses two research questions. First, to what degree can availability-awareness improve the efficiency of application placement on an unreliable substrate network? Second, can the placement ratio further be improved by redundant placement of services and virtual links?

The major contributions of this paper are threefold: first, we provide a computational availability-model that incorporates both node- and link-failure. Second, we employ an *Active/Active* redundancy formalism to improve the placement ratio. Our approach elegantly unifies both node and link replication. Third, we formulate the problem as an Integer

Linear Program (ILP). This is a very generic optimization model, for which a wide range of solution techniques exist. For an overview on solution approaches (both exact and approximate), consider [5]. In this paper we limit ourselves to a hybrid solver, which combines various solution techniques, to find optimal solutions to this newly formulated problem.

The remainder of this paper is structured as follows: related work is discussed in Section II. The problem is stated in Sections III and an implementation as an ILP is presented in Section IV. Subsequently, the set-up and results of performance evaluations are provided in Section V. Finally, the paper is concluded in Section VI.

II. Related work

First we briefly discuss related work in the general application placement domain. Subsequently we focus on the aspect of availability-awareness.

A. Application placement

The basic objective for a cloud is to provide users with the resources they demand. Those (virtualized) resources are realized by a complex physical infrastructure. Such an infrastructure is managed by highly automated management software, of which application placement is an integral part. Application placement performs both *admission control*, which determines *if* application requests can be accepted and the actual placement, determining *how/where* the required virtual resources will be hosted.

Early work on application placement merely considers nodal resources, such as CPU and memory capabilities. Deciding whether requests are accepted and where those virtual resources are placed then reduces to a Multiple Knapsack Problem (MKP) [6]. An MKP is known to be NP-hard and therefore optimal algorithms are hampered by scalability issues. A large body of work has been devoted to finding heuristic solutions. For instance, Xu et al. focus on the multi-objective VMs placement problem [7]. They propose a genetic algorithm with fuzzy multi-objective evaluation for efficiently searching the large solution space and conveniently combining possibly conflicting objectives. While Yi et al. propose an evolutionary game theoretic framework for adaptive and stable application deployment in clouds [8]. Other works include Network Interface Card (NIC) capabilities as a dimension in the MKP [9] and assumes an over-provisioned inner-network. While plausible within the boundaries of one datacenter, this condition rarely holds when a combination of multiple clouds or even a wireless environment is considered. When the application placement not only decides where computational entities are hosted, but also decides on how the communication between those entities is routed in the substrate network, then we speak of *network-aware* application placement. Network-aware application placement is closely tied to Virtual Network Embedding (VNE).

An example of a network-aware approach is the work from Moens et al. [10]. It employs an Service Oriented Architecture (SOA), in which applications are constructed as a collection of communicating services. This optimal approach performs node and link mapping simultaneously. In contrast, other works try to reduce computational complexity by performing those tasks in distinct phases [11], [12].

B. Availability-awareness in clouds

We define the availability of an application as the probability of it being accessible. On the one hand availability depends on the workload as excess demand can cause denial of service. For instance Breitgand et al. discuss how workload variations affect Service Level Agreement (SLA)-conformity [13]. Another cause for unavailability is failure in the substrate network, potentially causing virtual resources and applications to be off-line. In a wireless environment this aspect is most pressing. Jayasinghe et al. model cloud infrastructure as a tree structure with arbitrary depth [14]. Physical hosts on which VMs are hosted are the leaves of this tree, while the ancestors comprise Regions and Availability Zones. The nodes at bottom level are physical hosts where VMs are hosted.

Wang et al. were the first to provide a mathematical model to estimate the resulting availability from such a tree structure [15]. They calculate the availability of a single Virtual Machine (VM) as the probability that neither the leaf itself, nor any of its ancestors fail. Their work focuses on handling workload variations by a combination of vertical and horizontal scaling of VMs. Horizontal scaling launches or suspends additional VMs, while vertical scaling alters VM dimensions. The joint availability is then the probability that at least one of the VMs is available. While their model suffices for traditional clouds, it is ill-suited for a heterogeneous cloud environment as link failure and bandwidth limitations are disregarded.

A different approach is to try and guarantee that a virtual network can still be embedded in a physical network, after k network components fail. Ajtai et al. provide a theoretical framework for fault-tolerant graphs [16]. However in this model, hardware failure can still result in service outage as migrations may be required before normal operation can continue. As opposed to that, Yeow et al. define reliability as the probability that critical nodes of a virtual infrastructure remain in operation over all possible failures [17]. They propose an *Active/Passive* approach in which backup resources are pooled and shared across multiple virtual infrastructures. Their algorithm first determines the required redundancy level and subsequently performs the actual placement. However, decoupling those two operations is only permissible when link failure can be omitted and nodes are homogeneous.

Summarized, to the best of our knowledge, no cloud application placement algorithm exists which has a computational model for availability that regards both node and link failure. The optimal approach from Moens et al. [10] is not availability-aware, but matches our approach most closely and will be used as a reference in Section V.

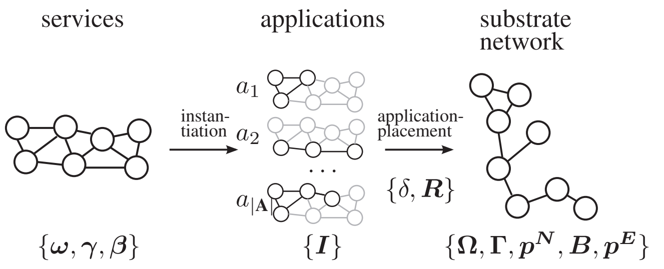
services
applications
substrate network
a_1
instan-tiation
a_2
application-placement
$\{\delta, \boldsymbol{R}\}$
$a_{|\mathbf{A}|}$
$\{\boldsymbol{\omega}, \boldsymbol{\gamma}, \boldsymbol{\beta}\}$
$\{\boldsymbol{I}\}$
$\{\boldsymbol{\Omega}, \boldsymbol{\Gamma}, \boldsymbol{p^N}, \boldsymbol{B}, \boldsymbol{p^E}\}$

Fig. 1: Overview of this work: applications $\{\boldsymbol{\omega}, \boldsymbol{\gamma}, \boldsymbol{\beta}\}$, composed of services $\{\boldsymbol{I}\}$, are placed on a substrate network where node $\{\boldsymbol{p^N}\}$ and link failure $\{\boldsymbol{p^E}\}$ is modelled. By increasing the redundancy δ we can guarantee a minimum availability level $\boldsymbol{R}$.

III. Problem statement

First we describe our model for application requests and substrate network. Subsequently we build on this to describe the application placement problem. An overview is provided in Figure 1. The symbols used are listed in Table I.

Symbol	Description
A	set of requested applications
S	set of services
ω_s	CPU requirement of service s
γ_s	memory requirement of service s
β_{s_1,s_2}	bandwidth requirement between services s_1 and s_2
$I_{a,s}$	instantiation of service s by application a: 1 if instanced, else 0
N	set of physical nodes comprising the substrate network
E	set of physical links (edges) comprising the substrate network
Ω_n	CPU capacity of node n
Γ_n	memory capacity of node n
p_n^N	probability of failure of node n
B_e	bandwidth capacity of link e
p_e^E	probability of failure of link e
R_a	required joint availability of application a: lower bound on the probability that at least one of the duplicates for a is available
δ	maximum allowed number of duplicates

TABLE I: Overview of input variables to the Cloud Application Placement Problem (CAPP).

A. Application graph

This paper considers an SOA, in which an application is described as its composition of services. The collected composition of all requested applications is represented by the instance matrix ($\boldsymbol{I}$).

Services have certain CPU ($\boldsymbol{\omega}$) and memory requirements ($\boldsymbol{\gamma}$). Additionally, bandwidth ($\boldsymbol{\beta}$) is required by the virtual links between any two services. A sub-modular approach allows sharing of memory resources amongst services belonging to multiple applications.

B. Network graph

Consider a substrate network consisting of nodes and links. Nodes have certain CPU ($\boldsymbol{\Omega}$) and memory capacities ($\boldsymbol{\Gamma}$). Physical links between nodes are characterized by a given bandwidth ($\boldsymbol{B}$). Both links and nodes have a known probability of failure, $\boldsymbol{p^N}$ and $\boldsymbol{p^E}$ respectively. Failures are considered to be independent.

C. Application placement problem

Availability not only depends on failure in the substrate network, but also on how the application is placed. *Nonredundant* application placement assigns each service and virtual link at most once, while its redundant counterpart can place those virtual resources more than once. In this work we limit ourselves to a static failure model, i.e. availability only depends on the current state of the network.

Availability in case of nonredundant placement is fairly straightforward. An application is available if none of the physical nodes and links that contribute to its placement fail. The case of redundant placement is discussed in Section IV.

IV. Formal problem description

This section is structured as follows: in Section IV-A we elaborate how we will implement node and link replication simultaneously. Subsequently, we formulate the problem as a binary ILP. The input variables to the algorithm were already introduced throughout the problem statement. Given those input variables, the algorithm finds a value for the decision variables listed in Section IV-B that minimizes the objective function (Section IV-D). The optimization is subject to the constraints listed in Section IV-C. Finally in Section IV-E we illustrate the concepts used in the ILP.

A. Redundant application placement

To allow both node and link replication, we introduce a *duplicate* as a complete placement of an application. A duplicate is either placed or not placed. If more than one duplicate is placed, the placement is said to introduce *redundancy*. A duplicate is available if none of the components used for its placement fail. The joint availability of an application is then the probability that at least one of its duplicates is available.

Duplicates of the same application can share physical components. An advantage of this is that since no more than one of these duplicates needs to be active simultaneously, they can share CPU, memory and networking resources.

The concept of duplicates, now allows us to reformulate the problem statement more precisely as:

"Given an application and network graph, how to place an application on a network as to maximize the number of placed applications, while satisfying a required level of availability for each application (R) and considering a maximum of δ duplicates per application?"

B. Decision variables

The decision variables are described in Table III. O indicates which application requests are accepted, while G provides detailed information about which duplicates are actually placed. Information about the assignment of services to physical nodes is contained in π, Π and U, while υ and Υ tell us how the virtual links are routed over the physical links. K, τ and T are directly used for availability calculation. Auxiliary variables are described in Table II.

C. Constraints

1) Admission control: At most, δ duplicates can be placed for each application:

$$|\mathbf{D}| = \delta. \tag{1}$$

An application can only be accepted if at least one of its duplicates is placed:

$$\forall a \in \mathbf{A} : O^a \leq \sum_{d \in \mathbf{D}} G^{d,a}. \tag{2}$$

2) Node-embedding: Nodal resources are only assigned to duplicates if they are considered placed:

$$\forall a \in \mathbf{A}, s \in \mathbf{S}, n \in \mathbf{N}, d \in \mathbf{D} : \pi^{d,a}_{s,n} \leq G^{d,a} \times I_{a,s}. \tag{3}$$

The number of services hosted for each accepted duplicate equals the total number of instantiated services. If a duplicate is not placed, no services are instantiated:

$$\forall a \in \mathbf{A}, d \in \mathbf{D} : G^{d,a} \times \sum_{s \in \mathbf{S}} I_{a,s} = \sum_{s \in S} \sum_{n \in N} \pi^{d,a}_{s,n}. \tag{4}$$

If a service is hosted on a node, then CPU resources must be provisioned:

$$\forall a \in \mathbf{A}, d \in \mathbf{D}, s \in \mathbf{S}, n \in \mathbf{N} : \pi^{d,a}_{s,n} \leq \Pi^a_{s,n}. \tag{5}$$

Per service each service is hosted on at most one node:

$$\forall a \in \mathbf{A}, d \in \mathbf{D}, s \in \mathbf{S} : \sum_{n \in \mathbf{N}} \pi^{d,a}_{s,n} \leq 1. \tag{6}$$

Symbol	Description
$\mathbf{C}$	set of physical components in the substrate network, i.e. nodes and edges ($\mathbf{C} = \mathbf{N} \cup \mathbf{E}$)
$\mathbf{D}$	set of duplicates
$\mathbf{M}$	set of minterms
$\mathbf{X}$	set of all possible states
$\mathbf{X}(m)$	particular state of the substrate network, the state of each component follows from m according to Equations 18 and 19
χ_c	state of physical component c
$b_c(m)$	value of χ_c for component c in minterm m
$\zeta(d,a)$	availability of duplicate d for application a
$Z(a)$	joint availability of application a

TABLE II: Overview of auxiliary symbols used throughout the formulation of the ILP.

Symbol	Description
O^a	acceptance of application a: 1 i.f.f. accepted
$G^{d,a}$	placement of duplicate d of application a: 1 i.f.f. placed
$\pi^{d,a}_{s,n}$	placement of service s for duplicate d of application a on node n: 1 i.f.f. hosted
$\Pi^a_{s,n}$	use of node n for hosting of service s by application a: 1 i.f.f. used
$U_{s,n}$	hosting of service s on node n: 1 i.f.f. hosted
$\upsilon^{d,a}_{s_1,s_2}(e)$	placement of virtual link between services s_1 and s_2 on physical link e for duplicate d of application a: 1 i.f.f. placed
$\Upsilon^a_{s_1,s_2}(e)$	use of physical link e by at least one duplicate of application a for the placement of the virtual link between s_1 and s_2: 1 i.f.f. placed
$K^{d,a}_c$	use of physical component c by duplicate d of application a: 1 i.f.f. used
$\tau^{d,a}_m$	coverage of minterm m by duplicate d of application a: 1 i.f.f. covered m
T^a_m	availability of application a when the state of the network equals $\mathbf{X}(m)$: 1 i.f.f. available

TABLE III: Overview of decision variables to the binary ILP: variables can only assume 0 or 1.

Conservation of CPU and memory resources dictates:

$$\forall n \in \mathbf{N} : \sum_{a \in \mathbf{A}} \sum_{s \in \mathbf{S}} \Pi^a_{s,n} \times \omega_s \leq \Omega_n \tag{7}$$

and

$$\forall n \in \mathbf{N} : \sum_{s \in S} U_{s,n} \times \gamma_s \leq \Gamma_n. \tag{8}$$

A service must be hosted on a node, as soon as it is used by one of the duplicates:

$$\forall s \in \mathbf{S}, \forall n \in \mathbf{N} : \sum_{a \in \mathbf{A}} \sum_{d \in \mathbf{D}} \pi^{d,a}_{s,n} \leq U_{s,n} \times |\mathbf{D}| \times \sum_{a \in \mathbf{A}} I_{a,s}. \tag{9}$$

3) Link-embedding: Multi Commodity Flow (MCF) constraints on each node can be expressed as: $\forall a \in \mathbf{A}, s_1, s_2 \in \mathbf{S}, d \in \mathbf{D}, n_1 \in \mathbf{N}$:

$$\sum_{(n_1,n_2) \in \mathbf{E}} \upsilon^{d,a}_{s_1,s_2}(n_1,n_2) - \sum_{(n_2,n_1) \in \mathbf{E}} \upsilon^{d,a}_{s_1,s_2}(n_2,n_1) = \pi^{d,a}_{s_1,n_1} - \pi^{d,a}_{s_2,n_1}. \tag{10}$$

$\Upsilon^a_{s_1,s_2}(e)$ indicates if at least one of an application's duplicates uses e for this virtual link: $\forall a \in \mathbf{A}, s_1 \in \mathbf{S}, s_2 \in \mathbf{S}, e \in \mathbf{E}, d \in \mathbf{D}$:

$$\upsilon^{d,a}_{s_1,s_2}(e) \leq \Upsilon^a_{s_1,s_2}(e). \tag{11}$$

The total bandwidth used per link cannot exceed the total link capacity:

$$\forall e \in \mathbf{E} : \sum_{s_1 \in \mathbf{S}} \sum_{s_2 \in \mathbf{S}} \sum_{a \in \mathbf{A}} \Upsilon^a_{s_1,s_2}(e) \times \beta_{s_1,s_2} \leq B_e. \tag{12}$$

4) Availability-awareness: For a duplicate to be available, each of the individual components it uses must be available. A component is used by a duplicate if it hosts any of the duplicate's services or virtual links: $\forall a \in \mathbf{A}, d \in \mathbf{D}, c \in \mathbf{C}, s_1, s_2 \in \mathbf{S}$:

$$K_c^{d,a} \geq \begin{cases} \pi_{s_1,c}^{d,a} & \text{if } c \in \mathbf{N} \\ \Upsilon_{s_1,s_2}^{d,a}(c) & \text{if } c \in \mathbf{E} \end{cases}. \tag{13}$$

The state of an individual component is described as:

$$\forall c \in \mathbf{C} : \chi_c = \begin{cases} 0 & \text{if } c \text{ fails} \\ 1 & \text{if } c \text{ does not fail} \end{cases}. \tag{14}$$

The probability that a component fails is given by:

$$\forall c \in \mathbf{C} : \mathrm{P}\left[\chi_c = 0\right] = \begin{cases} p_c^N & \text{if } c \in \mathbf{N} \\ p_e^N & \text{if } c \in \mathbf{E} \end{cases}. \tag{15}$$

The state of the substrate network can then be described as:

$$\mathbf{X} = (\chi_1, \chi_2, \ldots, \chi_{|\mathbf{C}|}). \tag{16}$$

We introduce the following notation to systematically describe all possible states of the substrate network, which we will further refer to as minterms:

$$\mathbf{M} = \{0, 1, \ldots, 2^{|\mathbf{C}|} - 1\} \tag{17}$$

and

$$\forall m \in \mathbf{M} : \mathbf{X}(m) = \mathbf{X} | \forall c \in \mathbf{C} : \chi_c = b_c(m), \tag{18}$$

where $b_c(m) \in \{0, 1\}$ is defined by:

$$\forall m \in \mathbf{M} : m = \sum_{c \in \{0,1,\ldots,|\mathbf{C}|-1\}} b_c(m) \times 2^{c-1}. \tag{19}$$

As component failures are assumed independent, the probability of each minterm is given by:

$$\begin{aligned} &\forall m \in \mathbf{M} : \mathrm{P}\left[\mathbf{X} = \mathbf{X}(m)\right] \\ &= \prod_{c \in \mathbf{C} | b_c(m) = 0} \mathrm{P}\left[\chi_c = 0\right] \times \prod_{c \in \mathbf{C} | b_c(m) = 1} \mathrm{P}\left[\chi_c = 1\right]. \end{aligned} \tag{20}$$

As stated earlier, a duplicate is available, if all physical components that contribute to its placement are on-line:

$$\forall a \in \mathbf{A}, d \in \mathbf{D} : \zeta(a, d) = \mathrm{P}\left[\bigcap_{c \in \mathbf{C}} (\chi_c{=}1) \cup (K_c^{d,a}{=}\,0)\right] \tag{21}$$

$$= \sum_{m \in \mathbf{M}} \tau_m^{d,a} \mathrm{P}\left[\mathbf{X}{=}\mathbf{X}(m)\right], \tag{22}$$

where we made use of Bayes' theorem:

$$\tau_m^{d,a} = \mathrm{P}\left[\bigcap_{c \in \mathbf{C}} (\chi_c{=}1) \cup (K_c^{d,a}{=}0) | \mathbf{X}{=}\mathbf{X}(m)\right]. \tag{23}$$

For the ILP this is reformulated as Equation 24. An additional $G^{d,a}$ term ensures that no minterm is covered when a duplicate is not placed ($G^{d,a}{=}0$): $\forall m \in \mathbf{M}, d \in \mathbf{D}, c \in \mathbf{C}, a \in \mathbf{A}$:

$$\tau_m^{d,a} \leq G^{d,a} + (b_c(m) - 1)\, K_c^{d,a}. \tag{24}$$

Finally, an application is available if at least one of its duplicates is available:

$$\forall a \in \mathbf{A}, m \in \mathbf{M} : T_m^a = \mathrm{P}\left[\bigcup_{d \in \mathbf{D}} \tau_m^{d,a}\right], \tag{25}$$

which can be formulated as:

$$\forall m \in \mathbf{M}, a \in \mathbf{A} : T_m^a \leq \sum_{d \in \mathbf{D}} \tau_m^{d,a}. \tag{26}$$

The joint availability of an application is then given by:

$$\forall a \in \mathbf{A} : Z(a) = \sum_{m \in \mathbf{M}} T_m^a \mathrm{P}\left[\mathbf{X}{=}\mathbf{X}(m)\right]. \tag{27}$$

Finally the condition that an application is only placed if the joint availability exceeds R_a can be written as:

$$\forall a \in \mathbf{A} : 1 - O^a + \sum_{m \in \mathbf{M}} T_m^a \mathrm{P}\left[\mathbf{X}{=}\mathbf{X}(m)\right] \geq R_a. \tag{28}$$

D. Objective function

The placement is sequentially optimized in multiple steps. In each step an objective function is minimized and results of previous steps are added as equality constraints. The objective functions are listed in the order in which they are used by the algorithm.

Maximize acceptance:

$$f_1(\mathbf{A}) = -\sum_{a \in \mathbf{A}} O^a. \tag{29}$$

Minimize bandwidth usage:

$$f_2(\mathbf{A}, \mathbf{E}, \mathbf{S}, \boldsymbol{\beta}) = \sum_{a \in \mathbf{A}} \sum_{e \in \mathbf{E}} \sum_{s_1, s_2 \in \mathbf{S}} \Upsilon_{s_1,s_2}^a(e) \times \beta_{s_1,s_2}. \tag{30}$$

Minimize CPU resources usage:

$$f_3(\mathbf{A}, \mathbf{N}, \mathbf{S}, \boldsymbol{\omega}) = \sum_{n \in \mathbf{N}} \sum_{a \in \mathbf{A}} \sum_{s \in \mathbf{S}} \Pi_{s,n}^a \times \omega_s. \tag{31}$$

Minimize the number of duplicates used:

$$f_4(\mathbf{A}, \mathbf{D}) = \sum_{a \in \mathbf{A}} \sum_{d \in \mathbf{D}} G^{d,a}. \tag{32}$$

The last objective function ensures that multiple duplicates of the same application are only placed if beneficial to maximize the placement ratio, or minimize resource usage.

E. Illustration

For a quick motivating example consider Figure 2. Application a consists of services s_1 and s_2. A one-directional virtual link (s_1, s_2) is required between them. This application is placed on a substrate network consisting of two physical nodes, interconnected by physical link (n_1, n_2).

The placement algorithm considers two possible duplicates: duplicate 1 places the services on multiple nodes, whereas duplicate 2 places both processes on the same physical node (see Table IV).

Table V provides an overview on how the availability of a can be calculated. First, there are 3 components, hence

State of the substrate network					Coverage of minterms			
m	χ_1	χ_2	χ_3	$\mathrm{P}[\mathbf{X}{=}\mathbf{X}(m)]$	$\tau_m^{d,a}$	$\tau_m^{1,a}$	$\tau_m^{2,a}$	T_m^a
0	0	0	0	$\mathrm{P}[\chi_1{=}0] \times \mathrm{P}[\chi_2{=}0] \times \mathrm{P}[\chi_3{=}0]$	$\neg(K_1^d \vee K_2^d \vee K_3^d)$	0	0	0
1	0	0	1	$\mathrm{P}[\chi_1{=}0] \times \mathrm{P}[\chi_2{=}0] \times \mathrm{P}[\chi_3{=}1]$	$\neg(K_1^d \vee K_2^d)$	0	0	0
2	0	1	0	$\mathrm{P}[\chi_1{=}0] \times \mathrm{P}[\chi_2{=}1] \times \mathrm{P}[\chi_3{=}0]$	$\neg(K_1^d \vee K_3^d)$	0	0	0
3	0	1	1	$\mathrm{P}[\chi_1{=}0] \times \mathrm{P}[\chi_2{=}1] \times \mathrm{P}[\chi_3{=}1]$	$\neg(K_1^d)$	0	0	0
4	1	0	0	$\mathrm{P}[\chi_1{=}1] \times \mathrm{P}[\chi_2{=}0] \times \mathrm{P}[\chi_3{=}0]$	$\neg(K_2^d \vee K_3^d)$	1	0	1
5	1	0	1	$\mathrm{P}[\chi_1{=}1] \times \mathrm{P}[\chi_2{=}0] \times \mathrm{P}[\chi_3{=}1]$	$\neg(K_2^d)$	1	0	1
6	1	1	0	$\mathrm{P}[\chi_1{=}1] \times \mathrm{P}[\chi_2{=}1] \times \mathrm{P}[\chi_3{=}0]$	$\neg(K_3^d)$	1	0	1
7	1	1	1	$\mathrm{P}[\chi_1{=}1] \times \mathrm{P}[\chi_2{=}1] \times \mathrm{P}[\chi_3{=}1]$	$\neg(0)$	1	1	1

TABLE V: States of the substrate network and corresponding coverage of minterms for the example shown in Figure 2.

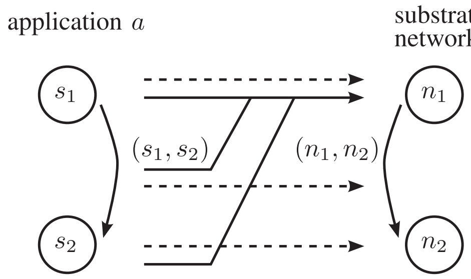

Fig. 2: Sample placement incorporating replication: the solid and dashed line represent the placement of duplicate 1 and 2 respectively.

	node n_1 component 1	node n_2 component 2	edge (n_1, n_2) component 3
duplicate 1	1	0	0
duplicate 2	1	1	1

TABLE IV: Physical components used (K) corresponding to Figure 2

the network has $2^3 = 8$ possible states. Next to that, the availability of individual duplicates ($\boldsymbol{\tau}$) is shown. Duplicate 1 covers minterms 4 through 7. Duplicate 2 does not cover any additional minterms. As a result the application is available for minterms 4 through 7 ($\boldsymbol{T}$), which correspond to the states for which χ_1 is on-line:

$$Z(a) = \sum_{m \in \mathbf{M}} T_m^{d,a} \mathrm{P}[\mathbf{X}{=}\mathbf{X}(m)] \tag{33}$$
$$= \mathrm{P}[\chi_1{=}1]. \tag{34}$$

V. Performance evaluation

To evaluate the performance of our algorithm we simulate a cloud environment with unreliable nodes.

The placement ratio is the ratio of applications that meet the availability requirement to the total of application requests:

$$\text{placement ratio} = \frac{\sum_{a \in \mathbf{A}} \mathrm{Q}(a)}{|\mathbf{A}|}, \tag{35}$$

where

$$\forall a \in \mathbf{A} : \mathrm{Q}(a) = \begin{cases} 0 & \text{if } Z(a) \leq R_a \\ 1 & \text{if } Z(a) > R_a \end{cases}. \tag{36}$$

The CPU Load Factor (CLF) is the ratio of total CPU demand of all application requests (assuming no duplication) to the total amount of available CPU resources. Consequently the CLF is a measure for the loading of the substrate network:

$$\mathrm{CLF} = \frac{\sum_{s \in \mathbf{S}} \sum_{a \in \mathbf{A}} I_{a,s} \times \omega_s}{\sum_{n \in \mathbf{N}} \Omega_n}. \tag{37}$$

A. Simulation model

Algorithm	**Author**	**δ**	**Availability-aware**	**Redundancy considered**
1	this work	1	yes	no
2	this work	2	yes	yes
3	this work	3	yes	yes
4	Moens et al. [10]	-	no	no

TABLE VI: Algorithms used throughout the evaluation. Redundancy is considered when more than one duplicate *can* be placed. When more than one duplicate is de facto placed, we say that redundancy is introduced.

Throughout the simulations, we evaluate 4 algorithms. An overview is provided in Table VI. We compare our algorithm with a maximum of 1, 2, or 3 duplicates per application and the algorithm from Moens et al. To simplify comparison we adjusted their algorithm in two ways: first, the bandwidth requirement of a virtual link must be fully met for a link to be placed, instead of originally 80%. Second, the objective is to place as many applications as possible, instead of placing as many CPU resources as possible.

For a required availability level of 0% the availability constraints are void. Therefore application placement does not benefit from neither availability-awareness, nor redundancy and all placement ratios are equal.

For one duplicate, our algorithm can be regarded as an availability-aware extension to traditional application placement, for two and three duplicates the algorithm can also introduce redundancy.

Two types of simulation set-ups are used:

1) Fixed substrate network dimensions: We evaluate each algorithm for an availability requirement of 0%, 90% and 99%. The algorithms considered are: our algorithm with 1, 2 and 3 duplicates respectively and the algorithm from Moens et al. Hence, 12 simulations are performed per *run*. A run is defined as a combination of application and substrate network. We create 10 bins for the CLF in the interval $[0; 1]$ and for each bin we generate 100 runs, which means in total 1000 runs. All runs have equal substrate network dimensions and an identical number of applications.

The application graph comprises 10 applications, composed of 3 services. Service requirements are modelled by $\omega_s \in \{20, 100\}$ and $\gamma_s \in \{75, 100\}$. Each service has 60% chance to be a part of a given application and each service is part of at least one application. Additionally, each application consists of at least one service. The bandwidth requirements are symmetric (i.e. $C_{s_1,s_2} = C_{s_2,s_1}$) and uniformly distributed in the interval $[0.02; 0.04]$.

The substrate network consists of 5 nodes and 8 undirected edges. Although the network dimensions are fixed, the topology is not. We only evaluate our algorithm using spanning trees, as fragmentation of resources would further increase variance. To generate a random spanning tree, we first construct a minimal spanning tree [18], and then add edges until the desired number of edges has been reached.

The bandwidth between nodes is symmetric (i.e. $B_{n_1,n_2} = B_{n_2,n_1}$) and equals either 0 or 1. Node and link failure are individually chosen randomly from the set $\{0\%, 2.5\%, 5\%\}$. Nodal resources are distributed as follows: nodal CPU and memory capacities are respectively $\Omega_n \in \{50; 200; 1000; 5000\}$ and $\Gamma_n \in \{100; 150; 200\}$

2) Variable substrate network dimensions: Using this set-up, the impact of substrate network dimensions is evaluated. Only one algorithm and one level of required availability is simulated for all runs, namely our algorithm with δ=2 and a required availability of 90%. The dimensions of the substrate network are chosen as $|\mathbf{N}| \in \{4, 5, 6\}$ and $|\mathbf{E}| \in \{3, 4, 5, 6, 7, 8, 9\}$. Additionally, only connected networks are considered:

$$|\mathbf{N}| + 1 \leq |\mathbf{E}| \leq \binom{|\mathbf{N}|}{2}. \tag{38}$$

For each combination of $|\mathbf{N}|$ and $|\mathbf{E}|$, 100 runs are generated, totalling 1500 runs. The remaining parameters are generated similarly to the previous set-up.

B. Results description

The optimization is performed using Gurobi 6.0.3 on the High Performance Cluster (HPC) core facility CalcUA at the University of Antwerp. The simulation results are preprocessed: the values shown in Figure 3, 4 and 5 each result from averaging out 100 simulations, error bars represent the first and third quantile.

1) Fixed substrate network dimension: Performance of the algorithm is evaluated as a function of the CLF. The results are shown for multiple levels of required availability and allowed number of duplicates.

First, the placement ratio is shown in Figure 3. For a required availability level (R_a) equal to zero, there is no availability constraint and all placement ratios equal the one from Moens et al.

When the required availability level is set to 90% we see a severely reduced placement ratio for Moens et al. while our algorithm still manages to accept the same number of applications. Relative to Moens et al. our availability-aware approach yields an increase of 36% in placement ratio. There is no additional advantage in using multiple duplicates.

As the required availability is further increased to 99% the placement ratio drops for all algorithms. The performance of our algorithms is however vastly superior. Relative to Moens et al. using one duplicate yields an improvement of 132%. From one duplicate to two duplicates there is an additional improvement of 54%. We report no benefit in considering more than 2 duplicates.

The large spread on measurements can be attributed to discrete steps in the placement ratio as only 10 applications are placed. The spread is largest about the middle of the CLF-range, where a larger variety of substrate networks fit those bins, than at the extreme end of the range.

The computation time is shown in Figure 4, the time required to formulate the problem model is omitted.

For Moens et al. the computation time is identical for (a), (b) and (c) as the application is not aware of availability requirements. The computation time for our algorithm with one duplicate is fairly independent of the required availability level. On average its computation time is 12 times higher when compared to the non-availability-aware approach. Availability-awareness comes at the expense of more computation time.

When redundancy is considered, computation time is highly dependent on required availability level. For instance consider the computation time of 2 duplicates versus 1 duplicate, for 90% this is 10 times slower, while for 99% this is even 40% slower. Next to that computation time increases as the level of redundancy to be considered increases. Consider the computation time of 3 duplicates versus 2 duplicates, for 90% and 99% this is respectively 3 and 11 times slower. Computation time clearly increases as the considered level of redundancy goes up. Additionally, the CLF impacts computation time, although this relation is more subtle.

2) Variable substrate network dimensions: The computation time is shown as a function of substrate network dimensions in Figure 5. Computation time increases rapidly when nodes are added. With regard to computation time, the increase in complexity clearly outweighs the lowered CLF.

The influence of edges on computation time is more subtle. For an increasing number of nodes, initially the computation time increases rapidly. When the network gets closer to being fully connected, the computation time increases more slowly and can even decrease. This phenomenon can be attributed to an increased probability of the existence of a reliable path between any two nodes. Hence, the benefit associated to

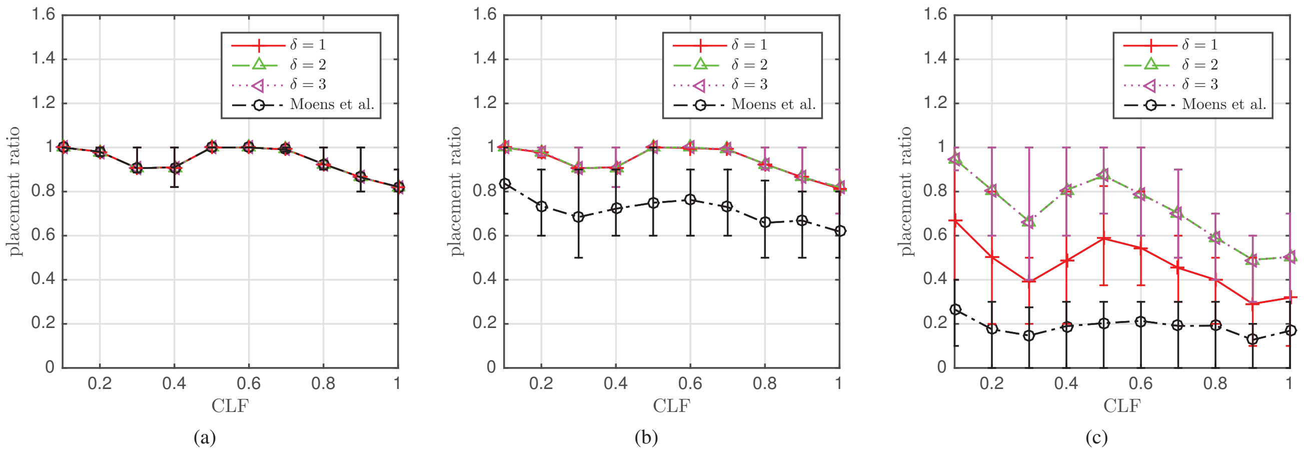

Fig. 3: Placement ratio for a required joint availability of respectively 0% (a), 90% (b) and 99% (c).

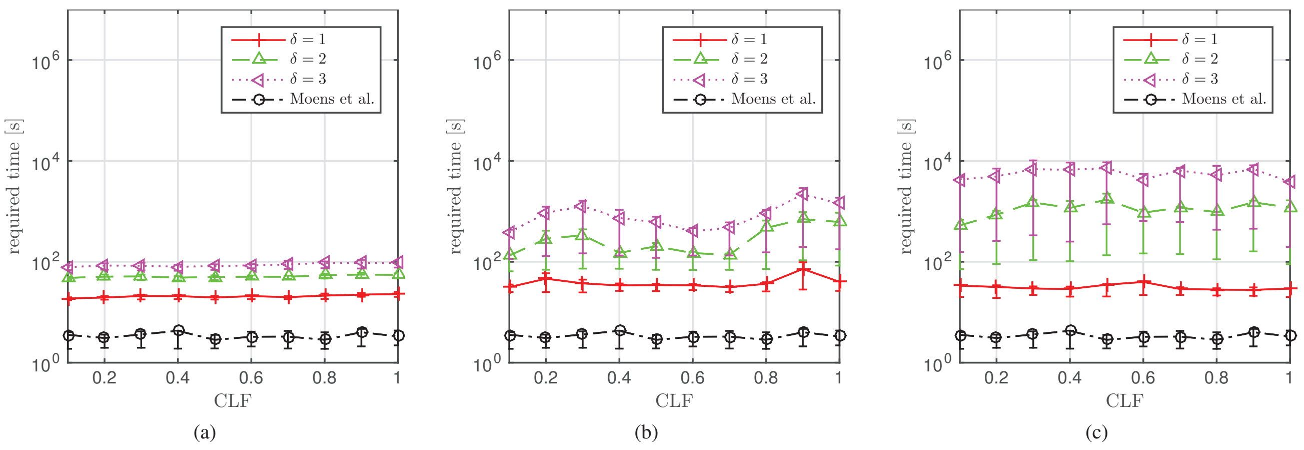

Fig. 4: Computation time for a required joint availability of respectively 0% (a), 90% (b) and 99% (c).

placing two duplicates decreases, which significantly lowers complexity. Moreover it is clear that our optimal approach is infeasible for any realistic dimension of the substrate network.

VI. Conclusion

In this paper we have considered the problem of efficiently placing applications on a physical network prone to both node and link failure. An optimal, availability-aware approach is proposed. To do this we extend the state-of-the art in network-aware application placement with the possibility to define duplication. The results demonstrate that performance of traditional cloud application placement algorithms is severely hampered by the unreliable nature of devices in a heterogeneous cloud environment. Fault-tolerance can drastically improve availability. This can be realized by applying two techniques: first, availability-awareness can significantly improve the placement ratio. Furthermore, redundancy can easily improve the placement ratio by another 50%. As the optimal algorithm does not scale, future work includes development of heuristics. Additionally, future work also includes synchronization amongst duplicates.

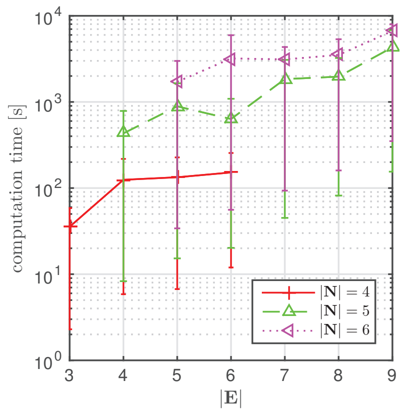

Fig. 5: Computation time for various substrate network dimensions: overall number of undirected edges ($|\mathbf{E}|$) and physical nodes ($|\mathbf{N}|$). Simulations are performed for a required availability level of 90% and at most 2 duplicates.

Acknowledgments

Part of this work has been funded by the iFEST and EMD projects, co-funded by iMinds and IWT.

REFERENCES

[1] R. Yu and T. Watteyne, “White paper: Reliable, low power wireless sensor networks for the internet of things: Making wireless sensors as accessible as web servers,” Dust Networks Product Group, Linear Technology Corp., Tech. Rep., 2013.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the internet of things,” in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

[3] A. Chandra, J. Weissman, and B. Heintz, “Decentralized edge clouds,” *Internet Computing, IEEE*, vol. 17, no. 5, pp. 70–73, Sept 2013.

[4] S. Herker, A. Khan, and X. An, “Survey on survivable virtual network embedding problem and solutions,” in *ICNS 2013. 9th IEEE International Conference on Networking and Services. Proceedings*, March 2013, pp. 99–104.

[5] K. Genova and V. Guliashki, “Linear integer programming methods and approaches–a survey,” 2011.

[6] R. S. Camati, A. Calsavara, and L. Lima Jr, “Solving the virtual machine placement problem as a multiple multidimensional knapsack problem,” in *ICN 2014, The Thirteenth International Conference on Networks*, 2014, pp. 253–260.

[7] J. Xu and J. A. B. Fortes, “Multi-objective virtual machine placement in virtualized data center environments,” in *Proceedings of the 2010 IEEE/ACM Int’L Conference on Green Computing and Communications & Int’L Conference on Cyber, Physical and Social Computing*, ser. GREENCOM-CPSCOM ’10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 179–188.

[8] Y. Ren, J. Suzuki, A. Vasilakos, S. Omura, and K. Oba, “Cielo: An evolutionary game theoretic framework for virtual machine placement in clouds,” in *Future Internet of Things and Cloud (FiCloud), 2014 International Conference on*, Aug 2014, pp. 1–8.

[9] H. Moens, E. Truyen, S. Walraven, W. Joosen, B. Dhoedt, and F. De Turck, “Cost-effective feature placement of customizable multi-tenant applications in the cloud,” *Journal of Network and Systems Management*, vol. 22, no. 4, pp. 517–558, 2014.

[10] H. Moens, B. Hanssens, B. Dhoedt, and F. De Turck, “Hierarchical network-aware placement of service oriented applications in clouds,” in *2014 IEEE/IFIP Network Operations and Management Symposium, Proceedings*, 2014, pp. 1–8.

[11] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, and J. Wang, “Virtual network embedding through topology-aware node ranking,” *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 2, pp. 38–47, 2011.

[12] Y. Zhu and M. Ammar, “Algorithms for assigning substrate network resources to virtual network components,” in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, April 2006, pp. 1–12.

[13] D. Breitgand, Z. Dubitzky, A. Epstein, A. Glikson, and I. Shapira, “Sla-aware resource over-commit in an iaas cloud,” in *Proceedings of the 8th International Conference on Network and Service Management*, ser. CNSM ’12. Laxenburg, Austria, Austria: International Federation for Information Processing, 2013, pp. 73–81.

[14] D. Jayasinghe, C. Pu, T. Eilam, M. Steinder, I. Whally, and E. Snible, “Improving performance and availability of services hosted on iaas clouds with structural constraint-aware virtual machine placement,” in *Services Computing (SCC), 2011 IEEE International Conference on*. IEEE, 2011, pp. 72–79.

[15] W. Wang, H. Chen, and X. Chen, “An availability-aware virtual machine placement approach for dynamic scaling of cloud applications,” in *Ubiquitous Intelligence Computing and 9th International Conference on Autonomic Trusted Computing (UIC/ATC), 2012 9th International Conference on*, Sept 2012, pp. 509–516.

[16] M. Ajtai, N. Alon, J. Bruck, R. Cypher, C.-T. Ho, M. Naor, and E. Szemeredi, “Fault tolerant graphs, perfect hash functions and disjoint paths,” in *Foundations of Computer Science, 1992. Proceedings., 33rd Annual Symposium on*, Oct 1992, pp. 693–702.

[17] W.-L. Yeow, C. Westphal, and U. C. Kozat, “Designing and embedding reliable virtual infrastructures,” *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 2, pp. 57–64, 2011.

[18] A. Broder, “Generating random spanning trees,” in *Foundations of Computer Science, 1989., 30th Annual Symposium on*, Oct 1989, pp. 442–447.