# A Topology-Aware Adaptive Deployment Framework for Elastic Applications

Matthias Keller*, Manuel Peuster*, Christoph Robbert*, Holger Karl*
*University of Paderborn – Compute Network Research Group
D-33098 Paderborn – Email: mkeller@upb.de, hkarl@ieee.org

***Abstract*—In Distributed Cloud Computing, applications are deployed over thousands of geographically distributed cloud sites. This new deployment approach promises not only improved application's quality of service but enables deploying network-critical applications otherwise not possible. A previously settled, static allocation of enough resources at each site is expensive. Adapting resource allocations during application lifetime could dramatically reduce expenses. They are triggered by complex algorithms as a reaction of changes in measured performance data. However, such adaptation algorithms and their performance data depends on specific application's requirements, constrains, and on optimization goals. Some interactive applications need a low packet round trip time. Other streaming applications need a high data rate to the user. Besides these individual characteristics of an adaptation, similarities exists in the management of such a geographical distribution: Applications and their components are deployed within Virtual Machines. The components have to find and communicate with each other.**

**Motivated by this, we present a framework taking care of necessary common functionality while been highly customizable to support a wide range of adaptation. Additionally, integrated adaptations can utilize combined application- and infrastructure-level data and are also able to reconfigure the application and the infrastructure. Finally, a steering-system supports state-ful and multi-tier applications to be deployed in an elastic and dynamic way.**

***Keywords*—*Elastic Applications Deployment; Network Topology; Adaptive Application Deployment; Cloud Computing; Geographically Distributed; Architecture***

## I. Introduction

Cloud applications are usually deployed at one site. Such a deployment has benefits, like simple provisioning, one contract partner, etc. However, if an application provider wants a higher Quality of Service (QoS), they often need to allocate resources at multiple sites. Application providers like Google already operate their applications at few sites around the world.

In the next years, more cloud providers will appear and with that, many, even thousands of cloud sites will be available. In line with this trend are Carriers, Internet Service Providers, and Mobile Operator [13], [21], [36], [37] who are already investigating the opportunities of "in-network" cloud resources. This trend is usually referred to as Distributed, Carrier, or Networked Clouds.

### A. Scenario

*Geographically Distributed:* New opportunities arise from distributed deployments across thousands of possible cloud sites:

- Improved network-related access properties (QoS) because of shorter distances between customers and application servers.
- Improved stability because of redundant application servers at independent sites.
- Opportunities to shape or redirect traffic by deciding where to serve the demand.
- Local advantages of cloud sites, such as price, available capacity, or energy efficiency.

On the other hand, we need to overcome new difficulties to exploit the new potential:

- Increased VM placement complexity because of multiple objectives and necessary information geographically spread.
- Increased deployment complexity because of interacting with multiple vendors, providers, infrastructures, and interfaces for resource allocation.
- More careful application distribution: Wrong distribution of or wrong communication among components of application could lead to over-used bottleneck components and this degrades overall performance.

*Adaptive Deployment:* Such a deployment can be realized simple: Application Providers allocate necessary resources at each data centre upfront. Additional allocated resources prepare for peak loads. Most of the time, this over-allocation causes unnecessary costs because in normal load situations they are not used. To solve this waste of money, the resource allocation is dynamically adapted to the current load situation. The adaptation process involves tracking the current load situation, computing an adapted resource allocation, and applying the reconfiguration. A manual processing is a very labour-intensive task, even if the manual part is limited to reconfiguration. Respectively, a high reaction time for adapting to changing load situation is the major drawback. Only an automation of all three steps can reduce the reaction time to a minimum. In short, only an automated, adaptive, dynamic deployment of applications across different cloud sites saves expenses while exploiting the new QoS opportunities at the same time.

*Complex Applications:* An automated deployment for a simple application is already algorithmically challenging (Section I-C). In the background of a continuesly changing resource allocation or deployment setup, two practical issues

arise for complex application. First, composition of components as in multi-tier architectures has to properly connect individual components to each other. In general, components (from one tier) not necessarily access the closest components (of a lower tier). Such components have to be properly reconfigured and reconnected to each other. Second, stopping a Virtual Machine of a state-full application could cause data lose. The infrastructure could support these applications to maintain their data integrity in a dynamic changing system. Hence, our system notifies the application of future VM operations a prior.

### B. Motivation

Our research focuses on the deployment of complex applications spread over many, geographically distributed cloud resources in an automated, adaptive, dynamic way. It is motivated by further improving application's network-related qualities of service (QoS) like response times for interactive applications or data rates for media-intensive applications. Such an agile deployment implies new business opportunities in monetizing such applications for traditional over-the-top application providers like Google or Microsoft, carrier-like ISPs, or Mobile Operators. Possible applications range from highly interactive applications like Cloud Gaming [22] to personalized content delivery platforms like Media Clouds [5], [46].

In particular, Network carriers have advantages over other players when deploying applications/services in their network: (a) Detailed and up-to-date information about the network topology is available for better VM placement decision. Other players simply don't have such information or can only access topology information on a coarse grained level, because network carries usually not share these vital business information. (b) In contrast to the other players, Network carriers could influence routing decisions. With this, they are able to offer application with great network connectivity.

Using our toolkit, a Network carrier can offer topology-aware adaptations as a service for over-the-top providers without sharing network topology data.

### C. Research

We need to answer two essential questions:

**(I) How many allocated resources are required where?** We decomposed this question into four sub-questions:

1) Which measured values (CPU utilization, incoming requests, etc.) either best reflect the application performance or a need for adaptation? We call them **Online Load and Performance Indicator**s (OLPIs) as they are continuously monitored and indicate the need for adaptation.
2) The application and its components have resource, scaling, and communication requirements. Where are appropriate cloud resources? Which component and how many are needed? How are these components distributed across these cloud resources?
3) How to encode – for example, as a formula – the relationship between these OLPIs and a target VM placement satisfying/improving the QoS?
4) Finally, factors like edge constraints and side-effects of using a shared infrastructure will influence the quality of the adaptation: (temporally low) prices, real measured performance, bandwidth bottlenecks, or migration costs.

Answers to these questions influence the design of VM placement algorithms.

**(II) How to deploy elastic applications?** This question covers three parts:

1) Which are the common functionalities for adaptations?
2) With an ever-changing allocation setup, how are components of a complex distributed application supported? We discuss this in Section III-D.
3) What does a flexible and extensible system architecture look like?

The first question (I) covers OLPI monitoring, gathering, and processing to compute a placement. A *placement* is the (re)configuration of an application's allocated resources or states. An *adaptation* describes the whole process of reacting to OLPIs and utilizes an algorithm to compute suitable placement. Because some placement algorithms already exists, we concentrate in this paper on question II by covering the following points:

- We propose a flexible framework that offers high-level interfaces for an adaptation plug-in. They simplify the retrieval of necessary input data for various placement algorithms and the implementation of application reconfiguration (Section III-B).
- The presented toolkit implements fundamental functionality useful for all placement algorithms like batch VM deployment, support for state-full applications, or complex application architectures (Section IV, Section III-D).
- Categorization of deployment schemes and our proposal – the application template in Section II-B.

We evaluate this system with an adaptive deployment of a sample complex application on our testbed (Section V).

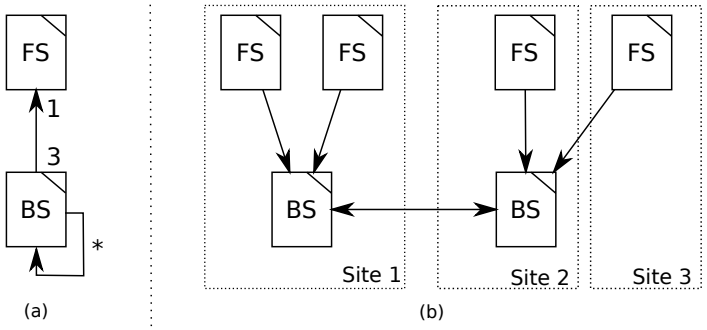


Figure 1: Sample complex application: (a) The architecture/template and (b) a sample allocation graph.

### D. A Sample Complex Application

As an illustration, the application in Figure 1 will serve as an example for this paper. It has a combined Two-Tier and Peer-to-Peer architecture that consists of two components: A frontend server (FS) and a backend server (BS). Users interact with FSs; these FSs access one BS; all BSs communicate with each other to create a P2P overlay.

Figure 1b provides an example deployment over three cloud sites. Arrows indicate the communication direction among the components, e.g. "A asks B". The following three aspects determine the application's adaptive deployment. First, the incoming request rate is the OLPI determining the adaptation. Crossing an OLPI threshold triggers the allocation of new FSs or shutdown of obsolete FSs. Second, for performance reasons, a limited scalability is defined: Each BS can serve three FSs. An edge from BS to FS with numbers in Figure 1a indicates that; Section III describes the application template. Finally, the adaptation uses topological data to launch VMs on cloud resources nearby customers (in the same region, area, country, autonomous system, or core network)

This example adaptation is simple, yet sufficient for the purpose of this paper: Describing the required information, the information exchange, and functionality for automatically adapting a complex application.

## II. Related Work

### A. Existing solutions for IaaS deployment

Recent development and research actives, result in a broad area of IaaS cloud deployment solutions. We compare them by concentrating on the following properties:

**1)** *Decentralized* solutions do not rely on a central component, e.g. for coordination.
**2)** *Multiple Sites/Clouds* describe VM deployment across different cloud locations.
**3)** *Complex Applications* or a composition of components are considered.
**4)** Support for *Elastic Applications* by providing help, hints, or triggers.
**5)** Different *Configuration* approaches:
**5a)** The solution *installs software* after deploying a base image in contrast to just providing predefined VM image deployment.
**5b)** Some solutions leverage possibilities by introducing an execution platform on the *PaaS layer*.
**5c)** Adaptations take not only resource allocation (on infrastructure level) into account but also application modes are *reconfigured*.
**6)** Which kind of *Adaptation* is supported:
**6a)** Human user *manually triggers* adaptation.
**6b)** A time schedule results in *time-triggered* changes.
**6c)** *Dynamically triggered* changes are load-adaptations (e.g. incoming requests), constraint-dependent (e.g. only below a price threshold), etc. They need a monitor-reaction-loop.
**6d)** *Application-level* data are monitored and considered in placement decisions.

Table I lists both commercial products and proposed research. For each of these solutions, the satisfying properties are marked.

Our solution covers a broader range of properties than any existing solution does: Most of the solutions consider allocation of resources at multiple locations/sites (property 2), but ignore either inter-dependency of complex application/composition of services (prop. 3) or support for elastic applications (prop. 4). We cover both at Section III-C and Section III-D.

Only a few deal with the potential of collecting application-level information (as OLPI; prop. 6d) and reconfiguring applications (prop. 5c) to dynamically improve/adapt deployment quality. Our steering system realizes both (Section III-D).

PaaS solutions, configuration frameworks (prop. 5a), application-level adaptations [35] or middlewares [45] are orthogonal to our solution. These solutions, in fact, can benefit from our work by using our toolkit on top of them. However, they need to integrate into our framework by providing OLPIs and reacting to steering events.

### B. Application Description Techniques

The most fundamental version of application deployment via IaaS is requesting single VMs. However, single requests are exhausting for today's complex, multi-tier applications. One solution is to describe the whole application, adaptation characteristics, and constraints at the initial request. Then, an adaptation logic can interpret this description and deploy the application. Some approaches [20], [23], [41], [44] improve deployment by describing the application more abstractly/-compactly. Existing descriptions can be categorized along two dimensions: 1) How are the application components described? 2) How are the adaptation characteristics described?

We identified three different categories to describe the application:

1a) In **Single-VM** [19], [39] deployments, a customer specifies and requests only individual VMs. This results in a labour- and time-intensive task for complex, widely distributed applications.
1b) An **Allocation Graph** [23] consists of nodes indicating VMs and their requirements. Edges specify connection properties between these VMs. It describes a concrete placement and allocation.
1c) An **Architecture Graph** [20] describes the components of the architecture and how they are connected. To obtain a placement (or Allocation Graph), such an Architecture Graph has to be transformed. We consider also a simple list of components as in [41] as an Architecture Graph without edges.

In general, a more abstract application description like the architecture graph leaves more freedom for placement algorithms to decide where to place how many resources. In contrast, an allocation graph is more limited, even if it is enriched with rules [23].

We identified two different categories of adaptations:

2a) **Rule-based** systems [20], [23], [39], [44] define reactions to conditions for rule engines. They evaluate them to deduce an adaptation.
2b) **Optimization-based** system [41] optimize a goal function, for instance, minimize costs. The result of such optimizations imply a placement.

As a combination of these approaches, we designed the **Application Template** as an Architecture Graph enriched with annotations for adaptations: Node's annotations specify VM resource requirements and edge annotations describe scaling relationships between components, which can be transformed in weights for optimization problems – as we already do with

Table I: Compared properties of related software solutions.

| Solution | 1 | 2 | 3 | 4 | 5a/b/c | 6a/b/c/d | Details |
|---|---|---|---|---|---|---|---|
| Ubuntu Juju [8] | – | – | x | x | x/–/– | x/–/–/– | Basic services are described as predefined charms. |
| RIM Carina [34] | – | x | x | x | –/–/– | x/x/x/– | It automate and speed up the deployment of services onto the OpenNebula private clouds. |
| OpenNebula Service Manager [32] | – | – | x | – | –/–/– | –/–/–/– | OpenNebula extension for managing multi-tier services with atomic operations. |
| Amazon Web Service [38], [39] | – | – | x | – | –/–/– | x/–/x/– | Provides monitoring and policy-driven auto-scaling for single-tier applications. |
| Rightscale [33] | – | x | x | – | –/x/x | x/x/x/– | Support higher-level services upon commercial IaaS clouds or for IaaS Open Source solutions. |
| CA 3Tera [2] | – | – | x | x | –/–/– | x/–/–/– | 3Teras AppLogic automates complex application deployment. |
| Flexiscale [16] | – | x | x | – | –/–/– | x/–/x/– | Scales multi-tier applications automatically. |
| Aeolus [11] | – | x | x | x | x/–/– | x/–/–/– | Aeolus's component model specifies compositions of services to automate deployments, planning of day-to-day activities such as software upgrade planning, service deployment, elastic scaling. |
| Moreno et al. [41] | – | x | – | – | –/–/– | –/x/x/– | Cloud Broker's static/dynamic scheduler can be configured by optimization criteria and contrains. Evaluation is done with high-throughput jobs. |
| Han et al. [20] | – | x | x | – | –/–/– | –/–/x/– | Their platform realize adaptive scaling of multi-tier application. Their solution is limited to those applications and rely on intermediate load-balancer. |
| Liu et al. [25] | – | – | x | – | –/x/x | x/–/–/x | Their work tackles reconfiguring elastic applications and is inspired by Java's Spring IoC concept. |
| Leitner et al. [24] | – | – | x | x | –/x/x | x/–/–/– | They propose a Java-based middleware for developing application to deploy automatically scaled |
| Mcconnell et al. [26] | – | x | x | – | –/–/– | x/–/–/– | Their cloud management solution considers inter-VM and VM-user-proximity like link data-rates. |
| Xabriel et al. [42] | – | – | x | – | –/–/– | x/–/–/– | They transform their service composition meta-model into an allocation-graph. |
| NHan et al. [27] | – | – | – | – | x/–/– | x/–/–/– | The authors proposes a model driven VM request and configuration system based on feature or functionality definitions. |
| Etchevers et al. [15] | x | x | – | – | x/–/– | x/–/–/– | A distributed self-configuration protocol installs and configures new VMs. |
| Ferrer et al. [14] | – | x | – | – | –/x/– | x/–/–/– | Their PaaS toolkit uses past measurement to deploy SLA-constrained services. |
| Agarwal et al. [3] | – | x | – | – | –/–/– | –/x/x/– | Their system distributes application-level items/content close to their users weighted by user's geographical location. |
| Buyya et al. [7] | – | x | – | – | –/–/– | –/x/–/x | They propose a cloud service market/directory, where broker can query and negotiate cloud resources from |
| Nishimura et al. [28] | – | x | – | – | –/–/– | –/–/x/– | Their system realizes adaptive deployment for state-full applications. |
| Xu et al. [44] | – | x | x | – | –/–/– | –/–/–/– | Presents carrier tailored service description as extension to OVF. |
| Application Deployment Toolkit | | x | x | x | –/–/x | x/x/x/x | Our customizable framework covers a broader range of properties than any existing solution does. |

our current research in progress. Section III-C provides details and Section V-B shows a sample.

### C. Adaptive Algorithms

This section describes recent work of placement algorithms for adaptation. Their management ranges from inside data centre [4], [20], [41] to across data centres [9], [26], [43]. Some include resource prices [9], [41], [43] and network-properties [9], [26], [41], [43] as costs. Adaptive algorithms react to user demand [9], [20], [26], [41], [43] or infrastructure measurement [4].

## III. Toolkit Architecture

### A. Overview

The architecture of our Application Deployment Toolkit (ADT) encompasses **three roles** (Figure 2): First, the *application provider* is the application owner or developer of an application. For example, an over-the-top provider offers services realized as a distributed application. Second, the *infrastructure provider* offers compute, data, and/or network resources as IaaS. For example, an Internet Service Provider offers "in-network" resources close to customers. Third, the ADT provider offers our proposed adaptive IaaS.

An application provider specifies the application architecture and adaptation configuration at the initial request. These

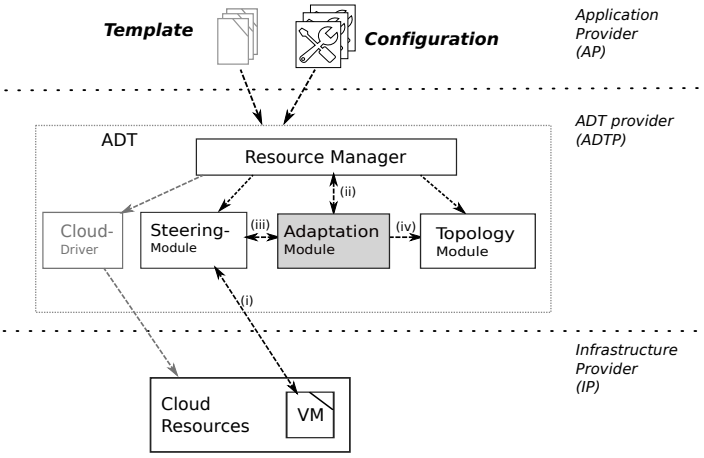


Figure 2: ADT Framework with plug-able Adaptation Module

specifications are part of the top level interface between ADT and an application provider (Section III-C). Beside this, ADT uses a common IaaS interface (Section IV) to interact with infrastructure providers and a new steering interface to interact with the application inside VMs (Section III-D).

All three roles are independent, but combinations are possible: If, for example, a mobile carrier offers and deploys its services, the same company fills out all roles. This carrier can also utilize

external resources, for instance, from compute centres or from other carriers; they are independent infrastructure provider.

**Framework:** One key element of ADT's architecture (Figure 2) is the customizable framework around the exchangeable Adaptation Module (AM) (Section III-B). Specialized AMs can realize a wide range of adaptation logics or placement algorithms, which vary depending on the provider's requirements, application's constraints, or optimization goals. Similarly, the exchanged data can be custom-tailored between the AM and the application in the VMs. The Steering Module (SM) implements this communication (Section III-D). The Topology Module (TM) provides a network topology model including available cloud sites and supports configuration of network related services (Section III-E). Overarching, the Resource Module (RM) takes operations from the AM to change the allocation or network services (Section III-F).

### *B. Adaptation Module*

The ADT can handle multiple applications at the same time. Each application's deployment is controlled by one Adaptation Module (AM). Many different AMs can coexists.

Each AM interacts over three interfaces (ii,iii,iv), shown in Figure 2:

- The AM can query the Topology Module (TM) for a list of available cloud resources or for network topology information with integrated resource information (iv). TM discovers, gathers, and stores related data; see Section III-E.
- Through the Steering Module (SM), the application components continuously report OLPIs; see Section III-D, VM→SM→AM (iii).
- The AM can call methods of SM, for sending data back to the application, for instance, to reconfigure internal states; see Section III-D, AM→SM→VM (iii).
- The Resource Manager (RM) deploys VMs on behalf of the AM (ii), see Section III-F.
- The TM provides also high-level interfaces to configure network-related services, like SDN, DNS, etc. Section III-E. They are exposed via RM and not directly; (ii), see Section III-F.
- AM can query the current state of requested or allocated resources; see Section III-F; RM→AM (ii).
- An application provider can query AM's status, internal states or reconfigure it via a top level interface; see Section III-C.

An adaptation cycle processes the OLPIs and decides to change the allocation or not. Parallel to an adaptation cycle the application sends continuesly OLPIs. They can arrive any time (iii) and thus the AM needs to maintain its own data structure to collect, merge, and smooth such data. If the AM recognizes a significant change, e.g. in the incoming request rates of some component, it recomputes the placement. Beside the OLPIs another two sources are input for a placement algorithm: The available cloud locations (iv) and the current allocation graph of the application (ii) – which component is allocated where. The results of a placement algorithm is a resource reconfiguration (ii), a network service setup (ii), or a application-level reconfiguration (iii).

**Framework:** Three interfaces contain place-holders for the Adaptation Module's custom-tailored data: First, the exchanged data between the AM and the application inside a VM. This data contains continuously reported OLPI. In addition, custom application-level hints can adjusting the adaptation. Second, in the opposite direction, the AM can send event to reconfigure the components inside the VM. Third, the application provider can reconfigure the AM and query AM internal data through ADT's top level interface.

However, the openness and flexibility comes with the limitation that three entities either provide or use a custom-tailored interface: The application provider (1) chooses an Adaptation Module (2), which itself requires particular OPLIs from the application through the Steering System (3). For that, the application VMs have to be prepared accordingly. The application provider (1) has to do that.

### *C. AP Interface*

The AP interface realizes the interaction between ADT and an application provider. A key point of this architecture is the deployment description, which contains the following information:

- The **Adaptation ID** specifies the Adaptation Module (AM) to be created.
- The **Application Template** describes the software architecture as a composition of single components – a graph with nodes as components and directed edges reflecting the inter-component communication. Nodes are annotated with information necessary for VM request: CPU, Memory, and VM image. Each edge describes "component A uses component B". Optionally, the graph is annotated with adaptation related weights, hints, or values.
- The **Adaptation Configuration** contains adaptation-specific data. This way, the Adaptation Module can receive a tailored configuration. In Section V-B we describe a sample Adaptation Module and its configuration.

This interface allows two extreme use-cases. On the one hand, an ADT provider offers a fixed and generic adaptation algorithm. The **Configuration** are parameters, for instances, key-value-pairs. An application provider (AP) can select such an predefined algorithm without developing its own AM. Then, the AP only need to configure the VM to provide the OLPI required by that algorithm.

On the other hand, an AP can develop a highly specialized algorithm, custom-tailored for one application. This enables to deploy applications with a more fine-tuned adaptation logic.

Our xml-based description language contains the three above elements. The Application Template has node annotations for resource requirements and images-urls. Edges contain annotations for scaling and inter-connection information. In addition, such information can also be encoded in extensions of the DMTF's OVF, CIM, OGF's OCCI, or OASIS's Standards [12], [17]–[19].

### D. Steering System

Until now, most virtualized systems strongly separate the application level and the infrastructure level. We soften this separation by developing the steering system – a bridge between both levels. The Steering Module (SM) communicates via a steering interface exposed by VMs (Figure 2 (ii)). The Steering System realizes four novel interactions between an application and the cloud infrastructure:

The first interface of a VM **allows application-level reconfigurations** (iii, i). After the application module (AM) computes a new placement, the framework supports the reconfiguration of individual running components. This reconfiguration data is send in an AM-specific format specified by the AM developer. The application provider needs to implement the steering interface inside its application VMs to react on this data.

The second VM interface realizes **elasticity support** for stateful application (i). Events are sent to running VMs before and after resource changes, such as VM launch, resume, suspend, destroy, migrate, etc. These events are additionally send to VMs connected to the affected VM. This enables the application inside the VM to change its state or operational mode, e.g. by migrating states before been shut down.

The third VM interface realizes **elasticity support** for complex application architectures (i). A placement for a complex application is similar to an Allocation Graph (see Section II-B). This graph includes the interconnections of each application component. The ADT automatically sends corresponding reconfiguration events to the VMs. The application provider only need to react in this events, e.g., by writing the components configuration.

Finally, the interface of the Steering Module (SM) **allows applications to provide OLPIs** (i, iii). The application provider need to setup the VM to send the OLPIs necessary for the selected AM.

The usage of our elasticity support is beneficial in the following six situation:

First, a new component/VM is deployed: After the VM boot, the component receives a bootstrap-event with connection data of its neighbors. All neighbors receive a connection-update-event with connection data of their new neighbor. For example, VM-firewall rules for incoming traffic can be set or the load is balanced across all outgoing edges.

Second, an old component is resumed: As the application's state probably had changed while being paused, this case is similar to case a). This event is sent the same way the first event is send. But component reactions might be different, e.g., flush caches, reset state.

Third, a component's resources increase/decrease, e.g. more cpu, memory, or storage: One resource-update-event is sent only to the affected component. It indicates which resource had changed. Usually, in-VM software is not notified that e.g. more cpus for more parallel processing are available. In an opposed situation, for example, memory drain could cause software crashes. The resource-update-event is sent before and after changes take place. This way, the component is able to proactively react, e.g. release memory before drain.

Fourth, a component is shut down: The component receives a shutdown-event some time before. This shutdown-event contains also a list of still running neighbors. This way, the component migrates its state, session, or data to those components. Former neighbors are informed that this component will be shutdown.

Fifth, a component is paused: Similar to case d) a pause-event is issued and the component is able to migrate data before the operation take place. Former neighbors are also informed. Usually, in-VM software is not able to detect pausing, but with this event, components can react more properly.

Sixth, the inter-connections of components change: This happens when the components have enough resources and are at the right locations, but the flow and paths of inter-component traffic had changed, for example, because of SDN reconfigurations. All affected components are informed about the new inter-component connection layout.

These events support an elastic application with necessary information to adapt itself to resource reconfigurations. Stateful applications receive hints to rescue their states to other running components. Components of complex architectures receive information on how the individual components are interconnected.

In general, migration is supported by sending a pause-event before and resume-event after migration. This is not only useful if connection data, for example, IP addresses, will change, but also if states in special hardware (GPU, FPGA, InfiniBand) need to be backed up and restored for migration [6].

### E. Topology Module

The Topology Module (TM) retrieves and shares information about the topology. Available cloud resources at geographically or topologically different sites correspond to different nodes or coordinates. Distances between pairs of nodes reflect network properties like latency or data rate. Nodes additionally store Infrastructure-as-a-Service (IaaS) properties about available Virtual Machine types and prices. Both IaaS and network properties need to be up to date, so that continues adaptation to the current situation is possible.

However, network properties could change fast or fluctuate depending on the congestion level of the network. In addition, send monitoring data can be delayed. Deployment decisions become inaccurate, if topology information is outdated. Measuring the network with probes in shorter intervals will reduce this inaccuracy to the cost of additional traffic. The trade-off can not be solved easily, so that either network operator has to accept the additional traffic or application provider the reduced deployment accuracy. Taking another trade-off into consideration between costs and realized quality, the application provider can compensate the inaccuracy by deploying more resources, if quality is more important than money. Or the application provider risks an eventual quality loss by thrifty allocate resources.

Besides this topology query possibility, the TM offers a high-level interface for network-related services. For example, placement algorithms belonging to the family of facility location problems, map application requests from users to appropriate but not necessarily closest server. Network services like DNS [40] can be configured to forward application requests

to corresponding servers. This interface can also be extended to configure Software Defined Networks to use routing information from placement algorithms.

In brief, the adaptation algorithm developer are relieved from writing their own implementation for network topology retrieval and network-related services configuration. They can use the high-level interfaces that are provided.

### F. Resource Manager

The global Resource Manager (RM) coordinates resource deployments, maintains current allocated resources, and executes low-level IaaS operations. The following four interactions affect the RM:

First, an Adaptation Module (AM) can **request a placement** or reconfigure a previous resource allocation. An annotated Allocation Graph (Section II-B) describes nodes as concrete VM instances to deploy. Receiving this high-level request, RM decomposes it into two kind of low-level operations: IaaS single-VM requests and steering events. The latter operation is sent to notify components about the resource changes (cf. Section III-D). These events contains interconnection information deduced from edges of the Allocation Graph.

AMs can **request the current state** of application as an annotated Allocation Graph with additional information about components' and VMs' states.

The RM also **creates and supervises** the AMs. AM custom configuration are forwarded during creation (cf. Section III-C). Additionally, the application provider can reconfigure the AM any time; however the AM needs to be able to correctly transfer from the old configuration to the new one.

**Network service-related requests** has to pass the RM. This enables the RM to track every infrastructure affecting action, allocation, or configuration. By tracking such operations, RM could be extended for billing and accounting functionality.

## IV. Application Deployment Toolkit

We realize the purposed framework as a Open-Source prototype available at [29]. This section describes the realized and simplified functionality.

*Steering Daemon:* The counterpart of the Steering Module on the ADT side is our steering daemon inside the VM, which is started during the boot process. The steering daemon is configurable in two ways: Firstly, incoming events from Adaptation Module start shell-scripts. They can then set configuration files and restart the component. Secondly, direct event processing is possible with a python. Section V-B describes a sample usage.

*Driver:* The low-level cloud operations are applied by cloud drivers, which interface to different IaaS flavours: EC2, OCCI, OpenNebula, OpenStack [19], [30], [31], [39] including vendor/project-specific modifications.

*Topology Module:* The current implementation only supports requesting the network topology information. In addition to the presented architecture, the testbed implementation of our topology module can configure the network properties of the testbed's network (Section V-C). A topology retrieval is in this case not necessary, as the testbed's network configuration contains the network topology.

However, we envision four approaches to extend the TM to discover the network topology: At first, a probing system, e.g. deployed in VMs, measures and collects network properties between sides. With an increase number of sides, all-to-all measurements will become infeasible. Embedding sites into a multi-dimensional internet coordinate system could solve that. In this second approach, network properties are only measured against the beacon server. The coordinates or better the distance between sites correspond to one network property, e.g. latency. Finally, the infrastructure provider shared topology data: Scharf et al. [37] investigated the Application-Layer Traffic Optimization (ALTO) to gather such topology information.

*Centralized:* In order to ease design we set a centralized architecture fix early. After having carefully laid out the necessary functionality, we need to transform this architecture to be scalable and distributed as a next step.

## V. Evaluation

### A. Scenario

We refer to Section I-D describing our prototype application and Figure 1 showing the architecture and an example deployment. The adaptation increase the number of frontend server (FS) at those sites with high load, with a lot incoming requests. The FS implementation is a python web server, which measures and reports its incoming request rates as Online Load and Performance Indicators (OLPIs) per IP address. The FS also realizes a mechanism to redirect the user to their FS. The Adaptation Module computes the user-FS mapping and reconfigures it at runtime.

### B. Framework in Action

*Initial Request:* To initialize our adaptive deployment, we have to specify three elements (cf. Section III-C): The Adaptation Module ID, the module's configuration, and the Application Template. Our threshold-based adaptation can be configured with up- and down-scale thresholds for the OLPIs. The application graph is shown in Figure 1a. Annotations reflect the scalability factors between components and the intercommunication pattern of the components: One backend server (BS) receives requests from up to three frontend servers (FSs) and each BS communicates with each other BS (see '*' in the diagram).

*Topology:* To shorten the development, we have a fix IP-mapping for "resources close to the user". Additionally, the topology module is configured with a fixed topology.

*Adaptation:* All running components report their OLPI to the Adaptation Module (AM). The AM maintains cumulated incoming rates per cloud site. The AM regularly checks the cumulated rates and if a threshold is crossed a FS is deployed or destroyed. Additionally, the AM checks the scaling requirement and adds BSs as necessary. The adaptation results in two actions. A new placement is executed via the resource manager. Afterwards the FSs are reconfigured with a new mapping which redirect users to their FS.

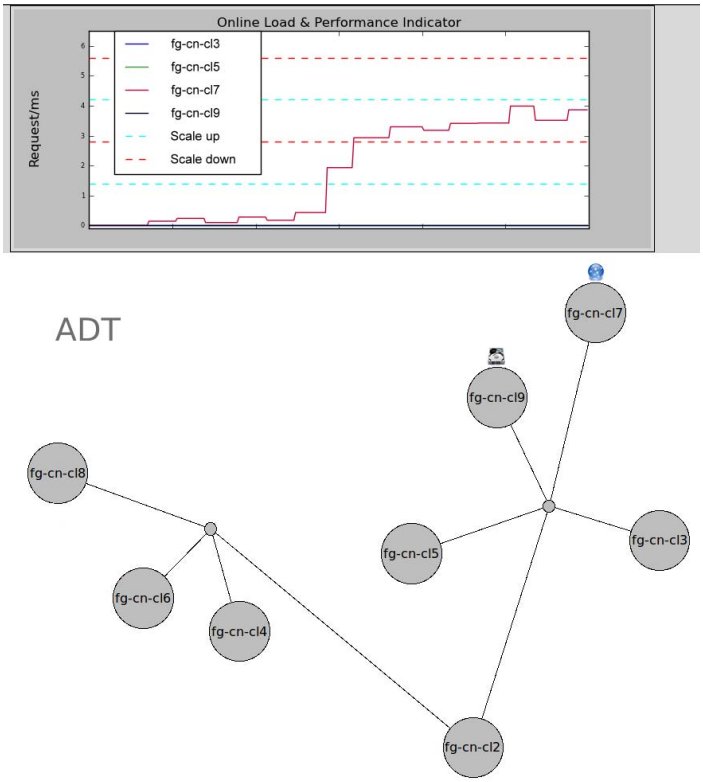


Figure 3: Testbed control panel; Upper half: OLPI over time; Lower half: Circles are sites, Icons indicating running VMs.

### C. Testbed

Our testbed consists of eight physical nodes simulating eight geographically distributed cloud sites. We use OpenStack [31] for VM deployment. ADT's TM can additionally configure artificial network scenarios (with delays, bottlenecks) by using netem [1] at runtime.

Figure 3 shows our control panel, which is not only builds upon ADT's web service interface but also remotely controls our traffic generator framework.

## VI. Conclusion

The presented framework contributes by describing necessary interfaces, functionalities, and data exchanges to deploy complex application across in a dynamic and adaptive way. Our system features a condensed, high-level interface for an customizable and exchangeable Adaptation Module.

With such a framework, future research can focus on developing placement algorithm. Necessary low-level tasks preform our framework. This supports not only research but also commercial exploitation by providing a thought-out architecture. In addition, the framework accesses current IaaS interfaces – so a deployment on Amazon's AWS can easily be done with most applications. Moreover, we successfully deployed an application on the heterogeneous testbed of the EU project SAIL [10], which is geographically distributed over Portugal, France, Sweden, and England.

Both novel approaches – describing the application via a compact annotated template and fusing the application- and infrastructure-level monitoring and reconfiguration – leverage more potential for optimization/adaptation, which further saves expenses and improves Quality-of-Service.

*Future Work:* We are currently investigating VM placements for response-time-critical application and data-rate-bounded applications using SDN.

In both scenarios, our algorithms compute not only which component has to be allocated where but also how the components are inter-connected, e.g., to avoid over-utilization. This inter-connection knowledge can be used to set routes for packets in a SDN. In addition, these algorithms might be extended to shape the traffic flow in a desired way, e.g., to do load balancing.

## VII. Acknowledgement

This research was supported by "Scalable and Adaptive Internet Solutions" (SAIL) project (funded by the european 7th framework programme; FP7-ICT-2009-5-257448; http://www.sail-project.eu/). Its developement is continued in "Comunicate. Green." (ComGreen) project (funded by Germany; BMWI's IT2Green Initiative; http://www.communicate-green.de/).

## References

[1] Network Emulator (netem). http://www.linuxfoundation.org/collaborate/workgroups/networking/netem. accessed 24.04.2013.

[2] CA 3Tera. http://www.3tera.com. accessed 12.04.2013.

[3] Agarwal, Sharad and Dunagan, John and Jain, Navendu. Volley: Automated data placement for geo-distributed cloud services. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation (NSDI'10)*, 2010.

[4] Andreolini, Mauro and Casolari, Sara and Colajanni, M and Messori, Michele. Dynamic load management of virtual machines in cloud architectures. *Cloud Computing*, 34:201–214, 2010.

[5] Bauer, Markus and Braun, Stefanie and Domschitz, Pater. Media Processing in the Future Internet. In *Proceedings of the 11th Wrzburg Workshop on IP: Visions of Future Generation Networks*, 2011.

[6] Birkenheuer, Georg and Brinkmann, Andre and Kaiser, Jürgen and Keller, Axel and Keller, Matthias and Kleineweber, Christoph and Konersmann, Christoph and Niehörster, Oliver and Schäfer, Thorsten and Simon, Jens and Wilhelm, Maximilian. Virtualized HPC: a contradiction in terms? *Software: Practice and Experience*, 2011.

[7] Buyya, Rajkumar and Ranjan, Rajiv and Calheiros, Rodrigo N. InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of. *Algorithms and Architectures for Parallel Processing*, pages 13–31, 2010.

[8] Canonical. Ubuntu Juju. https://juju.ubuntu.com. accessed 12.04.2013.

[9] Charrada, Faouzi Ben and Tebourski, Nourhene and Tata, Samir and Moalla, Samir. Approximate Placement of Service-Based Applications in Hybrid Clouds. In *Proceedings of the 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 161–166. Ieee, 6 2012.

[10] The SAIL Consortium. Description of Project-wide Scenarios and Use Cases. Deliverable FP7-ICT-2009-5-257448-SAIL/D.A.1, SAIL project, February 2011. Available online from http://www.sail-project.eu.

[11] Cosmo, Roberto Di and Zacchiroli, Stefano and Zavattaro, Gianluigi. Towards a Formal Component Model for the Cloud. In *Software Engineering and Formal Methods*, volume 7504 of *Lecture Notes in Computer Science*, pages pp. 156–171. Springer Berlin Heidelberg, 2012.

[12] DMTF. Common Information Model. http://www.dmtf.org/standards/cim. accessed 19.04.2013.

[13] Alastair et. al. BonFIRE: A Multi-cloud Test Facility for Internet of Services Experimentation. In *Proceedings of the 8th International ICST Conference, TridentCom 2012*, pages 81–96, 2012.

[14] Ferrer et. al. OPTIMIS: A holistic approach to cloud service provisioning. *Future Generation Computer Systems*, 28:66–77, 1 2012.

[15] Etchevers, Xavier and Coupaye, Thierry and Boyer, Fabienne and de Palma, Nol and Salaun, Gwen. Automated Configuration of Legacy Applications in the Cloud. In *Proceedings of the Fourth IEEE International Conference on Utility and Cloud Computing*, pages 170–177. IEEE, 12 2011.

[16] Ferraris, F.L. and Franceschelli, D. and Gioiosa, M.P. and Lucia, D. and Ardagna, D. and Di Nitto, E. and Sharif, T. Evaluating the Auto Scaling Performance of Flexiscale and Amazon EC2 Clouds. In *Proceedings of 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 423–429. IEEE, 2012.

[17] Organization for the Advancement of Structured Information Standards. https://www.oasis-open.org/standards. accessed 24.04.2013.

[18] Distributed Managment Task Force. Open Virtualization Format. http://www.dmtf.org/standards/ovf. accessed 19.04.2013.

[19] Open Grid Forum. Open Cloud Computing Interface. http://occi-wg.org/about/specification/. accessed 24.04.2013.

[20] Han, Rui and Guo, Li and Guo, Yike and He, Sijin. A Deployment Platform for Dynamically Scaling Applications in the Cloud. In *Proceedings of the Third International Conference on Cloud Computing Technology and Science*, pages 506–510. IEEE, 11 2011.

[21] Ibrahim, Ghida and Chadli, Youssef and Kofman, Daniel and Ansiaux, Alexandra. Toward a new Telco role in future content distribution services. In *2012 16th International Conference on Intelligence in Next Generation Networks*, pages 22–29. IEEE, 10 2012.

[22] Jarschel, Michael and Schlosser, Daniel and Scheuring, Sven and Hofeld, Tobias. An Evaluation of QoE in Cloud Gaming Based on Subjective Tests. In *Proceedings of the Fifth International Conference of Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, pages 330–335. IEEE, 6 2011.

[23] Koslovski, Guilherme Piegas and Primet, Pascale Vicat-Blanc and Charo, Andrea Schwertner. VXDL: Virtual Resources and Interconnection Networks Description Language. *Networks for Grid Applications*, 2:138–154, 2009.

[24] Leitner, Philipp and Satzger, Benjamin and Hummer, Waldemar and Inzinger, Christian and Dustdar, Schahram. CloudScale - a Novel Middleware for Building Transparently Scaling Cloud Applications. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing - SAC '12*, page 434. ACM Press, 3 2012.

[25] Liu, Huan. Rapid application configuration in Amazon cloud using configurable virtual appliances. In *Proceedings of the 2011 ACM Symposium on Applied Computing - SAC '11*, page 147. ACM Press, 3 2011.

[26] Mcconnell, Aaron and Parr, Gerard and Mcclean, Sally and Morrow, Philip and Scotney, Bryan. Towards a SLA-compliant Cloud Resource Allocator for N-tier Applications. In *Cloud Computing 2012*, pages 136–139, 2012.

[27] Nhan, Tam Le and Suny, Gerson and Jzquel, Jean-Marc. A Model-Driven Approach for Virtual Machine Image Provisioning in Cloud Computing. In *Service-Oriented and Cloud Computing*, volume 7592, pages 107–121, 9 2012.

[28] Nishimura et.al. Applying flexibility in scale-out-based web cloud to future telecommunication session control systems. In *Proceedings of the 16th International Conference on Intelligence in Next Generation Networks*, pages 1–7. IEEE, 10 2012.

[29] University of Paderborn; Computer Network Research Group. Application Deployment Toolkit - Source Package. http://agk-lpc27.cs.upb.de/adt-install.zip. accessed 24.04.2013.

[30] OpenNebula. OpenNebula.org Project. http://opennebula.org/about:about. accessed 24.04.2013.

[31] OpenStack. OpenStack Cloud Software. http://www.openstack.org/. accessed 25.04.2013.

[32] DSA Research. OpenNebula Service Manager. http://opennebula.org/software:ecosystem:oneservice. accessed 12.04.2013.

[33] Rightscale. http://www.rightscale.com. accessed 12.04.2013.

[34] RIM. Carina Environment Manager. https://github.com/blackberry/OpenNebula-Carina/blob/master/docs/GETTING_STARTED.md. accessed 24.04.2013.

[35] Rogrio de Lemos et. al. Software Engineering for Self-Adaptive Systems: A Second Research Roadmap. In *Software Engineering for Self-Adaptive Systems II*, volume 7475 of *Lecture Notes in Computer Science*, page 1.32. Springer Berlin Heidelberg, 2013.

[36] Sae Lor, Suksant and Vaquero, Luis M and Murray, Paul. In-NetDC : Data Centres in Core Networks. *IEEE Communication Letters*, 16:1–4, 2012.

[37] Scharf, Michael and Voith, Thomas and Roome, W. and Gaglianello, Bob and Steiner, Moritz and Hilt, Volker and Gurbani, Vijay K. Monitoring and abstraction for networked clouds. In *2012 16th International Conference on Intelligence in Next Generation Networks*, pages 80–85. IEEE, 10 2012.

[38] Amazon Web Service. Auto Scaling. http://aws.amazon.com/autoscaling/. accessed 24.04.2013.

[39] Amazon Web Service. Elastic Load Balancing. http://aws.amazon.com/en/elasticloadbalancing. accessed 24.04.2013.

[40] Amazon Web Service. Route 53. http://aws.amazon.com/en/route53/. accessed 25.04.2013.

[41] Tordsson, Johan and Montero, Rubn S. and Moreno-Vozmediano, Rafael and Llorente, Ignacio M. Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. *Future Generation Computer Systems*, 28:358–367, 2 2012.

[42] Xabriel J. Collazo-mojica, S. Masoud Sadjadi. A Metamodel for Distributed Ensembles of Virtual Appliances. In *SEKE 2011*, pages 560–565, 2011.

[43] Xu, Hong and Li, Baochun. Joint Request Mapping and Response Routing for Geo-distributed Cloud Services. In *Proceedings of the 32nd IEEE International Conference on Computer Communications (INFOCOM)*, pages 878–886, 2013.

[44] Xu, Huiyang and Song, Meina and Peng, Jin and Yu, Qing. Research on telecom service deployment in cloud environments. In *5th International Conference on Pervasive Computing and Applications*, pages 189–194. IEEE, 12 2010.

[45] Yin, Qin and Schpbach, A and Cappos, Justin and Baumann, Andrew and Roscoe, Timothy. Rhizoma: a runtime for self-deploying, self-managing overlays. *Middleware 2009*, 5896:184–204, 2009.

[46] Zhu, Wenwu and Luo, Chong and Wang, Jianfeng and Li, Shipeng. Multimedia Cloud Computing. *IEEE Signal Processing Magazine*, 28:59–69, 2011.