A Model for Managed Elements under Autonomic Cloud Computing Management

Rafael de Souza Mendes*, Rafael Brundo Uriarte†, Carlos Becker Westphall*
*Federal University of Santa Catarina, Florianópolis, Brazil
emails: rafael.mendes@posgrad.ufsc.br, westphal@inf.ufsc.br
†IMT School for Advanced Studies Lucca, Italy
email: rafael.uriarte@imtlucca.it

Abstract—**Due to the scale and dynamism of cloud computing, there is a need for new tools and techniques for its management. This paper proposes an approach to quantitative modelling of cloud components' behaviour, using double weighted Directed Acyclic Multigraphs (DAM) through the different abstraction levels of components. With this formalism, it is possible to analyse load propagation and its effects on the cloud elements from an Anything as a Service (xAAS) perspective. Such model enables the comparison, analysis and simulation of clouds, which assist the cloud management with the evaluation of modifications in the cloud structure and configuration. The existing solutions either do not have mathematical background, which hinders the comparison and production of structural variations in cloud models, or have the mathematical background, but are limited to a specific area (e.g., energy-efficiency), which does not provide support to the dynamic nature of clouds and to the different needs of the managers. In contrast, our model has a formal mathematical background and is generic. Furthermore, we present formalisms and algorithms that support the load propagation and the metrics of *services*, *systems*, *third-parties* providers and resources, such as: *computing*, *storage* and *networking*. To demonstrate the applicability of our solution, we have implemented a software framework for modelling *Infrastructure as a Service*, and conducted numerical experiments with hypothetical loads and behaviours.**

Keywords-Autonomic Cloud Computing; Cloud Computing Management; Simulation; Multigraph.

I. Introduction

The management of pooled resources according to high-level policies is a central requirement of the *as a service* model, as fostered by Cloud Computing (CC). The two major functions in CC management, planning and decision making, are challenging and are still an open issues in the field. In our previous work [1], we have presented a formal model, based on Direct Acyclic Multigraphs (DAM) [2], to model the cloud elements' behaviour regarding loads and evaluations. This formal model intents to reduce the gap between Autonomic CC [3], [4] management and well-established approaches in *decision theory* [5] and *managerial science* [6]. In this regard, was presented a *managed elements model* which make easier the inference of states, actions and consequences. These states, actions and consequences are the bases for planning models and the core of our proposal to fulfil the lack between CC and decision methods. This lack of formal models is highlighted by our previous efforts to develop methods for CC autonomic management: [4][7][8] and formalisms based on Service Level Agreement (SLA) [9].

Currently, the existing solutions which provide CC models can be classified into two main groups: general models, usually represented by simulators; and specific models, devised for a particular field (e.g., energy saving). The former lacks a mathematical formalisation that enables comparisons with variations on the modellings. The latter usually have the formal mathematical background but, since they are specific, they do not support reasoning on different management criteria and encompass only cloud elements related to the target area.

The main obstacle to establish formal general models is to express the conversion of loads from abstract elements (i.e., services or systems) to their concrete components (i.e., physical machines or third-party services). However, such model is mandatory to simulate and analyse qualitatively and quantitatively the CC elements' behaviour, which facilitate the evaluation of managerial decisions, especially if the model deals with abstraction and composition of these elements. The need of this model do express managerial knowledge increases as concept of CC moves away from the concept of infrastructure and Anything as a Service (xAAS) providers build high level cloud structures. To address this gap in the literature, we analyse the domain elements and characteristics to propose the *Cloud Computing Load Propagation* (C_2LP) graph-based model, which is a formal schema to express the *load flow* through the cloud computing components, and the load's impact on them. This schema is required because the system analysis is performed in design time and focus on the behaviour of data when passing through the cloud structures, however, the cloud management requires a view about the behaviour of the structures when the *loads* are passing through them, in runtime. Therefore, we define a *load* as the type and amount of effort to process services' requests in systems or resources.

For example, the (C_2LP) model enables the comparison of different cloud structures, the distinction of load bottlenecks, the expression of conversion of loads units (change in type) between elements, the quantitative analysis of the load propagation and the evaluation of the effects of processing a load on the cloud structure. In more general terms, such solution unifies heterogeneous abstraction levels of managed elements into a single model and can assist the decision-making tasks in processes, such as: *load balance*, *resource allocation*, *scale up/down* and *migrations*. Moreover, simulations performed using our model can be useful to predict the consequences of managerial decisions and external events, as well as the evolution of baseline behaviour, in several abstraction levels.

More specifically, we model the basic components of CC:

(i) *services*; (ii) *systems*; (iii) *resources*, in which systems are deployed, that can be *computing*, *storage* and *networking*; and (iv) *third-party* clouds that deploy services. This taxonomy permits putting together, based on Directed Acyclic Multigraphs, the CC elements on different abstraction levels. It enables the manager to access consolidated graph analytical tools to, e.g., measure the components interdependencies, which is used to improve the availability and resource allocation. In order to demonstrate the applicability and advantages of the C_2LPmodel, we present a use case where our model is used to compare and evaluate different managerial configurations over several quantitative behaviour in load propagation and evaluation.

This article is organised as follows. Section II discusses the existing cloud models, the works that inspired the definition of this model and the background information necessary for the appreciation of this work. In Section III, we present an overview of the model, formalise it, the propagation algorithm, and the evaluation process. Section IV describes the implementation and the analysis performed on a use case. Finally, in Section V, we discuss the limitations and the future directions for the research.

II. Related Work

This section presents the related works that propose models to describe and simulate clouds. We have analysed them from a *cloud provider management perspective*, considering their capacity to: *express general* cloud models, define *components* of the managed cloud instance; *compare* structures; *simulate* behaviours and provide *formal* specifications with mathematical background. Table I summarises model's comparisons and the discussion about the survey is presented as follows.

We grouped the proposals into two classes: *general* and *specific*. General models, such as CloudSim [10], GreenCloud [12], iCanCloud [14], EMUSIM [15] and MDCSim [17], are usually associated with simulators and used to evaluate several criteria at the same time. On the other hand, specific models are commonly associated with particular criterion evaluation, such as performance [18], security [20][21], accounting [22][23] or energy [24].

CloudSim [10] was originally built on top of GridSim [11] and focus on modelling data centres. Its framework is Java based and loads are modelled through a class called "*CloudLet*", or an extension of it. Depiste its popularity, CloudSim does not have a strong mathematical background. This lack of formalism hinders the investigation of data crossing between states and configuration parameter, which limit the exploration of the cloud behaviours. Furthermore, the core classes of CloudSim model data centre elements as: physical machines, virtual machines (VMs), networks and storages; and requires customisations to deal with more abstract elements, e.g., services. Finally, also the comparison of simulation structures is not straightforward with CloudSim.

Kliazovich et al. in [12] presented GreenCloud, an extension of the network simulator NS2 [13] that offers a fine-grained modelling of the energy consumed by the elements of the data centre, such as servers, switches, and links. GreenCloud is a programmatic framework based in C++ and TCL scripts that, despite having the statistic background of NS2, does not have itself an underlying mathematical formalism. It also focuses on the data centres view and need extensions to consider abstract elements as services and systems. Even though the authors provided a comparison between data centre architecture in [12], the model does not favor the comparison of simulation structures.

The simulator iCanCloud, presented in [14], is also a general data centre simulation tool. Based in C++, it has classes as "Hypervisor", "Scheduler" and "VM" in the core class structure, which demonstrates its high level of coupling with infrastructure. Although the authors proposed iCanCloud as "targeted to conduct large experiments", it does not offer native support to compare structural changes between simulations. As the other general simulator, iCanCloud lacks of mathematical formalisms.

EMUSIM [15] is an emulator and simulator that enables the extraction of information from the application behaviour – via emulation – and uses the information to generate a simulation model. The EMUSIM has been built on top of two frameworks: Automated Emulation Framework (AEF) [16] (an emulation testbed for grid applications) and CloudSim [10]. The objective of EMUSIM understand application' behaviour profiles, to produce more accurate simulations and, consequently, to adapt the Quality of Service (QoS) and the budget required for hosting the application in the Cloud. Although EMUSIM partially addresses the lack of capacity to model application of CloudSim, adding higher level modelling layer, it still lacks mathematical formalisms as well as the support to compare simulation structures.

Finally, MDCSim presents a multi-tier data centre simulation platform. However, this multi-tier modelling works with concrete elements, in resource level, as a *front-end tier/web server*, a *mid tier*/application server, and a *back-end tier*/database server. The MDCSim also works with some metrics in a higher abstraction level on specific Java elements as EJBs and Tomcat. This approach still lacks a representation for abstract elements, such as services and systems, where metrics and parameters are related to groups of elements (e.g., availability of a service depending on several elements).

Overall, works proposing general models are data centre focused and have evolved from Grid Computing, which may hinders their usage on service orchestration level and with third-parties cloud infrastructures, where data centre concepts are not applicable. Designers of autonomic management methods require the generation of cloud architectures and behaviours in a combinatorial fashion, in order to test plans, decisions and consequences on a wide number cloud architectures, features that not supported in these models.

In the second group of proposals, that is, frameworks devised for a specific area, in-depth analysis based on robust formalisms are usually provided, such as queue theory [24] [18], probability [20], fuzzy uncertainty [23] and heuristics [22]. However, their models do not fit well in integrated management methods that intend to find optimal configurations considering several criteria of distinct types. Moreover, specific optimisation models are usually sensible to structural changes, having no robustness to support the dynamic nature of the clouds.

Vilaplana et al. in [18] presented a queue theoretic modelling for performance optimisation for scale CC. The model has a strong mathematical background and is able to evaluate jobs and VM scheduling policies using simulations. Nevertheless, this optimisation is dependent on strong assumptions, i.e., that the back-end tier is modelled as an Open Jackson network [19]. The model is focused on evaluation and it is only partially capable of performing simulation. In fact, in the paper the authors employed CloudSim to implement the simulations used in the experiments.

In [20], Silva et al. proposed a model, based on equations to quantify the degree of *exposure* to risk, *deficiency* in risk modelling, and *impact* on an information asset. The model is used to evaluate cloud service providers and has a mathematical background. Although in our previous work [1] we have considered that the ability to generate hypothetical scenarios and evaluate them as a "simulation" feature, we reconsidered and redefined it as "feature not supported" since the model does not support runtime simulations.

Nesmachnow et al. in [22] introduced a broker that resells reserved VMs in IaaS providers as on-demand VMs for the customers. The authors presented a specific model to deal with the Virtual Machine Planning Problem, which was defined as an optimisation problem that maximises the profit of the broker. This problem is mathematically well formed as well as the model that supports the broker representation and the static components. We consider the experiments presented in the paper as simulations that were performed using real data gathered from public reports. However, we considered the simulation feature only as partially covered since the work does not enable runtime simulations.

Decision models for service admission are presented in [23], all with mathematical background and covering fuzzy uncertainty. The proposed models are specific for admission control and explicitly used to perform simulations. On the other hand, the *resource types* used to model different elements in the cloud (e.g., CPU, storage) do not cover the concept of "component". In fact, the model considers the existence of resources, from which services depend, but it just models classes of resources and their economical behaviour related to service admission. Thus, we consider the concept feature "component" only partially covered. Also, the models presented can be compared with respect to revenue and service request acceptance rate, but the general structure of the models lacks comparison parameters.

In [24] an energy-saving task scheduling algorithm is presented, based on the vacation queueing model. The modelling is specific for task scheduling and energy consumption optimisation. The work has a strong mathematical background which enables the comparison of results, but does not have ability to compare the model structure, resulting in a partial coverage for "comparison" criterion. The evaluation of energy consumption in nodes motivated us to define the feature "components" as covered. Finally, the criterion "simulation" was reviewed from the previous analysis in [1] and we consider the model's characterisation as *covered* since the authors used discrete event simulation tool in Matlab, that is equivalent to runtime-like simulators as the CloudSim.

The comparison between the related works is presented schematically in Table I, where: the column "Class" specifies if a work is general or specific; "Formalism" evaluates the mathematical background that supports the models; the column "Components" presents the capacity of a model to express cloud components; the ability to compare structures is depicted in the column "Comparison"; and, "Simulation" expresses the capacity to perform simulations using the models.

Considering the gap in the existing cloud modelling techniques, our proposal intents to model the load propagation and evaluation functions over a graph to obtain expressiveness, whilst keeping the mathematical background and components' details. We opt to model the "load flow" because it is one of the most important information for managerial decisions, such as: load balance, resource allocation, scale up/down and migrations.

TABLE I: COMPARISON BETWEEN RELATED MODELS. ■ REPRESENTS A FEATURE COVERED, □ A PARTIALLY COVERED ONE AND - WHEN THE FEATURE IS NOT SUPPORTED.

Model	Class	Formalism	Components	Comparison	Simulation
CloudSim [10]	General	-	■	-	■
GreenCloud [12]	General	-	■	-	■
iCanCloud [14]	General	-	■	-	■
EMUSIM [15]	General	-	■	-	■
MDCSim [17]	General	-	■	-	■
Chang[24]	Specific	■	□	■	■
Püschel [23]	Specific	■	□	□	■
Nesmachnow [22]	Specific	■	□	■	■
Silva [20]	Specific	■	□	□	-
Vilaplana [18]	Specific	■	□	■	□
C_2LP	General	■	■	■	□

III. MODELLING LOAD FLOW IN CLOUDS

In this section we discuss the main components of cloud structures and propose a formal model based on a directed acyclic multigraph to represent the load flow in clouds. In the Subsection III-A we present the concept of load and its importance for cloud management, as well as, its representation in different abstraction levels. Subsection III-B presents the structural model and its main components. In Subsection III-C, we formally define the data structures to represent *loads*, *configurations*, *states* and *functions*. Finally, Subsection III-D discusses the computational details of the propagation of the loads and the evaluation of the states for each cloud component.

A. Loads and Abstraction Levels

The concept of *load* is central in CC management literature and it is related to the *qualitative* and *quantitative* effort that an element requires to perform a task. However, in CC, it is necessary to manage components related to processing, networking, storage and complex systems, in several abstraction levels. Materially, the loads and the consumers' data that must be transformed, transported and persisted are the same thing. Nevertheless, the system analysts are focused on the *behaviour of data* through the cloud structures, whereas the cloud manager must pay attention to the behaviour of *cloud structures* when the data is passing through them.

In a view based on data centre elements, the loads are traditionally mapped to metrics of processing, networking and storing. This *concrete* view is not complete for CC since the providers can work with elements in other levels of abstraction. Providers in a xAAS fashion can have any type of elements in their structures which must be modelled – from physical resources, till only third-party services as resources of an orchestration system. This heterogeneity in the abstraction levels of managed cloud elements, and their compositional nature (or fractal nature), produces the need to model the load propagation through the abstraction levels.

This load propagation through the technology stack is fundamental to understand how the abstract loads on services' interfaces become concrete loads in the resources. For example, supposing a photography storage service with mobile and web interfaces, the upload of an array of photos can represent a load in the server-side interface (expressed in number of photos), whereas, the same load must be expressed in several loads on (virtual) links, (virtual) processors, and (virtual) storages, not necessarily related to time. In fact, the upload of an array of photos is an abstract load and can be an useful to perform billing metrics, but it can be not useful to measure performance, requiring the detailing to concrete loads, according to the cloud's service implementation. An autonomic manager agent, responsible for planning and decision making in runtime, must understand the quantitative relations into the managed cloud structure to work in real time.

Thus, using a graph to express the dependences between elements in different levels, the abstracter elements (services' interface) must appear in the roots of the graph, the concreter elements (resources) must appear in the leaves, whereas the intermediary elements (systems) orchestrate resources in order to implement the services. These concepts of services interfaces, systems and resources become relative terms which can adapted for any cloud implementation, independent of absolute level of operation regards to the IaaS, PaaS and SaaS taxonomy.

B. Modelling Clouds with C_2LP

In C_2LP, the structural arrangement of cloud elements is based in a *directed acyclic multigraph* (DAM) where the nodes of the graph represent components. To start a horizontal decomposition must be considered the four main types for CC elements:

- *Resources* are the base of any cloud, and can be classified in three *elementary computational function*: as *Computing*, *Storage* and *Networking*; Therefore, these components are always leaf nodes, even when virtualized or based on service orchestration (e.g., a storage block device built on email accounts). The elements with these *computational functions* constitute the sources of computing power into a cloud. The term "computing power" is used here not only for processing, but also for networking and storage, since the CC paradigm effectively offer these last as services, exposing their economical value.
- *Systems* are abstractions of orchestrated resources that implement services. They can be, e.g., applications and platforms. In the model, systems must be directly linked to at least one of each type of resource: computing, storage and networking. Nevertheless, these resources might be replaced by other systems or third-party services. In such cases, the relationship between the system and the element that represents the resource (e.g., another system or the third-party service) must be explicitly defined using stereotypes (virtual computing, virtual networking or virtual storage).
- *Third-Party Services* represent: (i) resources to system components, when the relation is explicitly annotated with the appropriated stereotype, and (ii) entire systems which provide services and abstract the underlying layers (e.g., email services). The latter models, for example, hybrid clouds or composed services.
- *Services* are interfaces between the cloud and the consumers. They must be connected with a respective system that implement them and never are directly linked to resource or third-party services. Services interfaces are the points on which the specification of the consumer's needs (SLAs) are attached. In your model the services interfaces can receive loads from a hypothetical common source (*), that symbolizes the consumer.

Directed edges define to which elements each cloud component can transmit load. Nodes have two main processes: *propagation* of the load; and *evaluation* of the impact of the load in the node itself. Remarkably, the resources components do not propagate load and are the only nodes that actually run the assigned load, while other elements are abstract (e.g., applications, middlewares, platforms and operations systems). Moreover, we consider in the model also the configurations settings of nodes, which impact the propagation and evaluation processes.

Providers offers services and receive requests from consumers. These request represent an economical demand by work, which in providers' assets represent workloads, or just: *loads*. The loads vary according to each cloud component and are changing in quality and quantity along the computing chain that compose the providers' assets. Therefore, each node in the DAM represents a function that convert the input load to output load, from the services (sources) to the resources (sinks). In the resources occurs the work, realizing the load and consuming computing power.

In fact, just low abstraction loads would need to be represented in the model, e.g., supposing an IaaS provider: link, processor and storage. However, the patterns of behaviour in low level loads become chaotic and unmanageable without information about the abstract component that guide the resources usage. Therefore, distributing load propagation functions over a graph is a simple way to represent complex function compositions on a conceptual network. Assuming that the loads flow from the sources (services) to the sinks (resources), and a node must have all incoming loads available to compute the outgoing loads, the propagation must be made in a breadth-first fashion.

Since loads might have different forms, we model these relations enabling multiple edges between nodes, which simplifies the understanding of the model. For example, a service transmits 10 giga FLoating-point Operations Per Second (FLOPS) and 100 giga bytes of data to third-party service. This case is modelled using two edges, one for each type of load to the third-party. In case of change in the structure (e.g., the executor of the loads finds a cheaper storage provider) the model can be adjusted simply by removing the storage edge between these nodes and adding it to this new third-party provider.

When the loads are realized in the resources, they produce several effects which can be measured by the monitoring. For example: resource usage, energy consumption, fails, costs, etc. The modelling of qualitative and quantitative relations between loads and their effects over the resource is a mandatory task to enable managerial planning and decision making. Nevertheless, measurable effects in resources can also signify metrics in system and services. For example, the sum of energy consumed in the processors, network and storage, in order to download a photo of 10GB, means the amount of energy consume do resolve a load of type "download photo" of size "10GB", in service level.

However, is not only the loads which determine the behaviours of the resources, but also the configuration parametrized by the cloud manager, and the accumulated effects from previous loads. On the other hand, non-leaf elements – which the evaluations depend of lower level elements – must consider: incoming loads, the accumulated state (*a priori*) and the state of lower elements (target nodes). This is represented in the model as distinct evaluation functions. In the C_2LP were modelled a set of evaluated functions for leaf nodes, with two inputs, and a set for non-leaf nodes, with three inputs. The both type of functions output a new node state which can contain several sub-evaluations (measures).

The propagation of evaluations is done after the propagation of loads, from bottom to top. This procedure will provide the amount of loads in each element of the model. With the loads and the configurations and accumulated state (*a priori* state) in the resources elements, it is possible to compute the new configurations and accumulated state (the *a posteriori* state). So, in the non-leaf nodes it is possible to compute the *a posteriori* state with its the *a priori* state and the *a posteriori* states of its dependencies (lower level elements). To perform the evaluation of whole graph, from the root nodes, it is necessary to perform a depth-first computing though the graph.

Figure 1 presents the modelling of a scenario, in which a cloud provides two services: an email and Infrastructure-as-a-Service (IaaS). The IaaS is provided by a third-party cloud. The email service instead, employs a system component to represent a software email server (in this case a Postfix). This component uses local computing and networking and storage from a third-party cloud. The relation (edge) between these components is annotated accordingly.

In the proposed scenario, we exemplify the load propagation with a request from consumers to send 2 new emails using the email service. These 2 emails are converted by the service component into 2 loads of type "transaction" and sent for the email server where they are converted into another types of load and propagated to the resources linked to the server.

The evaluation process of this scenario uses different metrics in each node and is marked as "eval:". For example, in the service level, the load of 2 emails was measured in terms financial cost and energy necessary to complete the request.

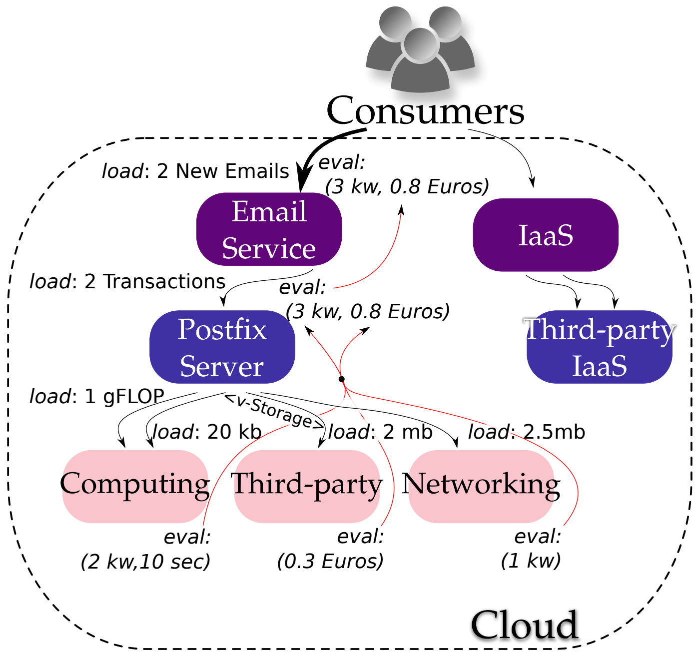

Figure 1: Example of the propagation of loads and the evaluation processes using the C_2LP model.

C. Formalisation of the Model

Formally, in C_2LP model, a cloud C can be expressed as $C = (V, E, \tau^V, \sigma, \Phi, \phi, \Gamma, \gamma, \Gamma', \gamma')$, where:

- V is the set of nodes $V = \{v_1, v_2, \dots, v_n\}$ of the multigraph, such that every item in V represents one element of the cloud and has one respective node-weight w_v, that usually is a vector of values;
- E is the set of directed edges where $E = \{e_1, e_2, \dots, e_m\} | e = (v, v')$, that describes the ability of a source node v to transmit a load to node v', such that each e_m also has a respective edge-weight $w_{v,v'}$;
- $\tau^V : V \rightarrow T^V$ is a bijective function which maps the nodes with the respective type, where the set T^V is the set of types of nodes, such that $T^V = \{'computing', 'storage', 'networking', 'system', 'service', 'third\_party'\}$;
- $\sigma : E_{\{\rightarrow system, \rightarrow third-party\}} \rightarrow \{none, vComputing, vStorage, vNetworking\}$ is a function which maps the edges that have systems and third-party services as target with the respective stereotype, characterising the relation between the source element with the target;
- Φ represents the set of propagation functions, where $\Phi = \{f_1, f_2, \dots, f_v\}$ and ϕ is a bijective function $\phi : V \rightarrow \Phi$ that maps each node for the respective propagation function. Each function in the set Φ is defined as $f_v : \mathbb{N}^n, \mathbb{R}^i \rightarrow \mathbb{R}^o$, where: the set $\mathbb{N}^n$ represents the space where the n-tuple for the configuration is contained; the set $\mathbb{R}^i$ represents the space where the n-tuple of incoming edge-weights is contained; and, $\mathbb{R}^o$ is the space where the n-tuple of the outgoing edge-weights is contained. To simplify the model and the algorithms, we consider that configurations are stored in the node-weight, such that w_v^{conf} represents the configuration part of the node-weight vector.
- Γ is the set of sets that contains the evaluation functions for the leaf nodes, such that there exists one function for each distinct evaluation metric (e.g., energy use, CO2 emission, ...). Then, $\Gamma = \{\Gamma_1, \Gamma_2, \dots \Gamma_k\}$, such that $\Gamma_k = \{g_{n+1}, g_{n+2}, \dots, g_m\}$. Each set Γ_k is related to a leaf node $v \in V_{[leaf]}$ through the bijective function $\gamma : V_{[leaf]} \rightarrow \Gamma$. Every g_{n+m} is stored in a distinct position of the node-weight vector of the respective node – representing *a partial state* of v – such that the full new state can be computed through the expression: $w'_v = (c_1, \dots, c_n, g_{n+1}(c_1, \dots, c_n, w_v^i), g_{n+2}(c_1, \dots, c_n, w_v^i), \dots, g_{n+m}(c_1, \dots, c_n, w_v^i))$, where: $c_1, \dots, c_n$ is the n-tuple with the configuration part of the node-weight w_v; w_v^i is the n-tuple with all incoming edge-weights $w_{*,v}$ of v; and w'_v is the new node-weight (full state) for v. The complete evaluation procedure is detailed in Figure 6;
- Γ' is the set of sets that holds the evaluation functions for non-leaf nodes. Therefore, $\Gamma' = \{\Gamma'_1, \Gamma'_2, \dots, \Gamma'_l\}$, such that each set $\Gamma'_l = \{g'_{n+1}, g'_{n+2}, \dots, g'_m\}$ contains the evaluation functions g'_{n+m}. Every Γ'_l is associated with a non-leaf node v through the bijective

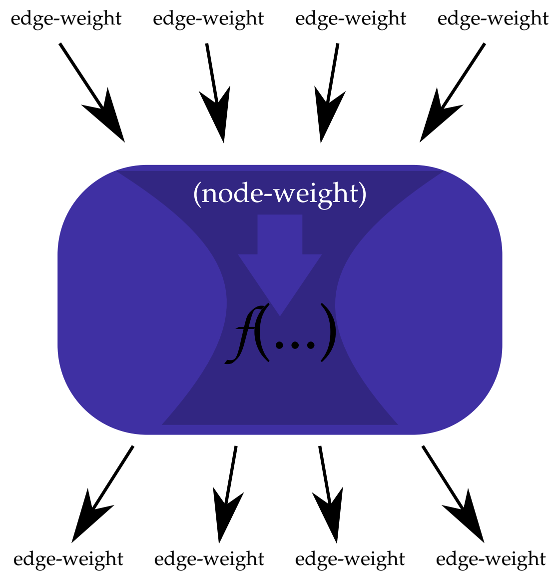

Figure 2: Illustration of load propagation in root or non-leaf nodes.

function $\gamma' : V_{non-leaf} \rightarrow \Gamma'$. Since the result of each function g'_{n+m} is stored in a distinct position of w'_v, it represents a partial state of the respective node v. A new full state of non-leaf nodes can be computed through the expression: $w'_v = (c_1, \ldots, c_n, g'_{n+1}(c_1, \ldots, c_n, w^i_v, w'_{u_v}), g'_{n+2}(c_1, \ldots, c_n, w^i_v, w'_{u_v}), \ldots, g'_{n+m}(c_1, \ldots, c_n, w^i_v, w'_{u_v}))$; where w'_v is the new node-weight of v, $c_1, \ldots, c_n$ is the n-tuple with the configuration part w^{conf}_v of the node-weight, w^i_v is the n-tuple with the incoming edge-weights $e_{*,v}$ of v, and w'_{u_v} is a tuple which puts together all node-weights of the successors of v (see Figure 6 for details).

The main objective of these formalisms is to specify the data structures that support a model validation, the load propagation, and elements evaluations. The details of each procedure concerned with propagation and evaluations are described in Subsection III-D.

D. Details on the Propagation and Evaluation

The load propagation consists in a top-down process that uses the *breadth-first* approach. In a breadth-first algorithm, all the incoming loads are available for a node before the inference of its outgoing loads. In the specific case on C_2LP the algorithm starts from the loads on the services, corresponding to the requests received from consumers. The Figure 2 illustrates the load propagation. The blue oblong represents a non-leaf element that has incoming edges, which the weights represent incoming loads. Alto, there is the node-weight that represents the *a priori* state, that contains the configurations and accumulated states. Both, the incoming loads and node-weight, are used as inputs for the node attached propagation function $f(\ldots)$, that produces a tuple with the output edge-weights.

The propagation process uses a queue with the service nodes (the roots of the graph). Then, a node v is picked from this queue and all its children are placed into the queue. Afterwards, a function $f_v = \phi(v)$ is executed to distribute the load, that is, to define all edge-weights for the outgoing edges of v. This procedure is repeated while the queue is not empty.

procedure BREADHTFIRSTPROPAGATION(C,W^V,W^E) ▷ Requires a cloud model $C = (V, E, \tau^V, \sigma, \Phi, \phi)$, the set of node-weights $W^V | \forall v \in V \wedge \exists! w_v \in W^V$ and the set of edge-weights $W^E | \forall e_{v,v'} \in E \wedge \exists! w_{v,v'} \in W^E$.
  $queue \leftarrow \varnothing$
  $enqueue(*)$
  repeat
    $v \leftarrow dequeue()$
    for each $u \in successorSet(v)$ **do**
      $enqueue(u)$
    end for ▷ enqueues the sucessor of each node
    $f_v \leftarrow \phi(v)$
    $w^{conf}_v \leftarrow configurationPart(w_v)$ ▷ gets the config. part of the node-weight (state).
    $w^i_v \leftarrow (w_{1,v}, w_{2,v}, \ldots, w_{u,v})$ ▷ builds the incoming edge-weights in a tuple w^i_v.
    $w^o_v \leftarrow f_v(w^{conf}_v, w^i_v)$ ▷ w^o_v contains the result of the propagation function.
    for each $w_{v,u} \in w^o_v$ **do**
      $W^E \leftarrow W^E \oplus w_{v,u}$ ▷ replaces the old value of $w_{v,u}$.
    end for▷ assign the values for the outgoing edges of v.
  until $queue \neq \varnothing$
   return W^E
end procedure

Figure 3: Breadth-first algorithm used for the load propagation.

The well defined method is detailed in Figure 3.

When the load is propagated to resources components (leaf nodes), they execute the load. This execution requires power and resources and can be evaluated in several forms. For example, *energy (kw)*, *performance*, *availability*, *accounting*, *security*, *CO_2 emissions* and other cloud specific feature units. This evaluation process takes every function $g_{n+m} \in \Gamma_k$ in order and computes each partial states, storing them into a position of the new node-weight w'_v. A finer description can be defined as: $w'_v = (w^{conf}_v, g_{n+1}(w^{conf}_v, w^i_v), \ldots, g_{n+m}(w^{conf}_v, w^i_v))$, such that w'_v represents the *a posteriori* state for the node v, w^{conf}_v are the configurations (*a priori* state) of v, w^i_v are the incoming edge-weights of v, and $g_{n+m} \in \gamma(v)$ are the evaluation functions associated with the node.

The process of evaluation in leaf nodes is depicted in the Figure 4, where the pink oblong represents a leaf node. In these nodes the edge-weights and the *a priori* node-weight serve as inputs for each function in the vector of evaluation functions, which produce a single value each one. These single values are grouped in a tuple that results in the *a posteriori* node weight.

The evaluations also include the non-leaf nodes since the load also passes through them and it is useful, e.g., to understand the load distribution and to identify bottlenecks. In the case of non-leaf nodes, the evaluation requires also the evaluation results of the bottom nodes. Therefore, this process is performed from the leaves to the roots using a *depth-first* approach.

A non-leaf node receives the tuples $(config, loads, children\_states)$, and evaluates by the processing of all $g'_{n+m} \in \gamma'(v)$ functions. A representation

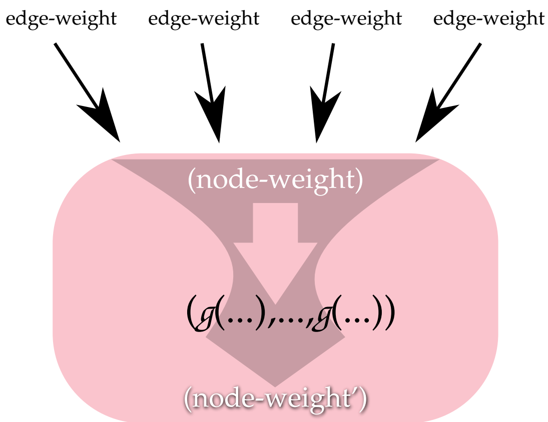

Figure 4: Illustration of evaluations in leaf nodes.

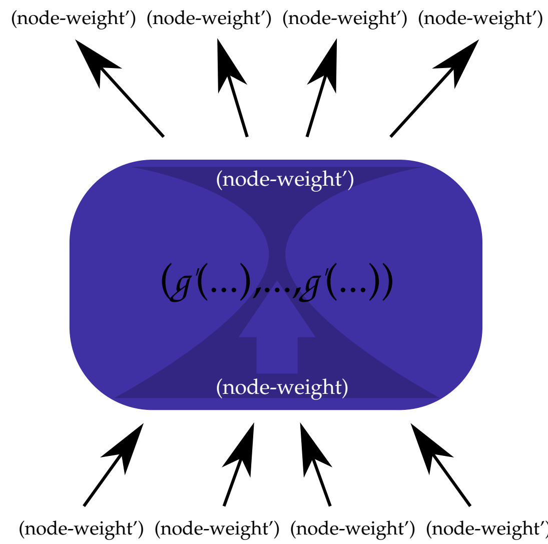

Figure 5: Illustration of evaluations in non-leaf nodes.

of this process can be described as: $w'_v = (w_v^{conf}, g'_{n+1}(w_v^{conf}, w_v^i, w'_{u_v}), \ldots, g'_{n+m}(w_v^{conf}, w_v^i, w'_{u_v}))$, such that w'_v represents the new node-weight (*a posteriori* state) for the node v, w_v^{conf} are the configuration part (*a priori* state) of node-weight into v, w_v^i represent the incoming edge-weights of v, w'_{u_v} are the computed node-weights of the successors of v, and $g'_{n+m} \in \gamma'(v)$ are the evaluation functions associated with the node.

The evaluation in a non-leaf node is depicted in the Figure 5, where the blue oblong represents a non-leaf. In this figure it is possible to observe the *a posteriori* node-weights from the lower level elements being "transmitted" through the edges. The proximity of node-weights with edges do not represent the association between them, but the transmission of one through the other. Into the node is depicted the vector of evaluation functions, which will receive: the *a priori* node-weight of the node itself and the *a posteriori* node-weights from the lower elements; and produce single values which are grouped, in order to compose *a posteriori* node-weight tuple for the node itself. This *a posteriori* node-weight is propagated for the upper elements through the edges. The node-weight in the superior edges have the same value, the computed *a posteriori* node-weight, for all edges. Also, the arrows do not represent the direction of the edges, but the information flow.

The complete evaluation process is detailed in Figure 6, where a stack is used to perform a depth-first computation. The first non-visited child of a current node is placed into the stack and will be used as current node. When all children of a node are evaluated, then the node is evaluated. If the node is a leaf node the g functions are used to compute the evaluations, otherwise, the g' functions are used instead.

These mathematical structures and algorithms provide a general framework for modelling and evaluation of clouds' elements behaviour in different abstraction levels. They can express and compute how service level loads are decomposed and converted, through the systems, till become resource level loads. In resource level, on concrete elements, the loads can be evaluated according to performance, availability and other objective metrics. At end, the same structures and algorithms can be used to compute objective metrics for abstract elements. The whole model serves to simulate and compare the impact of configuration's changes in any point of the cloud, supporting the managerial decision making.

IV. Experiments and Results

This section presents numerical experiments with the C_2LP model, based on a *service* modelling. These experiments serve to: (i) *test* the applicability of the model; (ii) illustrate the modelling with our formalism with an example; and (iii) *demonstrate* the model capacity to *generate quantitative behaviours* to manage loads, combining variations of propagation and evaluation functions.

To perform these experiments, we have implemented a use case using our model. This use case exemplifies the model's usage and serves to test its feasibility. The example of model's usage was made using hypothetical functions, since its objective is to prove the generation of simulations, the propagation and the evaluation. Nevertheless, our model can be used for modelling real-world clouds, provided that the propagation and evaluation functions are adjusted to the cloud instance.

As use case, we defined an *IaaS service* where consumers perform five operation: *deploy VM*, *undeploy VM*, *start VM*, *stop VM*, and *execute* tasks. To meet the demand for these services, we designed a hypothetical cloud infrastructure with which is possible to generate quantitative scenarios of propagation and evaluation – in a combinatorial fashion. Using this hypothetical infrastructure, we have tested some managerial configurations related to load distribution over the cloud elements, in order to evaluate the average utility for all quantitative scenarios. At the end, the configurations which achieve the best average utility for all quantitative scenarios were highlighted, depicting the ability of the model to simulate configuration consequences for the purpose of selecting configurations.

A. Use Case Modelling

To deal with the consumers' loads (deploy, undeploy, start, stop and execute) at service abstraction level, the infrastructure manages: the *service interface*; systems, such as *load balancers*, *cloud managers* and *cloud platforms*; and resources, such as *servers*, *storages* and *physical networks*. All operations invoked by consumers represent an incoming

```
procedure DEPTHFIRSTEVALUATION(C,W^V,W^E)    ▷ The same input described in Figure 3.
    β ← ∅                      ▷ initializes the set of visited nodes.
    stack ← ∅                  ▷ initializes the stack.
    push(*)                    ▷ starts from the hypothetical node.
    while stack ≠ ∅ do
        v ← peek()             ▷ gets a node without to remove it.
        for each u ∈ successorSet(v) do
            if u ∉ β then
                push(u)
                continue while
            end if
        end for ▷ if the for loop ends, all successors have been evaluated.
        w^conf ← configurationPart(w_v)    ▷ gets the config. part for v.
        w_v^i ← (w_1, w_2, ..., w_{u,v})   ▷ builds the n-tuple with the incomings of v.
        if isLeaf(v) then
            w'_v ← (w_v^conf, g_{n+1}(w_v^conf, w_v^i), ..., g_{n+m}(w_v^conf, w_v^i)), ∀g_{n+m} ∈ γ(v)
                ▷ computes the partial states and builds the new node-weight.
        else
            w'_{u_v} ← (w'_{u_1}, w'_{u_2}, ..., w'_{u_o})
                ▷ builds the computed node-weights for all u|∃e_{v,u} ∈ E.
            w'_v ← (w_v^conf, g'_{n+1}(w_v^conf, w_v^i, w'_{u_v}), ..., g'_{n+m}(w_v^conf, w_v^i, w'_{u_v})), ∀g'_{n+m} ∈ γ'(v)
                ▷ computes the partial states and builds the new node-weight.
        end if
        W^V ← W^V ⊕ w'_v        ▷ replaces the old state of v into the node-weights.
        if v ∉ β then
            β ← β ∪ v
        end if ▷ puts v in the visited set if it is not there.
        v ← pop()               ▷ gets and removes v from the stack.
    end while
    return W^V
end procedure
```

Figure 6: Depth-first algorithm to evaluate in specific metrics the impact of the load in each node.

load on the service interface, which is propagated to resources. In the resources the loads are evaluated to provide measures about *performance*, *availability*, *accounting*, *security* and *CO_2 emissions*. Once computed these measures for resource level elements it is possible to compute they also for systems and, at the end, for the service interfaces, getting service level measures.

The modelling of the use case was devised considering 21 components: 1 service, 9 systems, and 11 resources. The services represent the interface with customers. In this use case, the systems are: a *load balancer*; two *cloud manager* systems; and six *cloud platforms*. Also, between the resources there are: 8 physical computing servers (6 work servers and 2 managerial), 2 storages (1 work storage and 1 managerial), and 1 physical network. A detailed list of components is presented in Appendix I.

Regarding the edges and loads, each consumer's operation is modelled as an incoming edge in a *service interface node* – with the respective loads in the edge-weights. The service node forwards the loads for a *load balancer* system, where the propagation function decides to which *cloud manager* the load will be sent, whereas the *manager servers*, the *manager storage* and the *physical network* receive the loads by its operation. In the cloud mangers, the propagation function must decide to which *cloud platform* the loads will be sent and, at the same time, generate loads for the managerial resources. The cloud platform system effectively converts its loads into simple resource loads when uses the *work server*, *work storage* and *physical network*. The complete relation of load propagation paths is presented in Appendix I, where an element at the left side of an arrow can propagate loads for an element at the right. Furthermore, a graphical representation of these tables, which depicts the graph as a whole, is also presented in Appendix I.

Besides the node and the edges, the use case model required the definition of: • 4 types of propagation functions – one for the service and tree for each type of system; • 6 types of leaf evaluation functions – two specific performance evaluations, one for computing resources and another for storage and networking; plus, four common evaluation functions (availability, accounting, security and CO_2 emissions) for each type of resource; • 5 types of non-leaf evaluations functions.

We have modelled the possible combinations to distribute the loads $\{$1-deployVM, 2-undeployVM, 3-startVM, 4-stopVM, 5-compute$\}$ as a partition set problem [25], resulting in 52 distinct possibilities of load propagation. Also, we introduced 2 possible configurations into each evaluation function for leaf nodes. These configurations are related to the choice of constants into the function. For example, the performance of a computing resource depends on its capacity, that can be: $a = 50GFLOPs$ or $b = 70GFLOPs$. Considering 5 distinct evaluation functions over 11 leaf nodes, we have got $(2^5)^{11} = 2^{55}$ possible distinct configurations to test.

B. Evaluations

The numerical experiments were performed running the propagation procedure, followed by the evaluation of every simulation. For each possible propagation, we tested and summarised the 2^{55} configurations for evaluation functions. Then, we analysed the average time (p, in seconds), average availability (av, in %), average accounting (ac, in currency units), average security (s, in % of risk of data exposition), and average of CO_2 emissions (c, in grammes). Each value was normalised according to the average for all propagations, tested and summarised in a global utility function, described in (1) – where the overlined variables represent the normalised values.

Such results can be used by cloud managers to choose the best scenario according to the priorities of the policy or to provide as input for a decision-making process, such as Markov Chains.

$$u = -(\overline{av} + \overline{s} - (\overline{p} + \overline{ac} + \overline{c})) \quad (1)$$

The four best results of the fifty two numerical experiments are presented in Table II in ascending order. The configuration that achieves the best *average utility* is highlighted in bold. The *code* line in the table represents the propagation configuration, whereas the other lines contain the values obtained for each distinct evaluation type. The last row present the average utility

TABLE II: SUMMARY OF AVERAGE EVALUATIONS FOR EACH CONFIGURATION.

Criteria	Configuration			
Code	11221	11231	11232	**11212**
Time	180.59976	180.5999	180.60004	**180.59991**
Availability	0.9979606	0.99795955	0.9979587	**0.99795926**
Accounting	78.69924	78.69926	78.699234	**78.699265**
Security	0.9979606	0.99795955	0.9979587	**0.99795926**
Emissions	82848.31	82848.14	82848.51	**82848.74**
Utility	1.0526400204	1.0526410547	1.0526477776	**1.0526491889**

defined in Equation 1. To represent configuration we have adopted a set partition notation to express the propagation paths, such that each position in the code represents a type of load: 1-*deploy*, 2-*undeploy*, 3-*start*, 4-*stop*, and 5-*compute*. Considering that at leaves of the propagation graph there are 6 cloud platforms, a code 11212 indicates that the loads of type 1,2 and 4 were allocated on cloud platform 1, whereas the loads 3 and 5 ware allocated in the cloud platform 2.

These experiments present evidences that our model works as an engine to simulate and measure the impact of the propagation of loads through the several elements in the cloud. With the distribution of simple functions on a graph, we have demonstrated the capacity to compute a model that is rather complex,when treated purely with function composition and arithmetic. These experiments also shows that the metrics of behaviour can be simulated with the combinatorial representation of the parameters settings which generated the behaviour.

The breadth-first algorithm ensures that the nodes compute all loads before estimating their outputs. On the other hand, the model and the depth-first algorithm ensure that the computed measures generated by the actual resource consumption, which occurs in the leaves of the modelled cloud, can be composed. The loads are converted into different types (and units), according to the elements and specified functions. Also, the adjusts in the parameters in the node-weight allow the testing of several computed loads and measures, in different configuration scenarios. These parameters can be treated with combinatorics instead of programmatic simulators, since the total set of possible configurations becomes a well defined combinatorial problem.

V. CONCLUSION AND FUTURE WORKS

Several solutions have been proposed to model clouds. However, to the best of our knowledge, none is general and has mathematical formalism at the same time, which are essential characteristics for evaluation of decision making and autonomic management.

In this study, we have presented an approach with these characteristics to model clouds based in *Directed Acyclic Multigraph*, which has the flexibility of general models and the formalism of the specifics. Therefore, C_2LP is a flexible well-formed modelling tool to express flows of loads through the cloud components. This model supports the specification of elements in distinct abstraction levels, the generation of combinatorial variations in a use case modelling and the evaluation of the consequences of different configuration in the load propagation.

We developed a simulation software tool for the modelling of IaaS services and demonstrated the applicability of our approach through a use case. In this use case, we simulated several graph network theoretic analysis, evaluated and compared different configurations and, as a result, supplied the cloud managers with a numeric comparison of cost and benefits of each configuration. These experiments, demonstrated that this tools provides an essential support for the management of cloud.

In the future works we intent to develop a description language to specify the rules of association between cloud elements in order to compose de graph. Yet, we intent to study the fractal phenomena in cloud structures, in order to improve the managerial view about the relation between abstract and concrete elements, and the model's granularity. Also, is our desire to investigate how the different models – among the possible aggregations of metrics and parameters – impact the planning and decision making in management of cloud at runtime. At last, we intent to improve the C2LP adding order relations between the states, attached to nodes, in order to enable the model to encompass policies and SLAs.

ACKNOWLEDGEMENT

The present work was done with the support of CNPq agency, through the program Ciência sem Fronteiras (CsF), and the company Eletrosul Centrais Elétricas S.A. – in Brazil. The authors also would like to thank professor Rocco De Nicola and the group of SysMA research unit in Institute for advanced studies Lucca (IMT).

REFERENCES

[1] Rafael de S. Mendes, Rafael B. Uriarte, and Carlos B. Westphall, "C2LP: Modelling Load Propagation and Evaluation through the Cloud Components," In ICN 2016, The Fifteenth International Conference on Network, IARIA XPS Press, 2016, pages 28–36.

[2] Sekharipuram S. Ravi and Sandeep K. Shukla, "Fundamental Problems in Computing: Essays in Honor of Professor Daniel J. Rosenkrantz," Springer Netherlands, 2009.

[3] Rajkumar Buyya, Rodrigo N. Calheiros, and Xiaorong Li, "Autonomic cloud computing: Open challenges and architectural elements," In *Emerging Applications of Information Technology (EAIT), 2012 Third International Conference on*. IEEE, 2012, pp. 3–10.

[4] Rafael de S. Mendes et al., "Decision-theoretic planning for cloud computing," In ICN 2014, The Thirteenth International Conference on Networks, Iaria, vol. 7, no. 3 & 4, 2014, pages 191–197.

[5] Itzhak Gilboa, "Theory of Decision under Uncertainty," Cambridge University Press, 2009.

[6] Cliff Ragsdale, "Modeling & Decision Analysis," Thomson, 2008.

[7] Alexandre A. Flores, Rafael de S. Mendes, Gabriel B. Bräscher, Carlos B. Westphall, and Maria E. Villareal, "Decision-theoretic model to support autonomic cloud computing," In ICN 2015, The Fourteenth International Conference on Networks, Iaria, vol. 8, no. 1 & 2, 2015, pages 218–223.

[8] Rafael B. Uriarte, "Supporting Autonomic Management of Clouds: Service-Level-Agreement, Cloud Monitoring and Similarity Learning," PhD thesis, IMT Lucca, 2015.

[9] Rafael B. Uriarte, Francesco Tiezzi, and Rocco De Nicola, "SLAC: A formal service-level-agreement language for cloud computing," In Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, IEEE Computer Society, 2014, pages 419–426.

[10] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, Csar A. F. De Rose, and Rajkumar Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," Software: Practice and Experience, Wiley Online Library, vol. 41, no. 1, 2011, pages 23–50.

[11] Rajkumar Buyya and Manzur Murshed, "Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," Concurrency and computation: practice and experience, Wiley Online Library, vol. 14.no. 1315, 2002, pages 1175–1220.

[12] Dzmitry Kliazovich, Pascal Bouvry, and Samee Ullah Khan, "GreenCloud: a packet-level simulator of energy-aware cloud computing data centers," The Journal of Supercomputing, Springer, 2012, page 1263–1283.

[13] Teerawat Issariyakul and Ekram Hossain, "Introduction to network simulator NS2," Springer Science & Business Media, 2011.

[14] Alberto Núñez et al., "iCanCloud: A flexible and scalable cloud infrastructure simulator," Journal of Grid Computing, Springer, vol. 10, no. 1, 2012, pages 185–209.

[15] Rodrigo N. Calheiros, Marco A.S. Netto, César A.F. De Rose, and Rajkumar Buyya, "EMUSIM: an integrated emulation and simulation environment for modeling, evaluation, and validation of performance of cloud computing applications," Software: Practice and Experience, Wiley Online Library, vol. 43, no. 5, 2013, pages 595–612.

[16] Rodrigo N. Calheiros, Rajkumar Buyya, and Csar A. F. De Rose, "Building an automated and self-configurable emulation testbed for grid applications," Software: Practice and Experience, Wiley Online Library, vol. 40, no. 5, 2010, pages 405–429.

[17] Seung-Hwan Lim, Bikash Sharma, Gunwoo Nam, Eun Kyoung Kim, and Chita R. Das, "MDCSim: A multi-tier data center simulation, platform," In Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on, IEEE, 2009, pages 1–9.

[18] Jordi Vilaplana, Francesc Solsona, and Ivan Teixidó, "A performance model for scalable cloud computing," In 13th Australasian Symposium on Parallel and Distributed Computing (AusPDC 2015), ACS, vol. 163, 2015, pages 51–60.

[19] James R. Jackson "Networks of Waiting Lines," Operations Research, vol 5, no. 4, 1957, pages 518–521.

[20] Paulo F. Silva, Carlos B. Westphall, and Carla M. Westphall, "Model for cloud computing risk analysis," ICN 2015, Iaria, vol. 8, no. 1 & 2, 2015, page 152.

[21] Nada Ahmed and Ajith Abraham, "Modeling cloud computing risk assessment using machine learning," In Afro-European Conference for Industrial Advancement, Springer, 2015, pages 315–325.

[22] Sergio Nesmachnow, Santiago Iturriaga, and Bernabe Dorronsoro, "Efficient heuristics for profit optimization of virtual cloud brokers," Computational Intelligence Magazine, IEEE, vol. 10, no. 1, 2015, pages 33–43.

[23] Tim Püschel, Guido Schryen, Diana Hristova, and Dirk Neumann, "Revenue management for cloud computing providers: Decision models for service admission control under non-probabilistic uncertainty," European Journal of Operational Research, Elsevier, vol. 244, no. 2, 2015, pages 637–647.

[24] Chunling Cheng, Jun Li, and Ying Wang, "An energy-saving task scheduling strategy based on vacation queuing theory in cloud computing," Tsinghua Science and Technology, IEEE, vol. 20, no. 1, 2015, pages 28–39.

[25] Toufik Mansour, "Combinatorics of Set Partitions," CRC Press, 2012.

APPENDIX I: IMPLEMENTATION DETAILS

TABLE III: THE CLOUD ELEMENTS – NODES OF THE GRAPH.

CS - computing service	**CP21** - platform 21	**WS12** - work server 12
LB - load balancer	**CP22** - platform 22	**WS13** - work server 13
CM1 - cloud manager 1	**CP23** - platform 23	**WS21** - work server 21
CM2 - cloud manager 2	**MS1** - manager server 1	**WS22** - work server 22
CP11 - platform 11	**MS2** - manager server 2	**WS23** - work server 23
CP12 - platform 12	**MSTO** - manager storage	**WSTO** - work storage
CP13 - platform 13	**WS11** - work server 11	**PN** - physical network

TABLE IV: THE LOAD PROPAGATION RELATIONS – EDGES OF THE GRAPH.

$\xrightarrow{5}$CS	CM1$\xrightarrow{5}$CP11	CP11 $\rightarrow$ WS11	CP21 $\rightarrow$ PN
CS$\xrightarrow{5}$LB	CM1$\xrightarrow{5}$CP12	CP11$\rightarrow$PN	CP21$\rightarrow$WSTO
LB$\xrightarrow{5}$CM1	CM1$\xrightarrow{5}$CP13	CP11$\rightarrow$WSTO	CP22$\rightarrow$W22
LB$\xrightarrow{5}$CM2	CM1 $\rightarrow$ PN	CP12$\rightarrow$WS12	CP22 $\rightarrow$ PN
LB $\rightarrow$ MS1	CM2 $\rightarrow$ MS2	CP12 $\rightarrow$ PN	CP22 $\rightarrow$ WSTO
LB $\rightarrow$ MS2	CM2 $\rightarrow$ MSTO	CP12 $\rightarrow$ WSTO	CP23 $\rightarrow$ W23
LB $\rightarrow$ WSTO	CM2 $\xrightarrow{5}$ CP21	CP13$\rightarrow$ W13	CP23 $\rightarrow$ PN
LB $\rightarrow$ PN	CM2 $\xrightarrow{5}$ CP22	CP13$\rightarrow$ PN	CP23 $\rightarrow$ WSTO
CM1 $\rightarrow$ MS1	CM2 $\xrightarrow{5}$ CP23	CP13 $\rightarrow$ WSTO	
CM1 $\rightarrow$ MSTO	CM2 $\rightarrow$ PN	CP21 $\rightarrow$ W21	

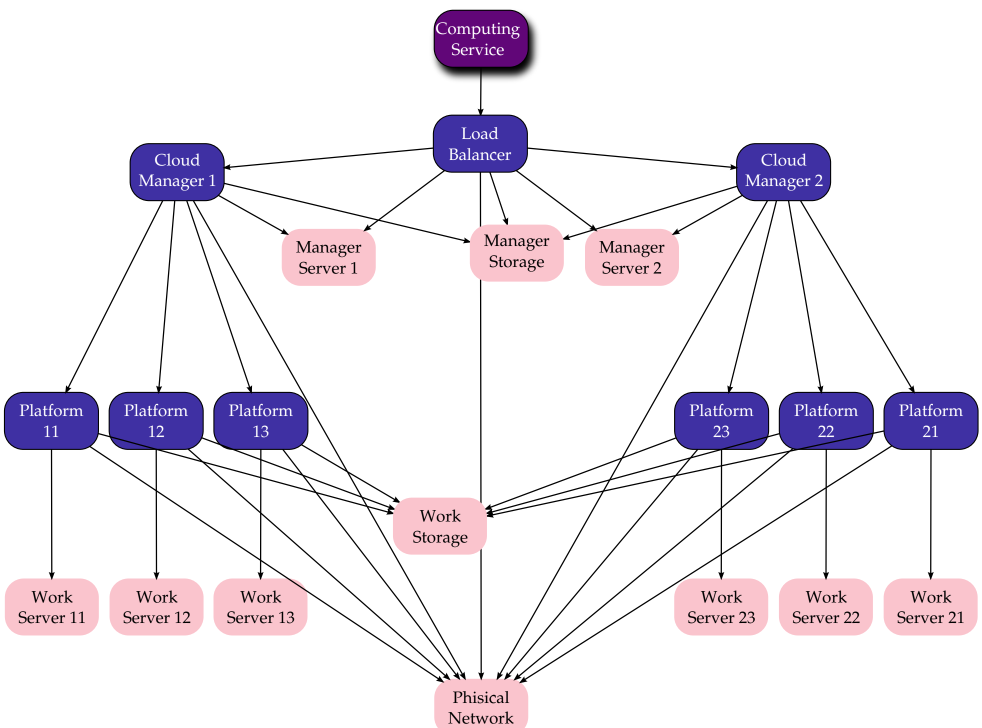

Figure 7: Graphical representation of structural arrangement for the modelling use case.

TABLE V: PROPAGATION FUNCTIONS.

Types	Declarations	Definitions
service	$(w_1, \cdots, w_5) \xmapsto{f^{CS}} (w'_1, \cdots, w'_5)$.	w_n is the weight for $\xrightarrow{n} CS)$. w'_n is the weight for $(CS \xrightarrow{n} LB)$. $w'_n = w_n \vert \forall w'_n \in f^{CS}$.
balancer	$(c_1, \ldots, c_5, w_1, \cdots, w_5) \xmapsto{f^{LB}}$. $(w'_1, \ldots, w'_{14})$	$c_n \in \{CM1, CM2\}$, are the configurations which represent the targets of each load $w_n \vert 1 \leqslant n \leqslant 5$. $$w'_n = \left\{ \begin{array}{ll} w_n & \text{if } c_n = CM1 \\ 0 & \text{otherwise} \end{array} \right. \Bigg\vert 1 \leqslant n \leqslant 5$$. $$w'_{n+5} = \left\{ \begin{array}{ll} w_n & \text{if } c_n = CM2 \\ 0 & \text{otherwise} \end{array} \right. \Bigg\vert 1 \leqslant n \leqslant 5$$. $w'_{1 \geqslant n \geqslant 5}$, are the weights in the edges LB$\xrightarrow{5}$CM1. $w'_{6 \geqslant n \geqslant 10}$, are the weights in the edges LB$\xrightarrow{5}$CM2. $w'_{11} = 1Gflop$, is the a constant computing load in LB$\rightarrow$MS1. $w'_{12} = 1Gflop$, is the a constant computing load in LB$\rightarrow$MS2. $w'_{13} = 50GB$, is the a constant storage load in LB$\rightarrow$MSTO. $w'_{14} = w_1 + 40$, is the load over LB$\rightarrow$PN, such that w_1 is the VM image size in *GB*, comes from *deploy VM* operation, and 40 is a constant value in *GB* for the another operations.
cloud manager	$(c_1, \ldots, c_5, w_1, \cdots, w_5) \xmapsto{f^{CMn}}$ $(w'_1, \ldots, w'_{18})$	$c_n \in \{CPm1, CPm2, CPm3\}$, are the configurations which represent the targets of each load $w_n \vert 1 \leqslant n \leqslant 5$. $$w'_n = \left\{ \begin{array}{ll} w_n & \text{if } c_n = CPm1 \\ 0 & \text{otherwise} \end{array} \right. \Bigg\vert 1 \leqslant n \leqslant 5$$. $$w'_{n+5} = \left\{ \begin{array}{ll} w_n & \text{if } c_n = CPm2 \\ 0 & \text{otherwise} \end{array} \right. \Bigg\vert 1 \leqslant n \leqslant 5$$. $$w'_{n+10} = \left\{ \begin{array}{ll} w_n & \text{if } c_n = CPm3 \\ 0 & \text{otherwise} \end{array} \right. \Bigg\vert 1 \leqslant n \leqslant 5$$. $w'_{16} = 1Gflop$, is the a constant computing load in CMn$\rightarrow$MSn. $w'_{17} = 50GB$, is the a constant storage load in CMn$\rightarrow$MSTO. $w'_{18} = w_1 + 40$, is the load over CMn$\rightarrow$PN, such that w_1 is the VM image size in *GB*, comes from *deploy VM* operation, and 40 is a constant value in *GB* for the another operations.
cloud platform	$(w_1, \cdots, w_5) \xmapsto{f^{CPnn}}$ (w'_1, w'_2, w'_3).	$w_1, \cdots, w_5$, are the main loads come from the service, associatively, w_1 – deploy VM, w_2 – undeploy VM, w_3 – start VM, w_4 – stop VM, and w_5 – compute tasks. w'_1, w'_2 and w'_3 are, respectively, the edge-weight for the arcs CPnn$\rightarrow$WSnn, CPnn$\rightarrow$WSTO and CPnn$\rightarrow$PN, where: $w'_1 = w_1 - w_2 + w_3 - w_4 + w_5$; $w'_2 = w_1 - w_2 + 1MB$; $w'_3 = w_1 + w_3 - w_4 + 1MB$.

TABLE VI: EVALUATION FUNCTIONS FOR LEAF NODES.

Types	Functions
computing specific functions	*performance* (*duration*): $d(load) = \frac{load}{capacity}$, where $load$ is expressed in GFlop, $capacity$ is a constant of 70GFLOPs and d is the total time to resolve the load. *energy increment* (kWh): $energy_{increment}(load)$ here is considered a linear function which returns the amount of energy necessary to process the load above the average consumption of standby state. For computing have been considered $0.001kW$ per GFLOP.
storage and network specific functions	*performance* (*duration*): $d(load) = \frac{load}{capacity}$, where $load$ is expressed in GByte, $capacity$ is a constant of 1GBps and d is the total time to resolve the load. For the networking resources this concept is intuitively associated with the network throughput, however, for storage is necessary to explain that the performance refers to throughput of the data bus. *energy increment* (kW): $energy_{increment}(load)$ for data transmission is assumed as linear, and was here considered $0.001kW$ per GB transferred.
common functions	*availability*: $av(load) = 1 - p_{fault}(d(load))$, where p_{fault} is the probability which a fault occurs during the load processing. Here will be considered a linear naive probability, such that $p_{fault}(d) = d \times 0.01$. *accounting*: $ac(load) = price_{energy} \times energy_{total}$, where $price_{energy}$ is a constant of $0.38US\$/kW$ or $0.58US\$/kW$, depending on node configuration; and $energy_{total} = energy_{increment}(load) + energy_{average}(d(load))$, such that $energy_{average}(d(load)) = d(load) \times 0.1kW$ is the shared energy spent by the cloud by time slot, and $energy_{increment}(load)$ is the increment of energy result of resource usage. *security* (*risk of data exposition*): $s(load) = 1 - p_{exposure}(load)$, where $p_{exposure}(load)$ is the probability that the load processing results in data exposure and $s(load)$ is the trustability of the operation. The $p_{exposure}(load)$ is calculated as 0.001 for each second of operation. *CO_2 emission*: $c = energy_{total} \times 400$, where $energy_{total}$ was defined in the accounting evaluation function and 400 is a constant which represents the grammes of CO_2 per kW.

TABLE VII: EVALUATION FUNCTIONS FOR NON-LEAF NODES.

Types	Declarations	Definitions
performance	maximum duration of loads sent for successor nodes.	$p_v(w_1, \ldots, w_5, w'_1, \ldots, w'_n) = max(w'_1[p], \ldots, w'_n[p])$, where p_v represents the total time to process the incoming loads, and $w'_n[p]$ represents the specific part of in the node-weight of n successor nodes, regards to the duration to process the loads sent by the node v.
availability	the product of the availability of successor nodes according to the sent loads.	$av_v(w_1, \ldots, w_5, w'_1, \ldots, w'_n) = \prod w'_n[av]$, where av_v represents the total availability of a node v according its dependencies, and $w'_n[av]$ represents the availability part in node-weights of the successors of v, related to the loads sent.
accounting	the sum of costs relative to the sent loads for successor nodes.	$ac_v(w_1, \ldots, w_5, w'_1, \ldots, w'_n) = \sum w'_n[ac]$, where ac_v is the total cost related to v and regards to the loads processed in the successors, and $w'_n[ac]$ is the accounting part of the successors' node-weight.
security	the product of security (regards to data exposition) of successor nodes according to the sent loads.	$s_v(w_1, \ldots, w_5, w'_1, \ldots, w'_n) = \prod w'_n[s]$, where s_v represents the total security measure of a node v, and $w'_n[s]$ represents the security measure part in node-weights of the successors of v, related to the loads sent.
CO_2 emission	the sum of total emissions relative to the loads sent to the successor nodes.	$c_v(w_1, \ldots, w_5, w'_1, \ldots, w'_n) = \sum w'_n[ac]$, where c_v is the total CO_2 emission associated with a node v, and $w'_n[ac]$ is the node-weight part associated with the emissions caused by the loads sent from v.