Traffic Management for Cloud Federation

Wojciech Burakowski[1(✉)], Andrzej Beben[1], Hans van den Berg[2], Joost W. Bosman[3], Gerhard Hasslinger[4], Attila Kertesz[5], Steven Latre[6], Rob van der Mei[3], Tamas Pflanzner[5], Patrick Gwydion Poullie[7], Maciej Sosnowski[1], Bart Spinnewyn[6], and Burkhard Stiller[7]

[1] Warsaw University of Technology, Warsaw, Poland
{wojtek,abeben,m.sosnowski}@tele.pw.edu.pl
[2] Netherlands Organisation for Applied Scientific Research, The Hague, Netherlands
j.l.vandenberg@tno.nl
[3] Centrum Wiskunde & Informatica, Amsterdam, Netherlands
{j.w.bosman,r.d.van.der.mei}@cwi.nl
[4] Deutsche Telekom AG, Bonn, Germany
Gerhard.Hasslinger@telekom.de
[5] University of Szeged, Szeged, Hungary
{keratt,tamas.pflanzner}@inf.u-szeged.hu
[6] University of Antwerp - iMINDS, Antwerp, Belgium
{steven.latre,bart.spinnewyn}@uantwerpen.be
[7] University of Zürich - CSG@IfI, Zürich, Switzerland
{poullie,stiller}@ifi.uzh.ch

Abstract. The chapter summarizes activities of COST IC1304 ACROSS European Project corresponding to traffic management for Cloud Federation (CF). In particular, we provide a survey of CF architectures and standardization activities. We present comprehensive multi-level model for traffic management in CF that consists of five levels: Level 5 - Strategies for building CF, Level 4 - Network for CF, Level 3 - Service specification and provision, Level 2 - Service composition and orchestration, and Level 1 - Task service in cloud resources. For each level we propose specific methods and algorithms. The effectiveness of these solutions were verified by simulation and analytical methods. Finally, we also describe specialized simulator for testing CF solution in IoT environment.

Keywords: Cloud federation · Traffic management
Multi-layer model · Service provision · Service composition

1 Introduction

Cloud Federation (CF) extends the concept of cloud computing systems by merging a number of clouds into one system. Thanks to this, CF has a potentiality to offer better service to the clients than it can be done by a separated cloud. This can happen since CF has more resources and may offer wider scope of services. On the other hand, the management of CF is more complex comparing to

I. Ganchev et al. (Eds.): Autonomous Control for a Reliable Internet of Services, LNCS 10768, pp. 269–312, 2018.
https://doi.org/10.1007/978-3-319-90415-3_11

this which is required for a standalone cloud. So, the effective management of resources and services in CF is the key point for getting additional profit from such system. CF is the system composing of a number of clouds connected by a network, as it is illustrated on Fig. 1. The main concept of CF is to operate as one computing system with resources distributed among particular clouds.

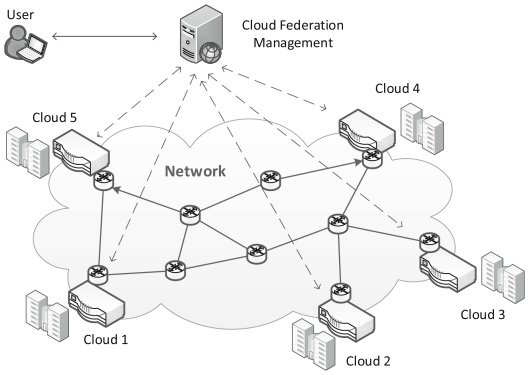


Fig. 1. Exemplary CF consisting of 5 clouds connected by network.

In this chapter we present a multi-level model for traffic management in CF. Each level deals with specific class of algorithms, which should together provide satisfactory service of the clients, while maintaining optimal resource utilization.

The structure of the chapter is the following. In Sect. 2 we present discussed CF architectures and the current state of standardization. The proposed multi-level model for traffic management in CF is presented in Sect. 3. Section 4 describes a simulation tool for analyzing performance of CF in Internet of Things (IoT) environment. Finally, Sect. 5 summarizes the chapter.

2 Cloud Federation Architectures

2.1 Cloud Architectural Views

In general CF is envisaged as a distributed, heterogeneous environment consisting of various cloud infrastructures by aggregating different Infrastructure as a Service (IaaS) provider capabilities coming from possibly both the commercial and academic area. Nowadays, cloud providers operate geographically diverse data centers as user demands like disaster recovery and multi-site backups became widespread. These techniques are also used to avoid provider lock-in issues for users that frequently utilize multiple clouds. Various research communities and standardization bodies defined architectural categories of infrastructure clouds. A current EU project on "Scalable and secure infrastructures for cloud operations" (SSICLOPS, www.ssiclops.eu) focuses on techniques for the management of federated private cloud infrastructures, in particular cloud networking

techniques within software-defined data centers and across wide-area networks. The scope of the SSICLOPS project includes high cloud computing workloads e.g. within the CERN computing cloud (home.cern/about/computing) as well as cloud applications for securing web access under challenging demands for low delay. An expert group set up by the European Commission published their view on Cloud Computing in [1]. These reports categorize cloud architectures into five groups.

- Private Clouds consist of resources managed by an infrastructure provider that are typically owned or leased by an enterprise from a service provider. Usually, services with cloud-enhanced features are offered, therefore this group includes Software as a Service (SaaS) solutions like eBay.
- Public Clouds offer their services to users outside of the company and may use cloud functionality from other providers. In this solution, enterprises can outsource their services to such cloud providers mainly for cost reduction. Examples of these providers are Amazon or Google Apps.
- Hybrid Clouds consist of both private and public cloud infrastructures to achieve a higher level of cost reduction through outsourcing by maintaining the desired degree of control (e.g., sensitive data may be handled in private clouds). The report states that hybrid clouds are rarely used at the moment.
- In Community Clouds, different entities contribute with their (usually small) infrastructure to build up an aggregated private or public cloud. Smaller enterprises may benefit from such infrastructures, and a solution is provided by Zimory.
- Finally, Special Purpose Clouds provide more specialized functionalities with additional, domain specific methods, such as the distributed document management by Google's App Engine. This group is an extension or a specialization of the previous cloud categories.

The third category called hybrid clouds are also referred as cloud federations in the literature. Many research groups tried to grasp the essence of federation formation. In general, cloud federation refers to a mesh of cloud providers that are interconnected based on open standards to provide a universal decentralized computing environment where everything is driven by constraints and agreements in a ubiquitous, multi-provider infrastructure. Until now, the cloud ecosystem has been characterized by the steady rising of hundreds of independent and heterogeneous cloud providers, managed by private subjects, which offer various services to their clients.

Buyya et al. [2] envisioned Cloud Computing as the fifth utility by satisfying the computing needs of everyday life. They emphasized and introduced a market-oriented cloud architecture, then discussed how global cloud exchanges could take place in the future. They further extended this vision suggesting a federation oriented, just in time, opportunistic and scalable application services provisioning environment called InterCloud. They envision utility oriented federated IaaS systems that are able to predict application service behavior for intelligent down and up-scaling infrastructures. They list the research issues of

flexible service to resource mapping, user and resource centric Quality of Service (QoS) optimization, integration with in-house systems of enterprises, scalable monitoring of system components. They present a market-oriented approach to offer InterClouds including cloud exchanges and brokers that bring together producers and consumers. Producers are offering domain specific enterprise Clouds that are connected and managed within the federation with their Cloud Coordinator component.

Celesti et al. [3] proposed an approach for the federation establishment considering generic cloud architectures according to a three-phase model, representing an architectural solution for federation by means of a Cross-Cloud Federation Manager, a software component in charge of executing the three main functionalities required for a federation. In particular, the component explicitly manages:

1. the discovery phase in which information about other clouds are received and sent,
2. the match-making phase performing the best choice of the provider according to some utility measure and
3. the authentication phase creating a secure channel between the federated clouds. These concepts can be extended taking into account green policies applied in federated scenarios.

Bernstein et al. [4] define two use case scenarios that exemplify the problems of multi-cloud systems like

1. Virtual Machines (VM) mobility where they identify the networking, the specific cloud VM management interfaces and the lack of mobility interfaces as the three major obstacles and
2. storage interoperability and federation scenario in which storage provider replication policies are subject to change when a cloud provider initiates subcontracting. They offer interoperability solutions only for low-level functionality of the clouds that are not focused on recent user demands but on solutions for IaaS system operators.

In the Federated Cloud Management solution [5], interoperability is achieved by high-level brokering instead of bilateral resource renting. Albeit this does not mean that different IaaS providers may not share or rent resources, but if they do so, it is transparent to their higher level management. Such a federation can be enabled without applying additional software stack for providing low-level management interfaces. The logic of federated management is moved to higher levels, and there is no need for adapting interoperability standards by the participating infrastructure providers, which is usually a restriction that some industrial providers are reluctant to undertake.

2.2 Standardization for Cloud Federation

Standardization related to clouds, cloud interoperability and federation has been conducted by the ITU (International Telecommunication Union) [6],

IETF (Internet Engineering Task Force) [7], NIST (National Institute of Standards and Technology) [8] and IEEE (Institute of Electrical and Electronics Engineers) [9]. In 2014, the ITU released standard documents on the vocabulary, a reference architecture and a framework of inter-cloud computing. The latter provides an overview, functional requirements and refers to a number of use cases. The overview distinguishes between:

- Inter-cloud Peering: between a primary and secondary CSP (i.e. Cloud Service Provider), where cloud services are provided by the primary CSP who establishes APIs (application programming interfaces) in order to utilize services and resources of the secondary CSP,
- Inter-cloud Intermediary: as an extension of inter-cloud peering including a set of secondary CSPs, each with a bilateral interface for support of the primary CSP which offers all services provided by the interconnected clouds, and
- Inter-cloud Federation: which is based on a set of peer CSPs interconnected by APIs as a distributed system without a primary CSP with services being provided by several CSPs. For each service, the inter-cloud federation may act as an inter-cloud intermediary with a primary CSP responsible for the service. The user population may also be subdivided and attributed to several CSPs.

The main functional requirements to set up and operate a cloud federation system are:

- Networking and communication between the CSPs,
- Service level agreement (SLA) and policy negotiations,
- Resource provisioning and discovery mechanisms,
- Resource selection, monitoring and performance estimation mechanisms,
- Cloud service switch over between CSPs.

Finally, the ITU [6] takes a number of use cases into account to be addressed by could interconnection and federation approaches:

- Performance guarantee against an abrupt increase in load (offloading),
- Performance guarantee regarding delay (optimization for user location),
- Guaranteed availability in the event of a disaster or large-scale failure,
- Service continuity (in the case of service termination of the original CSP), service operation enhancement and broadening service variety,
- Expansion and distribution of cloud storage, media and virtual data center,
- Market transactions in inter-cloud intermediary pattern and cloud service rebranding.

The standardization on cloud federation has many aspects in common with the interconnection of content delivery networks (CDN). A CDN is an infrastructure of servers operating on application layers, arranged for the efficient distribution and delivery of digital content mostly for downloads, software updates and video streaming. The CDN interconnection (CDNI) working group

of the IETF provided informational RFC standard documents on the problem statement, framework, requirements and use cases for CDN interconnection in a first phase until 2014. Meanwhile specifications on interfaces between upstream/downstream CDNs including redirection of users between CDNs have been issued in the proposed standards track [7]. CDNs can be considered as a special case of clouds with the main propose of distributing or streaming large data volumes within a broader service portfolio of cloud computing applications. The underlying distributed CDN architecture is also useful for large clouds and cloud federations for improving the system scalability and performance. This is reflected in a collection of CDNI use cases which are outlined in RFC 6770 [7] in the areas of:

- footprint extension,
- offloading,
- resilience enhancement,
- capability enhancements with regard to technology, QoS/QoE support, the service portfolio and interoperability.

The CDNI concept is foreseen as a basis for CDN federations, where a federation of peer CDN systems is directly supported by CDNI. A CDN exchange or broker approach is not included but can be build on top of core CDNI mechanisms.

In 2013, NIST [8] published a cloud computing standards roadmap including basic definitions, use cases and an overview on standards with focus on cloud/grid computing. Gaps are identified with conclusions on priorities for ongoing standardization work. However, a recently started standards activity by the IEEE [9] towards intercloud interoperability and federation is still motivated by today's landscape of independent and incompatible cloud offerings in proprietary as well as open access architectures.

3 Multi-level Model for Traffic Management in Cloud Federation

Developing of efficient traffic engineering methods for Cloud Federation is essential in order to offer services to the clients on appropriate quality level while maintaining high utilization of resources. These methods deal with such issues as distribution of resources in CF, designing of network connecting particular clouds, service provision, handling service requests coming from clients and managing virtual resource environment. The proposed traffic management model for CF consists of 5 levels, as it is depicted on Fig. 2. Below we shortly discuss objectives of each level of the model.

Level 5: This is the highest level of the model which deals with the rules for merging particular clouds into the form of CF. The addressed issue is e.g. amount of resources which would be delegated by particular clouds to CF. We assume that the main reason for constituting federation is getting more profit

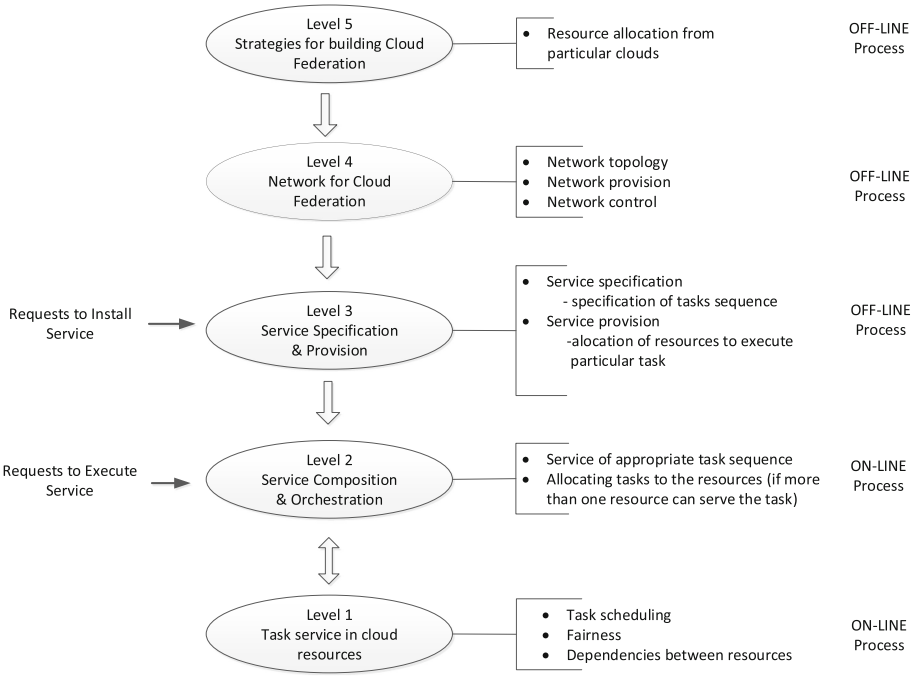


Fig. 2. Traffic management model for Cloud Federation

comparing to the situation when particular clouds work alone. So, this level deals with the conditions when CF can be attractive solution for cloud owners even if particular clouds differ in their capabilities, e.g. in amount of resources, client population and service request rate submitted by them.

Level 4: This level deals with design of the CF network for connecting particular clouds. Such network should be of adequate quality and, if it is possible, its transfer capabilities should be controlled by the CF network manager. The addressed issues are: required link capacities between particular clouds and effective utilization of network resources (transmission links). We assume that network capabilities should provide adequate quality of the offered by CF services even when resources allocated for a given service (e.g. virtual machines) come from different clouds. Effective designing of the network in question is especially important when CF uses network provided by a network operator based on SLA (Service Level Agreement) and as a consequence it has limited possibilities to control network. Currently such solution is a common practice.

Level 3: This level is responsible for handling requests corresponding to service installation in CF. The installation of new service requires: (1) specification of the service and (2) provision of the service. Specification of the service is provided in the form of definition of appropriate task sequence that is executed in CF when a client asks for execution of this service. Furthermore, provision of

the service corresponds to allocation of resources when particular tasks can be executed.

Level 2: This level deals with service composition and orchestration processes. So, the earlier specified sequence of tasks should be executed in response to handle service requests. Service composition time should meet user quality expectations corresponding to the requested service.

Level 1: The last and the lowest level deals with task execution in cloud resources in the case when more than one task is delegated at the same time to be served by a given resource. So, appropriate scheduling mechanisms should be applied in order to provide e.g. fairness for tasks execution. In addition, important issue is to understand dependencies between different types of resources in virtualized cloud environment.

3.1 Level 5: Strategy for Cloud Resource Distribution in Federation

3.1.1 Motivation and State of the Art

Cloud Federation is the system that is built on the top of a number of clouds. Such system should provide some additional profits for each cloud owner in comparison to stand-alone cloud. In this section we focus on strategies, in which way clouds can make federation to get maximum profit assuming that it is equally shared among cloud owners.

Unfortunately, there are not too many positions dealing with discussed problem. For instance in [10] the authors consider effectiveness of different federation schemes using the M/M/1 queueing system to model cloud. They assume that profit get from a task execution depends on the waiting time (showing received QoS) of this task. Furthermore, they consider scenarios when the profit is maximized from the perspective of the whole CF, and scenarios when each cloud maximizes its profit. Another approach is presented in [11], where the author applied game theory to analyze the selfish behavior of cloud owner selling unused resources depending on uncertain load conditions.

3.1.2 Proposed Model

In the presented approach we assume that capacities of each cloud are characterized in terms of number of resources and service request rate. Furthermore, for the sake of simplicity, it is assumed that both types of resources and executed services are the same in each cloud. In addition, execution of each service is performed by single resource only. Finally, we will model each cloud by well-known loss queueing system $M/M/c/c$ (e.g. [12]), where c denotes number of identical cloud resources, arrival service request rate follows Poisson distribution with parameter λ, service time distribution is done by negative exponential distribution with the rate $1/h$ (h is the mean service time). The performances of cloud system are measured by: (1) P_{loss}, which denotes the loss rate due to lack of available resources at the moment of service request arrival, and (2) $A_{carried} = \lambda h(1 - P_{loss})$, which denotes traffic carried by the cloud, that corresponds directly to the resource utilization ratio.

Now, let us search for the appropriate scheme for building CF system. For this purpose, let us consider a number, say N, of clouds that intend to build CF where the i-th cloud ($i = 1, ..., N$) is characterized by two parameters (λ_i and c_i). In addition, the mean service times of service execution are the same in each cloud $h_1 = h_2 = ... = h_N = h$. Subsequently we assume that $h = 1$, and as a consequence offered load $A = \lambda h$ will be denoted as $A = \lambda$. Next, the assumed objective function for comparing the discussed schemes for CF is to maximize profit coming from resource utilization delegated from each cloud to CF. Furthermore, the profit is equally shared among clouds participating in CF. Such approach looks to be reasonable (at least as the first approach) since otherwise in CF we should take into account requests coming from a given cloud and which resource (from each cloud) was chosen to serve the request.

We consider three schemes:

- Scheme no. 1 (see Fig. 3): this is the reference scheme when the clouds work alone, denoted by SC.
- Scheme no. 2 (see Fig. 4): this scheme is named as full federation and assumes that all clouds dedicate all theirs resources and clients to the CF system. This scheme we denote as FC.
- Scheme no. 3 (see Fig. 5): for this scheme we assume that each cloud can delegate to CF only a part of its resources as well as a part of service requests coming from its clients. This scheme we name as PCF (Partial CF).

First, let us compare the performances of schemes SC and FC in terms of resource utilization ratio and service request loss rate. The first observation is that FC scheme will have lower loss probabilities as well as better resource utilization ratio due to larger number of resources. But the open question is in which way to share profit gained from FC scheme when the clouds are of different capabilities? Table 1 shows exemplary results for the case, when the profit, which is consequence of better resources utilization, is shared equally among clouds.

The results from Table 1 show that, as it was expected, FC scheme assures less service request loss rate and better resource utilization ratio for most of clouds (except cloud no. 1 that is under loaded). Note, that if we share the profit equally, the clouds with smaller service requests rate can receive more profit from FC

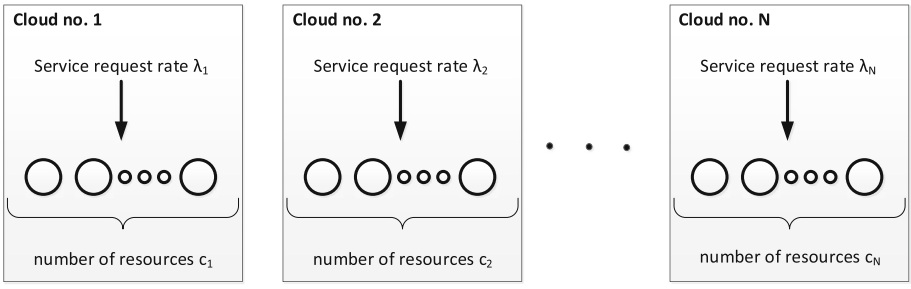


Fig. 3. Scenario with clouds working in separate way

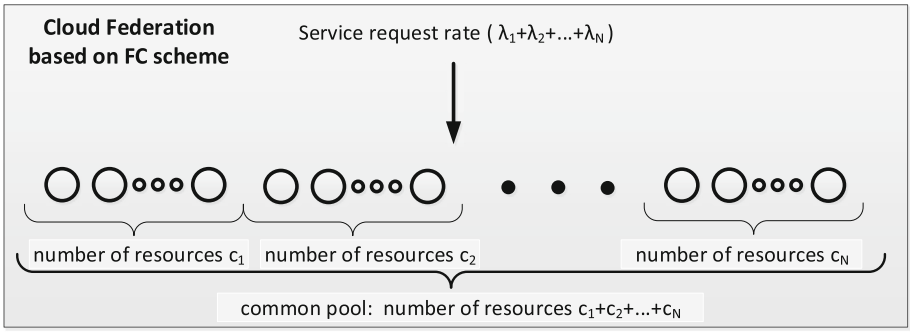


Fig. 4. Scenario with clouds creating Cloud Federation based on full federation scheme

scheme comparing to the SC scheme while the clouds with higher service request rate get less profit comparing to the SC scheme. So, one can conclude that FC scheme is optimal solution when the capabilities of the clouds are similar but if they differ essentially then this scheme simply fails.

Scheme no. 3 mitigates the drawbacks of the schemes no. 1 and no. 2. As it was above stated, in this scheme we assume that each cloud can delegate to CF only a part of its resources as well as a part of service request rate submitted by its clients. The main assumptions for PFC scheme are the following:

Table 1. Exemplary results comparing SC and FC schemes in terms of loss rate and resource utilization parameters. Number of clouds $N = 5$, values of λ: $\lambda_1 = 0.2, \lambda_2 = 0.4, \lambda_3 = 0.6, \lambda_4 = 0.8$, the same mean service times $h_1 = h_2 = h_3 = h_4 = h_5 = 1$, Number of resources in each cloud: $c_1 = c_2 = c_3 = c_4 = c_5 = 10$.

Cloud characteristics			SC scheme		FC scheme	
No.	Service requests rate	Number of resources	Resource utilization	Loss rate [%]	Resource utilization	Loss rate[%]
1	2	10	0.2	<0.01	0.6	0.02
2	4	10	0.398	0.54	0.6	0.02
3	6	10	0.575	4.3	0.6	0.02
4	8	10	0.703	12	0.6	0.02
5	10	10	0.786	21	0.6	0.02

1. we split the resources belonging to the i-th cloud ($i = 1, ..., N$), say c_i, into 2 main subsets:
   - set of private resources that are delegated to handle only service requests coming from the i-th cloud clients
   - set of resources dedicated to Cloud Federation for handling service requests coming from all clouds creating Cloud Federation, denoted as c_{i3}

2. we again split the private resources into two categories:
   - belonging to the 1st category, denoted as c_{i1}, which are dedicated as the first choice to handle service requests coming from the i-th cloud clients
   - belonging to the 2nd category, denoted as c_{i2}, which are dedicated to handle service requests coming from the i-th cloud clients that were not served by resources from 1st category as well as from common pool since all these resources were occupied.

The following relationship holds:

$$c_i = c_{i1} + c_{i2} + c_{i3}, \text{ for } i = 1, ..., N. \tag{1}$$

The handling of service requests in PFC scheme is shown on Fig. 5. The service requests from clients belonging e.g. to cloud no. i ($i = 1, ..., N$) are submitted as the first choice to be handled by private resources belonging to the 1st category. In the case, when these resources are currently occupied, then as the second choice are the resources belonging to common pool. The number of common pool resources equals $(c_{13} + c_{23} + ... + c_{N3})$. If again these resources are currently occupied then as the final choice are the resources belonging to the 2nd category of private resources of the considered cloud. The service requests are finally lost if also no available resources in this pool.

Next, we show in which way we count the resources belonging to particular clouds in order to get maximum profit (equally shared between the cloud owners). We stress that the following conditions should be satisfied for designing size of the common pool:

Condition 1: service request rate (offered load) submitted by particular clouds to the common pool should be the same. It means that

$$P_{loss1}(\lambda_1, c_{11})\lambda_1 = P_{loss2}(\lambda_2, c_{21})\lambda_2 = ... = P_{lossN}(\lambda_N, c_{N1})\lambda_N \tag{2}$$

where the value of $P_{loss}(\lambda_i, c_{i1})$ we calculate from the analysis of the system $M/M/n/n$ by using Erlang formula:

$$P_{lossi}(\lambda_i, c_{i1}) = \frac{\frac{\lambda_i^{c_{i1}}}{c_{i1}!}}{\sum_{j=0}^{c_{i1}} \frac{\lambda_i^j}{j!}}$$

Note that we only require that mean traffic load submitted from each cloud to common pool should be the same. Let us note, that the service request arrival processes from each cloud submitted to this pool are generally different. It is due to the fact that these requests were not served by 1st category of private resources and as a consequence they are not still Poissonian.

Condition 2: the number of resources dedicated from each cloud to the common pool should be the same

$$c_{13} = c_{23} = ... = c_{N3}.$$

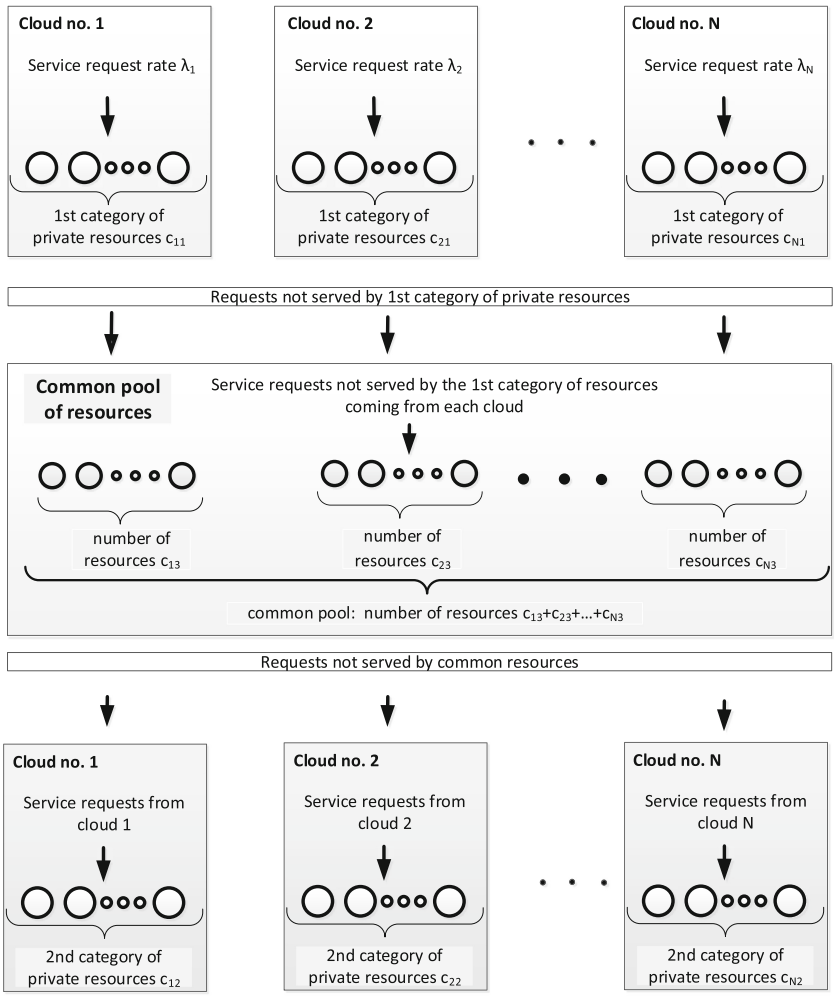


Fig. 5. Handling of service requests in PFC scheme.

Finally, the algorithm for calculating resource distribution for each cloud is the following:

Step 1: to order λ_i $(i = 1, ..., N)$ values from minimum value to maximum. Let the k-th cloud has minimum value of λ.

Step 2: to calculate (using Formula 2) for each cloud the values of the number of resources delegated to category 1 of private resources, c_{i1} $(i = 1, ..., N)$ assuming that $c_{k1} = 0$.

Step 3: to choose the minimum value from set of $(c_i - c_{i1})$ $(i = 1, ..., N)$ and to state that each cloud should delegate this number of resources to the common pool. Let us note that if for the i-th cloud the value of $(c_i - c_{i1}) \leq 0$ then no

common pool can be set and, as a consequence, not conditions are satisfied for Cloud Federation.

Step 4: to calculate from the Formula 1 the number of 2nd category of private resources c_{i2} $(i = 1, ..., N)$ for each cloud.

3.1.3 Exemplary Results

Now we present some exemplary numerical results showing performances of the described schemes. The first observation is that when the size of common pool grows the profit we can get from Cloud Federation also grows.

Example: In this example we have 10 clouds that differ in service request rates while the number of resources in each cloud is the same and is equal to 10. Table 2 presents the numerical results corresponding to traffic conditions, number of resources and performances of the systems build under SC and PFC schemes. The required amount of resources belonging to particular categories were calculated from the above described algorithm.

Table 2 says that thanks to the PFC scheme we extend the volume of served traffic from 76,95 up to 84,50 (about 10%). The next step to increase Cloud Federation performances is to apply FC scheme instead of PFC scheme.

Table 2. Numerical results showing comparison between SC and PFC schemes.

Clouds			SC scheme		PFC scheme									
No.	Service requests rate	Number of resources	Load served by cloud	Loss rate [%]	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10
1	7.5	10	6.75	10	7.50	0	5	5	0.00	2.34	4.82	7.16	3.5	0.41
2	8.4	10	7.22	14	7.50	1	4	5	0.89	2.10	4.82	7.82	6.3	0.60
3	8.4	10	7.22	14	7.50	1	4	5	0.89	2.10	4.82	7.82	6.3	0.60
4	9.3	10	7.61	18	7.50	2	3	5	1.79	1.75	4.82	8.35	10	0.74
5	9.3	10	7.61	18	7.50	2	3	5	1.79	1.75	4.82	8.35	10	0.74
6	10.2	10	7.91	22	7.50	3	2	5	2.69	1.26	4.82	8.77	14	0.86
7	10.2	10	7.91	22	7.50	3	2	5	2.69	1.26	4.82	8.77	14	0.86
8	11.1	10	8.17	26	7.50	4	1	5	3.58	0.68	4.82	9.08	19	0.91
9	11.1	10	8.17	26	7.50	4	1	5	3.58	0.68	4.82	9.08	19	0.91
10	12	10	8.38	30	7.50	5	0	5	4.49	0.00	4.82	9.31	23	0.92
Total	97.5	100	76.95		75	25	25	50	22.39	13.91	48.2	84.50		7.55

L1: offered load to common pool
L2: number of the 1st category of private resources
L3: number of the 2nd category of private resources
L4: number of resources delegated to common pool
L5: load served by the 1st category of private resources
L6: load served by the 2nd category of private resources
L7: load served by common pool of resources
L8: total load served by clouds
L9: loss rate [%]
L10: load served gain comparing to SC scheme

Unfortunately, it is not possible to be done in a straightforward way. It needs a moving of resources or service request rates between particular clouds. Table 3 presents moving of service request rates in the considered example to make transformation from PFC scheme into the form of FC scheme. For instance, cloud no. 1 should buy value of service request rate of 2.25 while cloud no. 10 should sell value of service request rate also of 2.25. Finally, after buying/selling process, one can observe that the profit gained from FC scheme is greater than the profit we have got from PFC scheme and now is equal to 91.50 (19% comparing to SC scheme and 8% comparing to PFC scheme).

Concluding, the presented approach for modeling different cloud federation schemes as FC and PFC could be only applied for setting preliminary rules for establishing CF. Anyway, it appears that in some cases by using simple FC scheme we may expect the problem with sharing the profit among CF owners. More precisely, some cloud owners may lost or extend their profits comparing to the case when their clouds work alone. Of course, more detailed model of CF is strongly required that also takes into account such characteristics as types of offered services, prices of resources, charging, control of service requests etc.

Table 3. Example showing system transformation into FC scheme.

Clouds			FC scheme						
No.	Service requests rate	Number of resources	Service requests rate to sell	Service requests rate to buy	L1	L2	L3	L4	L5
1	7.5	10	0	2.25	9.75	9.15	6.2	9.09	9.01
2	8.4	10	0	1.35	9.75	9.15	6.2	9.09	9.01
3	8.4	10	0	1.35	9.75	9.15	6.2	9.05	8.97
4	9.3	10	0	0.45	9.75	9.15	6.2	9.05	8.97
5	9.3	10	0	0.45	9.75	9.15	6.2	9.01	8.93
6	10.2	10	0.45	0	9.75	9.15	6.2	9.01	8.93
7	10.2	10	0.45	0	9.75	9.15	6.2	8.96	8.89
8	11.1	10	1.35	0	9.75	9.15	6.2	8.96	8.89
9	11.1	10	1.35	0	9.75	9.15	6.2	8.92	8.85
10	12	10	2.25	0	9.75	9.15	6.2	9.15	9.15
Total	97.5	100	5.85	5.85	97.5	91.5		91.5	91.5

L1: offered load to common pool
L2: load served by common pool of resources
L3: loss rate [%]
L4: load served gain comparing to PFC scheme
L5: load served gain comparing to SC scheme

3.2 Level 4: Network for Cloud Federation

3.2.1 Motivation and State of the Art

The services offered by CF use resources provided by multiple clouds with different location of data centers. Therefore, CF requires an efficient, reliable and secure inter-cloud communication infrastructure. This infrastructure is especially important for mission critical and interactive services that have strict QoS requirements. Currently, CF commonly exploits the Internet for inter-cloud communication, e.g. CONTRAIL [13]. Although this approach may be sufficient for non-real time services, i.e., distributed file storage or data backups, it inhibits deploying more demanding services like augmented or virtual reality, video conferencing, on-line gaming, real-time data processing in distributed databases or live video streaming. The commonly used approach for ensuring required QoS level is to exploit SLAs between clouds participating in CF. These SLAs are established on demand during the service provisioning process (see Level 3 of the model in Fig. 2) and use network resources coming from network providers. However, independently established SLAs lead to inefficient utilization of network resources, suffer scalability concerns and increase operating expenditures (OPEX) costs paid by CF. These negative effects become critical for large CFs with many participants as well as for large cloud providers offering plethora of services. For example, the recent experiences of Google cloud point out that using independent SLAs between data centers is ineffective [14]. Therefore, Google creates their own communication infrastructure that can be optimized and dynamically reconfigured following demands of currently offered services, planned maintenance operations as well as restoration actions taken to overcome failures.

3.2.2 Proposed Solution

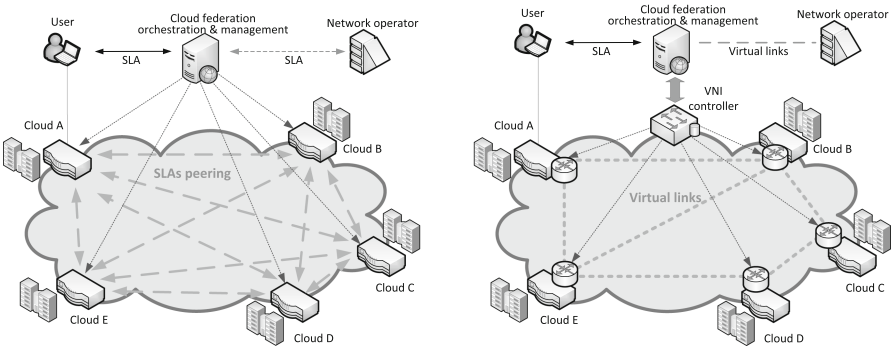
The proposed approach for CF is to create, manage and maintain a Virtual Network Infrastructure (VNI), which provides communication services tailored for inter-cloud communication. The VNI is shared among all clouds participating in CF and is managed by CF orchestration and management system. Actually, VNI constitutes a new "service component" that is orchestrated during service provisioning process and is used in service composition process. The key advantages of VNI are the following:

1. The common orchestration of cloud and VNI resources enables optimization of service provisioning by considering network capabilities. In particular, CF can benefit from advanced traffic engineering algorithms taking into account knowledge about service demands and VNI capabilities, including QoS guarantees and available network resources. The objective function of designed algorithms may cover efficient load balancing or maximization and fair share of the CF revenue.
2. New communication facilities tailored for cloud services:
   – The cloud services significantly differ in QoS requirements, e.g. interactive services are delay sensitive, while video on demand or big data storage

demands more bandwidth. Therefore, VNI should differentiate packet service and provide QoS guaranties following user's requirements. The key challenge is to design a set of Classes of Services (CoS) adequate for handling traffic carried by federation. These CoSs are considered in the service orchestration process.

– The VNI should offer multi-path communication facilities that support multicast connections, multi-side backups and makes effective communication for multi-tenancy scenarios. The key challenge is developing a scalable routing and forwarding mechanisms able to support large number of multi-side communications.

The VNI is created following the Network as a Service (NaaS) paradigm based on resources provided by clouds participating in CF. Each cloud should provide: (1) virtual network node, which is used to send, receive or transit packets directed to or coming from other clouds, and (2) a number of virtual links established between peering clouds. These links are created based on SLAs agreed with network provider(s). The VNI exploits advantages of the Software Defined Networking (SDN) concept supported by network virtualization techniques. It makes feasible separation of network control functions from underlying physical network infrastructure. In our approach, CF defines its own traffic control and management functions that operate on an abstract model of VNI. The management focuses on adaptation of VNI topology, provisioning of resources allocated to virtual nodes and links, traffic engineering, and costs optimization. On the other hand, this VNI model is used during the service composition phase for dynamic resource allocation, load balancing, cost optimization, and other short time scale operations. Finally, decisions taken by VNI control functions on the abstract VNI model are translated into configuration commands specific for particular virtual node.



(a) communication based on SLA peering. (b) communication based on VNI.

Fig. 6. Two reference network scenarios considered for CF.

Figure 6 shows the reference network scenarios considered for CF. Figure 6a presents the scenario where CF exploits only direct communication between peering clouds. In this scenario, the role of CF orchestration and management is limited to dynamic updates of SLAs between peering clouds. Figure 6b presents scenario where CF creates a VNI using virtual nodes provided by clouds and virtual links provided by network operators. The CF orchestration and management process uses a VNI controller to setup/release flows, perform traffic engineering as well as maintain VNI (update of VNI topology, provisioning of virtual links).

The Control Algorithm for VNI. The VNI is controlled and managed by a specialized CF network application running on the VNI controller. This application is responsible for handling flow setup and release requests received from the CF orchestration and management process as well as for performing commonly recognized network management functions related to configuration, provisioning and maintenance of VNI. The flow setup requires a specialized control algorithm, which decides about acceptance or rejection of incoming flow request. Admission decision is taken based on traffic descriptor, requested class of service, and information about available resources on routing paths between source and destination. In order to efficiently exploit network resources, CF uses multi-path routing that allows allocating bandwidth between any pair of network nodes up to the available capacity of the minimum cut of the VNI network graph. Thanks to a logically centralized VNI architecture, CF may exploit different multi-path routing algorithms, e.g. [15,16]. We propose a new k-shortest path algorithm which considers multi-criteria constraints during calculation of alternative k-shortest paths to meet QoS objectives of classes of services offered in CF. We model VNI as a directed graph $G(N, E)$, where N represents the set of virtual nodes provided by particular cloud, while E is the set of virtual links between peering clouds. Each link $u \rightarrow v, u, v \in N, u \rightarrow v \in E$, is characterized by a $m-$dimensional vector of non-negative link weights $w(u \rightarrow v) = [w_1, w_2, \ldots, w_m]$ which relates to QoS requirements of services offered by CF. Any path p established between two nodes is characterized by a vector of path weights $w(p) = [w_1(p), w_2(p), \ldots, w_m(p)]$, where $w_i(p)$ is calculated as a concatenation of link weights w_i of each link belonging to the path p. The proposed multi-criteria, k-shortest path routing algorithm finds a set of Pareto optimum paths, $f \in F$, between each pair of source to destination nodes. A given path is Pareto optimum if its path weights satisfy constraints: $w_i(f) < l_i, i = 1, \ldots, m$, where L is the vector of assumed constraints $L = [l_1, l_2, \ldots, l_m]$ and it is non-dominated within the scope of the considered objective functions. Note that proposed multi-criteria, k-shortest path routing algorithm runs off-line as a sub-process in CF network application. It is invoked in response to any changes in the VNI topology corresponding to: instantiation or release of a virtual link or a node, detection of any link or node failures as well as to update of SLA agreements.

The VNI control algorithm is invoked when a flow request arrives from the CF orchestration process. The algorithm is responsible for: (1) selection of a subset of feasible alternative routing paths which satisfy QoS requirements of

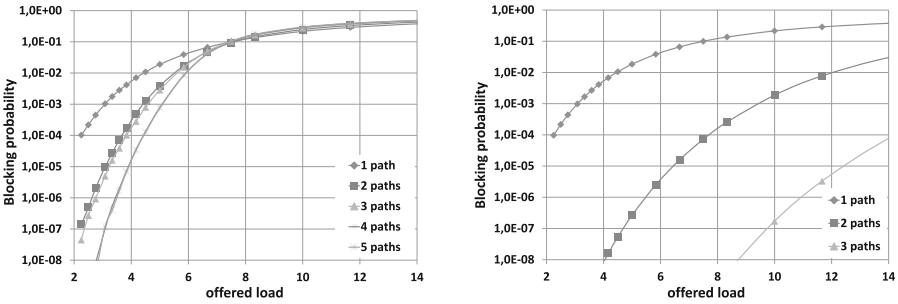
the requested flow. Notice, that bandwidth requested in the traffic descriptor may be satisfied by a number of alternative path assuming flow splitting among them, (2) allocation of the flow to selected feasible alternative routing paths, and (3) configuration of flow tables in virtual nodes on the selected path(s). The main objective of the proposed VNI control algorithm is to maximize the number of requests that are served with the success. This goal is achieved through smart allocation algorithm which efficiently use network resources. Remark, that flow allocation problem belongs to the NP-complete problems. The allocation algorithm has to take decision in a relatively short time (of second order) to not exceed tolerable request processing time. This limitation opt for using heuristic algorithm that find feasible solution in a reasonable time, although selected solution may not be the optimal one.

The proposed VNI control algorithm performs the following steps:

1. *Create a decision space.* In this step the algorithm creates a subset of feasible alternative paths that meet QoS requirements from the set of k-shortest routing paths. The algorithm matches QoS requirements with path weights $w(p)$. Then, it checks if selected subset of feasible alternative paths can meet bandwidth requirements, i.e. if the sum of available bandwidth on disjointed paths is greater than requested bandwidth. Finally, the algorithm returns the subset of feasible paths if the request is accepted or returns empty set $\emptyset$, which results in flow rejection.
2. *Allocate flow in VNI.* In this step, the algorithm allocates flow into previously selected subset of feasible paths. The allocation may address different objectives, as e.g. *load balancing, keeping the flow on a single path, etc.* depending on the CF strategy and policies. In the proposed algorithm, we allocate the requested flow on the shortest paths, using as much as possible limited number of alternative paths. So, we first try to allocate the flow on the latest loaded shortest path. If there is not enough bandwidth to satisfy demand, we divide the flow over other alternative paths following the load balancing principles. If we still need more bandwidth to satisfy the request, we consider longer alternative paths in consecutive steps. The process finishes when the requested bandwidth is allocated.
3. *Configure flow tables.* In the final step, the VNI control algorithm configures allocated paths using the abstract model of VNI maintained in the SDN controller. The actual configuration is performed by the management system of particular cloud using e.g. Open Flow protocol, net conf or other.

3.2.3 Performance Evaluation

The experiments focus on performance evaluation of the proposed VNI control algorithm. They are performed assuming a model of CF comprising n clouds offering the same set of services. A CF network assumes a full mesh topology where peering clouds are connected by virtual links. In this model the number of degree of freedom in selecting alternative paths is relatively large. Our experiments are performed by simulation. We simulate flow request arrival process and



(a) balanced load (b) unbalanced load

Fig. 7. Blocking probabilities of flow requests served by VNI using different number of alternative paths.

analyze the system performances in terms of request blocking probabilities. We analyze the effectiveness of the VNI control algorithm under the following conditions: (1) number of alternative paths established in VNI, and (2) balanced and unbalanced load conditions. Notice, that results related to a single path, denoted as *1 path*, correspond to the strategy based on choosing only direct virtual links between peering clouds, while other cases exploit multi-path routing capabilities offered by VNI.

Figure 7 presents exemplary results showing values of request blocking probabilities as a function of offered load obtained for VNI using different number of alternative paths. Figure 7a corresponds to balanced load conditions where each relation of source to destination is equally loaded in the network. Furthermore, Fig. 7b shows values of blocking probabilities for extremely unbalanced load conditions, where flows are established between a chosen single relation. One can observe that using VNI instead of direct communication between peering clouds leads to significant decreasing of blocking probabilities under wide range of the offered load up to the limit of the working point at blocking probability at the assumed level of 0.1. One can also observe that by using alternative paths we significantly increase carried traffic under the same blocking probability. Moreover, the gain from using alternative paths is mostly visible if we use the first alternative path. Increasing the number of alternative paths above four or five practically yields no further improvement. The gain becomes especially significant under unbalanced load conditions.

3.3 Level 3: Service Provision

Motivation. While traditionally a cloud infrastructure is located within a datacenter, recently, there is a need for geographical distribution [17]. For instance, cloud federation can combine the capabilities of multiple cloud offerings in order to satisfy the user's response time or availability requirements. Lately, this need for geo-distribution has led to a new evolution of decentralization. Most notably,

the extension of cloud computing towards the edge of the enterprise network, is generally referred to as fog or edge computing [18]. In fog computing, computation is performed at the edge of the network at the gateway devices, reducing bandwidth requirements, latency, and the need for communicating data to the servers. Second, mist computing pushes processing even further to the network edge, involving the sensor and actuator devices [19].

Compared to a traditional cloud computing environment, a geo-distributed cloud environment is less well-controlled and behaves in an ad-hoc manner. Devices may leave and join the network, or may become unavailable due to unpredictable failures or obstructions in the environment.

Additionally, while in a data-center heterogeneity is limited to multiple generations of servers being used, there is a large spread on capabilities within a geo-distributed cloud environment. Memory and processing means range from high (e.g. servers), over medium (e.g. cloudlets, gateways) to very low (e.g. mobile devices, sensor nodes). While some communication links guarantee a certain bandwidth (e.g. dedicated wired links), others provide a bandwidth with a certain probability (e.g. a shared wired link), and others do not provide any guarantees at all (wireless links).

Reliability is an important non-functional requirement, as it outlines *how* a software systems realizes its functionality [20]. The unreliability of substrate resources in a heterogeneous cloud environment, severely affects the reliability of the applications relying on those resources. Therefore, it is very challenging to host reliable applications on top of unreliable infrastructure [21].

Moreover, traditional cloud management algorithms cannot be applied here, as they generally consider powerful, always on servers, interconnected over wired links. Many algorithms do not even take into account bandwidth limitations. While such an omission can be justified by an appropriately over provisioned network bandwidth within a data-center, it is not warranted in the above described geo-distributed cloud networks.

State of the Art. In this section, the state of the art with regard to the Application Placement Problem (APP) in cloud environments is discussed. Early work on application placement merely considers nodal resources, such as Central Processing Unit (CPU) and memory capabilities. Deciding whether requests are accepted and where those virtual resources are placed then reduces to a Multiple Knapsack Problem (MKP) [22]. An MKP is known to be NP-hard and therefore optimal algorithms are hampered by scalability issues. A large body of work has been devoted to finding heuristic solutions [23–25].

When the application placement not only decides where computational entities are hosted, but also decides on how the communication between those entities is routed in the Substrate Network (SN), then we speak of *network-aware* APP. Network-aware application placement is closely tied to Virtual Network Embedding (VNE) [26]. An example of a network-aware approach is the work from Moens et al. [27]. It employs a Service Oriented Architecture (SOA), in which applications are constructed as a collection of communicating services. This optimal approach performs node and link mapping simultaneously.

In contrast, other works try to reduce computational complexity by performing those tasks in distinct phases [28,29].

While the traditional VNE problem assumes that the SN network remains operational at all times, the Survivable Virtual Network Embedding (SVNE) problem does consider failures in the SN. For instance, Ajtai et al. try and guarantee that a virtual network can still be embedded in a physical network, after k network components fail. They provide a theoretical framework for fault-tolerant graphs [30]. However, in this model, hardware failure can still result in service outage as migrations may be required before normal operation can continue.

Mihailescu et al. try to reduce network interference by placing Virtual Machines (VMs) that communicate frequently, and do not have anti-collocation constraints, on Physical Machines (PMs) located on the same racks [31]. Additionally, they uphold application availability when dealing with hardware failures by placing redundant VMs on separate server racks. A major shortcoming is that the number of replicas to be placed, and the anti-collocation constraints are user-defined.

Csorba et al. propose a distributed algorithm to deploy replicas of VM images onto PMs that reside in different parts of the network [32]. The objective is to construct balanced and dependable deployment configurations that are resilient. Again, the number of replicas to be placed is assumed predefined.

SiMPLE allocates additional bandwidth resources along multiple disjoint paths in the SN [33]. This proactive approach assumes splittable flow, i.e. the bandwidth required for a Virtual Link (VL) can be realized by combining multiple parallel connections between the two end points. The goal of SiMPLE is to minimize the total bandwidth that must be reserved, while still guaranteeing survivability against single link failures. However, an important drawback is that while the required bandwidth decreases as the number of parallel paths increases, the probability of more than one path failing goes up exponentially, effectively reducing the VL's availability.

Chowdhury et al. propose Dedicated Protection for Virtual Network Embedding (DRONE) [34]. DRONE guarantees Virtual Network (VN) survivability against single link or node failure, by creating two VNEs for each request. These two VNEs cannot share any nodes and links.

Aforementioned SVNE approaches [30–34] lack an availability model. When the infrastructure is homogeneous, it might suffice to say that each VN or VNE need a predefined number of replicas. However, in geo-distributed cloud environments the resulting availability will largely be determined by the exact placement configuration, as moving one service from an unreliable node to a more reliable one can make all the difference. Therefore, geo-distributed cloud environments require SVNE approaches which have a computational model for availability as a function of SN failure distributions and placement configuration.

The following cloud management algorithms have a model to calculate availability. Jayasinghe et al. model cloud infrastructure as a tree structure with arbitrary depth [35]. Physical hosts on which Virtual Machines (VMs) are hosted

are the leaves of this tree, while the ancestors comprise regions and availability zones. The nodes at bottom level are physical hosts where VMs are hosted. Wang et al. were the first to provide a mathematical model to estimate the resulting availability from such a tree structure [36]. They calculate the availability of a single VM as the probability that neither the leaf itself, nor any of its ancestors fail. Their work focuses on handling workload variations by a combination of vertical and horizontal scaling of VMs. Horizontal scaling launches or suspends additional VMs, while vertical scaling alters VM dimensions. The total availability is then the probability that at least one of the VMs is available. While their model suffices for traditional clouds, it is ill-suited for a geo-distributed cloud environment as link failure and bandwidth limitations are disregarded.

In contrast, Yeow et al. define reliability as the probability that critical nodes of a virtual infrastructure remain in operation over all possible failures [37]. They propose an approach in which backup resources are pooled and shared across multiple virtual infrastructures. Their algorithm first determines the required redundancy level and subsequently performs the actual placement. However, decoupling those two operations is only possible when link failure can be omitted and nodes are homogeneous.

Availability Model. In this section we introduce an availability model for geo-distributed cloud networks, which considers any combination of node and link failures, and supports both node and link replication. Then, building on this model, we will study the problem of guaranteeing a minimum level of availability for applications. In the next section, we introduce an Integer Linear Program (ILP) formulation of the problem. The ILP solver can find optimal placement configurations for small scale networks, its computation time quickly becomes unmanageable when the substrate network dimensions increase. Subsequently two heuristics are presented: (1) a distributed evolutionary algorithm employing a pool-model, where execution of computational tasks and storage of the population database (DB) are separated (2) a fast centralized algorithm, based on subgraph isomorphism detection. Finally, we evaluate the performance of the proposed algorithms.

3.3.0.1 Application Requests. We consider a SOA, which is a way of structuring IT solutions that leverage resources distributed across the network [38]. In a SOA, each application is described as its composition of services. Throughout this work, the collected composition of all requested applications will be represented by the instance matrix ($\boldsymbol{I}$).

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

3.3.0.2 Cloud Infrastructure. Consider a substrate network consisting of nodes and links. Nodes have certain CPU ($\boldsymbol{\Omega}$) and memory capabilities ($\boldsymbol{\Gamma}$). Physical links between nodes are characterized by a given bandwidth ($\boldsymbol{B}$). Both links and

Table 4. Overview of input variables to the Cloud Application Placement Problem (CAPP).

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

nodes have a known probability of failure, $\boldsymbol{p^N}$ and $\boldsymbol{p^E}$ respectively. Failures are considered to be independent.

3.3.0.3 The VAR Protection Method. Availability not only depends on failure in the SN, but also on how the application is placed. Non-redundant application placement assigns each service and VL at most once, while its redundant counterpart can place those virtual resources more than once. The survivability method presented in this work, referred to as VAR, guarantees a minimum availability by application level replication, while minimizing the overhead imposed by allocation of those additional resources. VAR uses a static failure model, i.e. availability only depends on the current state of the network. Additionally, it is assumed that upon failure, switching between multiple application instances takes place without any delay. These separate application instances will be referred to as duplicates. Immediate switchover yields a good approximation, when the duration of switchover is small compared to the uptime of individual components. A small switchover time is feasible, given that each backup service is preloaded in memory, and CPU and bandwidth resources have been preallocated. Furthermore, immediate switchover allows condensation of the exact failure dynamics of each component, into its expected availability value, as long as the individual components fail independently (a more limiting assumption).

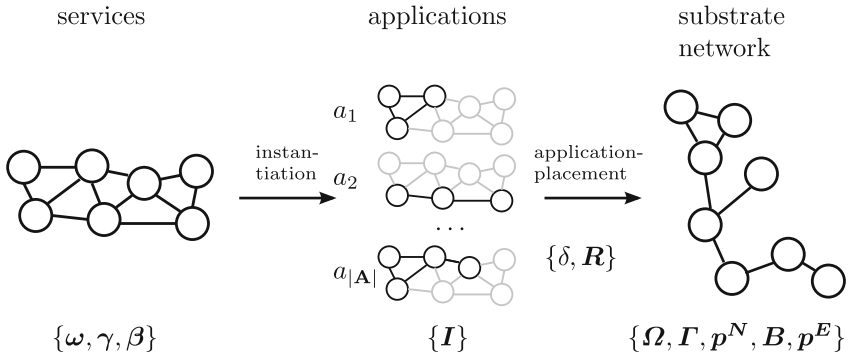


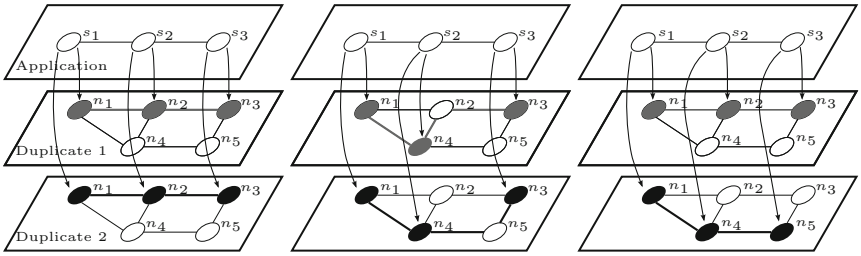
Fig. 8. Overview of this work: services $\{\boldsymbol{\omega}, \gamma, \boldsymbol{\beta}\}$, composing applications $\{\boldsymbol{I}\}$, are placed on a substrate network where node $\{\boldsymbol{p^N}\}$ and link failure $\{\boldsymbol{p^E}\}$ is modeled. By increasing the redundancy δ, a minimum availability $\boldsymbol{R}$ can be guaranteed.

Table 5. An overview of resource sharing amongst identical services and VLs.

	Sharing of resources		
	CPU	Memory	Bandwidth
Within application	Yes	Yes	Yes
Amongst applications	No	Yes	No

In the VAR model, an application is available if at least one of its duplicates is on-line. A duplicate is on-line if none of the PMs and Physical Links (PLs), that contribute its placement, fail. Duplicates of the same application can share physical components. An advantage of this reuse is that a fine-grained tradeoff can be made between increased availability, and decreased resource consumption. An overview of resources' reuse is shown in Table 5. In Fig. 9 three possible placement configurations using two duplicates are shown for one application. In Fig. 9a both duplicates are identical, and no redundancy is introduced. The nodal resource consumption is minimal, as CPU and memory for s_1, s_2, and s_3 are provisioned only once. Additionally, the total bandwidth required for (s_1, s_2), and (s_2, s_3) is only provisioned once. The bandwidth consumption of this configuration might not be minimal, if consolidation of two or three services onto one PM is possible. This placement configuration does not provide any fault-tolerance, as failure of either n_1, n_2 or n_3, or $(n_1, n_2), (n_2, n_3)$ results in downtime.

When more than one duplicate is placed and the resulting arrangements of VLs and services differ, then the placement is said to introduce redundancy. However, this increased redundancy results in a higher resource consumption. In Fig. 9b the application survives a singular failure of either (n_4, n_2), (n_2, n_3), (n_4, n_5), or (n_5, n_3). The placement configuration depicted in Fig. 9c survives all singular failures in the SN, except for a failure of n_1.



(a) 0 replicated services (b) 1 replicated VL (c) 2 replicated services replicated VL

Fig. 9. Illustration of the VAR protection method.

Formal Problem Description. The algorithms presented in this work are based on the optimisation model proposed in [39]. In this section we briefly describe the model but refer to [39] for a more elaborate discussion. Our model consists of two main blocks: the cloud-environment and the set of applications. To model the problem we define the following constraints. We refer to [39] for the mathematical representation.

- The total amount of duplicates for each application is limited by δ.
- An application a is placed correctly if and only if at least one duplicate of a is placed.
- A service is correctly placed if there is enough CPU and memory available in all PMs.
- A service will only be placed on a PM if and only if it is used by at least one duplicate.
- The total bandwidth of a PL cannot be higher than the aggregate bandwidth of the VLs that use the PL.
- A VL can use a PL if and only if the PL has sufficient remaining bandwidth.
- An application is only placed if the availability of the application can be guaranteed.

If a service is placed on the same PM, for multiple duplicates or for multiple applications, or the same VL is placed on a PL, they can reuse resources (see Table 5). Therefore, if service s is placed twice on PM n for the same application then there is no need to allocate CPU and memory twice. Only if service s is placed for a different application additional CPU resources must be allocated.

The problem we solve is to maximise the number of accepted applications.

Results. For a description of the proposed heuristics, and an extensive performance analysis, featuring multiple application types, SN types and scalability study we refer the interested reader to [40].

In reliable cloud environments (or equivalently, under low availability requirements) it is often acceptable to place each VN only once, and not bother about availability [27]. However, when the frequency of failures is higher (or if availability requirements increase), then one of the following measures should be

taken. First, one can improve the availability by placing additional backups, which fail independently of one another. However, this approach works best in homogeneous cloud environments, where one can use the same number of backup VN embeddings, regardless of the exact placement configuration. In heterogeneous environments a fixed redundancy level for each application either results in wasted SN resources, or a reduced placement ratio. In the context of cloud federation, the reliability of the links interconnecting the different cloud entities can be highly heterogeneous (leased lines, or best-effort public internet). Therefore, to further improve revenue, cloud federation should take these failure characteristics into consideration, and estimate the required replication level.

3.4 Level 2: Service Composition and Orchestration

Service composition and orchestration have become the predominant paradigms that enable businesses to combine and integrate services offered by third parties. For the commercial viability of composite services, it is crucial that they are offered at sharp price-quality ratios. A complicating factor is that many attractive third-party services often show highly variable service quality. This raises the need for mechanisms that promptly adapt the composition to changes in the quality delivered by third party services. In this section, we discuss a real-time QoS control mechanism that dynamically optimizes service composition in real time by learning and adapting to changes in third party service response time behaviors. Our approach combines the power of learning and adaptation with the power of dynamic programming. The results show that real-time service recompositions lead to dramatic savings of cost, while meeting the service quality requirements of the end-users.

3.4.1 Background and Motivation

In the competitive market of information and communication services, it is crucial for service providers to be able to offer services at competitive price/quality ratios. Succeeding to do so will attract customers and generate business, while failing to do so will inevitably lead to customer dissatisfaction, churn and loss of business. A complicating factor in controlling quality-of-service (QoS) in service oriented architectures is that the ownership of the services in the composition (sub-services) is decentralized: a composite service makes use of sub-services offered by third parties, each with their own business incentives. As a consequence, the QoS experienced by the (paying) end user of a composite service depends heavily on the QoS levels realized by the individual sub-services running on different underlying platforms with different performance characteristics: a badly performing sub-service may strongly degrade the end-to-end QoS of a composite service. In practice, service providers tend to outsource responsibilities by negotiating Service Level Agreements (SLAs) with third parties. However, negotiating multiple SLAs in itself is not sufficient to guarantee end-to-end QoS levels as SLAs in practice often give probabilistic QoS guarantees and SLA violations can still occur. Moreover probabilistic QoS guarantees do not necessarily capture time-dependent behavior e.g. short term service degradations.

Therefore, the negotiation of SLAs needs to be supplemented with *run-time QoS-control* capabilities that give providers of composite services the capability to properly respond to short-term QoS degradations (real-time composite service adaptation). Motivated by this, in this section we propose an approach that adapts to (temporary) third party QoS degradations by tracking the response time behavior of these third party services.

3.4.2 Literature and Related Work

The problem of QoS–aware optimal composition and orchestration of composite services has been well–studied (see e.g. [41,42]). The main problem addressed in these papers is how to select one concrete service per abstract service for a given workflow, in such a way that the QoS of the composite service (as expressed by the respective SLA) is guaranteed, while optimizing some cost function. Once established, this composition would remain unchanged the entire life–cycle of the composite web service. In reality, SLA violations occur relatively often, leading to providers' losses and customer dissatisfaction. To overcome this issue, it is suggested in [43–45] that, based on observations of the actually realised performance, re–composition of the service may be triggered. During the re–composition phase, new concrete service(s) may be chosen for the given workflow. Once re–composition phase is over, the (new) composition is used as long as there are no further SLA violations. In particular, the authors of [43–45] describe *when* to trigger such (re–composition) event, and *which adaptation actions* may be used to improve overall performance.

A number of solutions have been proposed for the problem of *dynamic, run–time* QoS–aware service selection and composition within SOA [46–49]. These (proactive) solutions aim to adapt the service composition dynamically at run–time. However, these papers do not consider the stochastic nature of response time, but its expected value. Or they do not consider the cost structure, revenue and penalty model as given in this paper.

In the next section, we extend the approach presented in [48] such that we can learn an exploit response-time distributions on the fly. The use of classical reinforcement-learning techniques would be a straight forward approach. However, our model has a special structure that complicates the use of the classical Temporal Difference learning (TD) learning approaches. The solution of our DP formulation searches the stochastic shortest path in a stochastic activity network [50]. This DP can be characterized as a hierarchical DP [51,52]. Therefore classical Reinforcement Learning (RL) is not suitable and hierarchical RL has to be applied [52]. Also changes in response-time behavior are likely to occur which complicates the problem even more. Both the problem structure and volatility are challenging areas of research in RL. Typically RL techniques solve complex learning and optimization problems by using a simulator. This involves a Q value that assigns utility to state–action combinations. Most algorithms run off-line as a simulator is used for optimization. RL has also been widely used in on–line applications. In such applications, information becomes available gradually with

time. Most RL approaches are based on environments that do not vary over time. We refer to [51] for a good survey on reinforcement learning techniques.

In our approach we tackle both the hierarchical structure, and time varying behavior challenges. To this end we are using empirical distributions and updating the lookup table if significant changes occur. As we are considering a sequence of tasks, the number of possible response time realizations combinations explodes. By discretizing the empirical distribution over fixed intervals we overcome this issue.

3.4.3 Composition and Orchestration Model

We consider a composite service that comprises a sequential workflow consisting of N tasks identified by $T_1, \ldots, T_N$. The tasks are executed one–by–one in the sense that each consecutive task has to wait for the previous task to finish. Our solution is applicable to any workflow that could be aggregated and mapped into a sequential one. Basic rules for aggregation of non–sequential workflows into sequential workflows have been illustrated in, e.g. [48,50,53]. However, the aggregation leads to coarser control, since decisions could not be taken for a single service within the aggregated workflow, but rather for the aggregated workflow patterns themselves.

The workflow is based on an unambiguous functionality description of a service ("abstract service"), and several functionally identical alternatives ("concrete services") may exist that match such a description [54]. Each task has an abstract service description or interface which can be implemented by external service providers.

The workflow in Fig. 10 consists of four abstract tasks, and each task maps to three concrete services (alternatives), which are deployed by (independent) third–party service providers. For each task T_i there are M_i concrete service providers $\mathrm{CS}^{(i,1)}, \ldots, \mathrm{CS}^{(i,M_i)}$ available that implement the functionality corresponding to task T_i. For each request processed by $\mathrm{CS}^{(i,j)}$ cost $c^{(i,j)}$ has to be paid. Furthermore there is an end–to–end response-time deadline δ_p. If a request is processed within δ_p a reward of R is received. However, for all requests that are not processed within δ_p a penalty V had to be paid. After the execution of a single task within the workflow, the orchestrator decides on the next concrete service to be executed, and composite service provider pays to the third party provider per single invocation. The decision points for given tasks are illustrated at Fig. 10 by A, B, C and D. The decision taken is based on (1) execution costs, and (2) the remaining time to meet the end–to–end deadline. The response time of each concrete service provider $\mathrm{CS}^{(i,j)}$ is represented by the random variable $D^{(i,j)}$. After each decision the observed response time is used for updating the response time distribution information of the selected service. Upon each lookup table update the corresponding distribution information is stored as reference distribution. After each response the reference distribution is compared against the current up-to date response time distribution information.

In our approach response-time realizations are used for learning an updating the response-time distributions. The currently known response-time distribution

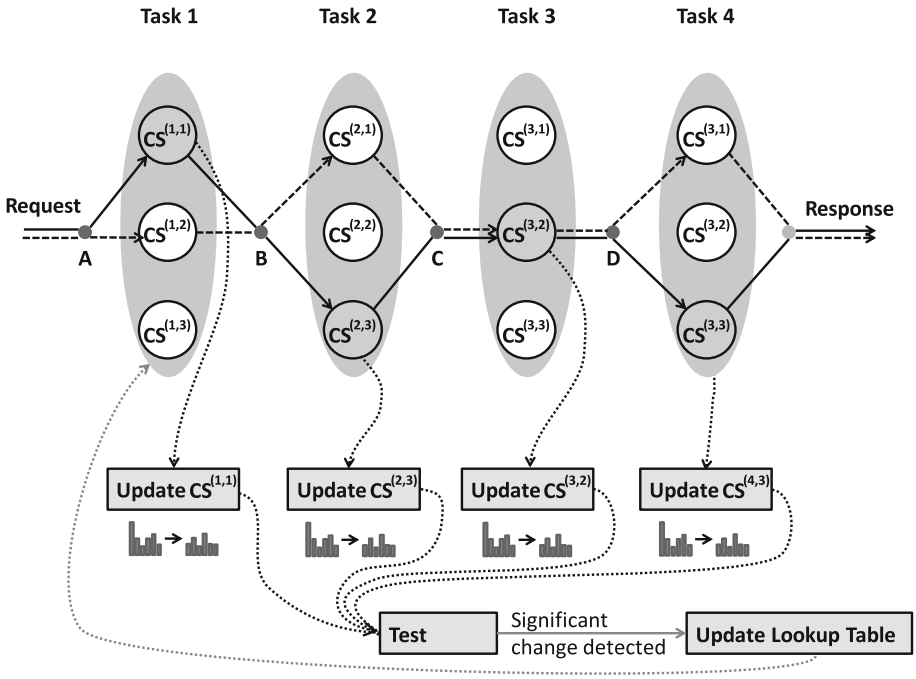


Fig. 10. Orchestrated composite web service depicted by a sequential workflow. Dynamic run–time service composition is based on a lookup table. Decisions are taken at points A–D. For every used concrete service the response-time distribution is updated with the new realization. In this example a significant change is detected. As a result for the next request concrete service 2 is selected at task 1.

is compared against the response-time distribution that was used for the last policy update. Using well known statistical tests we are able to identify if an significant change occurred and the policy has to be recalculated. Our approach is based on fully dynamic, run–time service selection and composition, taking into account the response–time commitments from service providers and information from response-time realizations. The main goal of this run–time service selection and composition is profit maximization for the composite service provider and ability to adapt to changes in response-time behavior of third party services.

By tracking response times the actual response-time behavior can be captured in empirical distributions. In [48] we apply a dynamic programming (DP) approach in order to derive a service-selection policy based on response-time realizations. With this approach it is assumed that the response-time distributions are known or derived from historical data. This results in a so called lookup table which determines what third party alternative should be used based on actual response-time realizations.

3.4.4 Real Time QoS Control

In this section we explain our real-time QoS control approach. The main goal of this approach is profit maximization for the composite service provider, and ability to adapt to changes in response-time behavior of third party services. We realize this by monitoring/tracking the observed response-time realizations. The currently known empirical response-time distribution is compared against the response-time distribution that was used for the last policy update. Using well known statistical tests we are able to identify if an significant change occurred and the policy has to be recalculated. Our approach is based on fully dynamic, run–time service selection and composition, taking into account the response–time commitments from service providers and information from response-time realizations. We illustrate our approach using Fig. 11. The execution starts with an initial lookup table at step (1). This could be derived from initial measurements on the system. After each execution of a request in step (2) the empirical distribution is updated at step (3). A DP based lookup table could leave out unattractive concrete service providers. In that case we do not receive any information about these providers. These could become attractive if the response-time behavior changes. Therefore in step (4), if a provider is not visited for a certain time, a probe request will be sent at step (5b) and the corresponding empirical distribution will be updated at step (6a). After each calculation of the lookup table, the current set of empirical distributions will be stored. These are the empirical distributions that were used in the lookup table calculation and form a reference response-time distribution. Calculating the lookup table for every new sample is expensive and undesired. Therefore we propose a strategy where the lookup table will be updated if a significant change in one of the services is detected. For this purpose the reference distribution is used for detection of response-time distribution changes. In step (5a) and step (6a) the reference distribution and current distribution are retrieved and a statistical test is applied for detecting change in the response-time distribution. If no change is detected then the lookup table remains unchanged. Otherwise the lookup table is updated using the DP. After a probe update in step (5b) and step (6b) we immediately proceed to updating the lookup table as probes are sent less frequently. In step (7) and step (8) the lookup table is updated with the current empirical distributions and these distributions are stored as new reference distribution. By using empirical distributions we are directly able to learn and adapt to (temporarily) changes in behavior of third party services.

Using a lookup table based on empirical distributions could result in the situation that certain alternatives are never invoked. When other alternatives break down this alternative could become attractive. In order to deal with this issue we use probes. A probe is a dummy request that will provide new information about the response time for that alternative. As we only receive updates from alternatives which are selected by the dynamic program, we have to keep track of how long ago a certain alternative has been used. For this purpose to each concrete service provider a probe timer $U^{(i,j)}$ is assigned with corresponding probe time–out $t_p^{(i,j)}$. If a provider is not visited in $t_p^{(i,j)}$ requests ($U^{(i,j)} > t_p^{(i,j)}$)

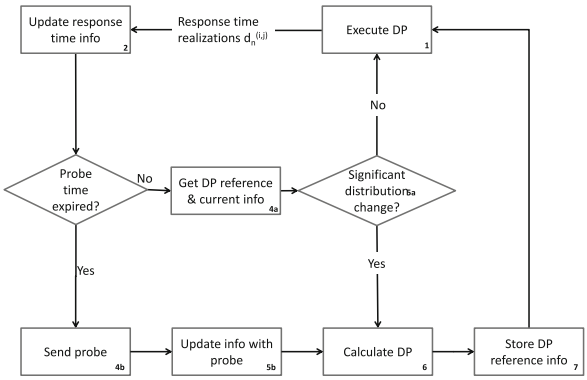


Fig. 11. Real-time QoS control approach.

then the probe timer has expired and a probe will be collected incurring probe cost $c_p^{(k,j)}$. If for example, in Fig. 10, the second alternative of the third task has not been used in the last ten requests, the probe timer for alternative two has value $U^{(3,2)} = 10$. After a probe we immediately update the corresponding distribution. No test is applied here as probes are collected less frequent compared to processed requests.

In order to evaluate the proposed QoS control methods we have performed extensive evaluation testing in an experimental setting. The results show that real-time service re-compositions indeed lead to dramatics savings in cost, while still meeting QoS requirements of the end users. The reader is referred to [55] for the details.

3.5 Level 1: Resource Management in Virtualized Infrastructure

Level 1 deals with the dependencies of different physical resources, such as Central Processing Unit (CPU) time, Random Access Memory (RAM), disk I/O, and network access, and their effect on the performance that users perceive. These dependencies can be described by functions that map resource combinations, i.e. resource vectors, to scalars that describe the performance that is achieved with these resources. Therefore, such *utility functions* describe how the combination of different resources influences the performance users perceive [56]. Accordingly, utility functions (a) indicate in which ratios resources have to be allocated, in order to maximize user satisfaction and efficiency, (b) are determined by technical factors, and (c) are investigated in this section.

3.5.1 Methodology

In order to get an idea about the nature of utility functions that VMs have during runtime, dependencies between physical resources, when utilized by VMs, and effects on VM performance are investigated as follows. Different workloads are

executed on a VM with a changing number of Virtual CPUs (VCPU) and Virtual RAM (VRAM) (this influences how many physical resources the VM can access) and varying load levels of the host system (this simulates contention among VMs and also influences how many physical resources the VM can access).

A machine with a 2.5 Gigahertz (GHz) AMD Opteron 6180 SE processor with 24 cores and 6 and 10 MB of level 2 and 3 cache, respectively, and 64 GB of ECC DDR3 RAM with 1333 Mhz is used as host system. VM and host have a x86-64 architecture and run Ubuntu 14.04.2 LTS, Trusty Tahr, which was the latest Ubuntu release, when the experiments were conducted.

3.5.1.1 Measurement Method. Resource consumption of VMs is measured by monitoring the VM's (qemu [57]) process. In particular, the VM's CPU time and permanent storage I/O utilization is measured with *psutil* (a python system and process utilities library) and the VM's RAM utilization by the VM's proportional set size, which is determined with the tool *smem* [58].

3.5.1.2 Workloads. Workloads are simulated by the following benchmarks of the Phoronix test suite [59].

Apache. This workload measures how many requests the Apache server can sustain concurrently.

Aio-stress. This benchmark assesses the speed of permanent storage I/O (hard disk or solid state drive). In a virtualized environment permanent storage can be cached in the host system's RAM. Therefore, this test not necessarily results in access to the host system's permanent storage.

7zip. This benchmark uses 7zip's integrated benchmark feature to measure the system's compression speed.

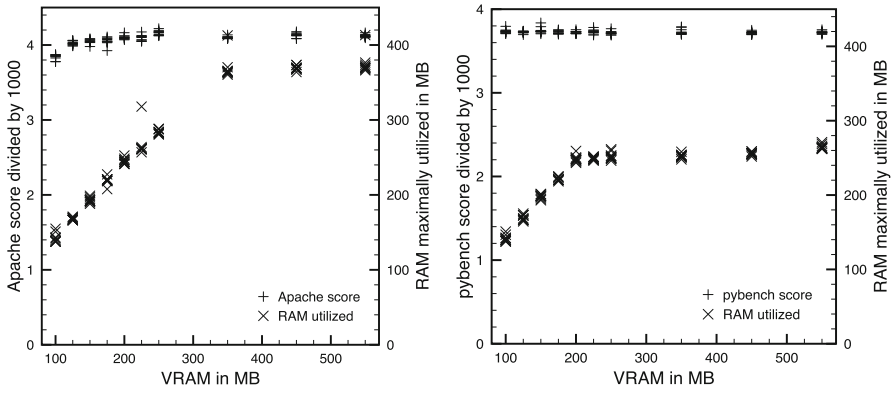
PyBench. This benchmark measures the execution time of Python functions such as BuiltinFunctionCalls and NestedForLoops. Contrary to all other benchmarks, here a lower score is better.

3.5.2 Results

This section presents selected results from [60] that were achieved with the setup described above.

3.5.2.1 RAM. Figure 12 shows the scores a VM achieves on the Apache and PyBench benchmark and the RAM it utilizes depending on the VRAM. For each VRAM configuration 10 measurements are conducted.

Figure 12a shows that when the VM executes Apache, it never utilizes more than 390 MB of RAM. In particular, for a VM with 100 to 350 MB of VRAM the amount of RAM that is maximally utilized continuously increases but does not further increase, when more than 350 MB of VRAM are added. Therefore, Fig. 12a shows that a VM with less than 350 MB of VRAM utilizes all RAM that is available, which seems to imply, that this amount of RAM is critical for performance. However, Fig. 12a also depicts that the Apache score only increases for up to 250 MB of VRAM and that this increase is marginal compared to the



(a) Apache benchmark (b) PyBench benchmark

Fig. 12. Benchmark scores and RAM utilization depending on a VM's VRAM

increase of RAM that is utilized. Therefore, the dependency between VRAM and utilized RAM is much stronger than the dependency between VRAM/utilized RAM and Apache score. In particular, while the RAM utilization more than doubles, the Apache scores vary by less than 10%. This is particularly interesting, because this configuration range includes 100 MB of VRAM which constrains the VM's RAM utilization to less than half of what the VM alone (without executing any workload) would utilize.

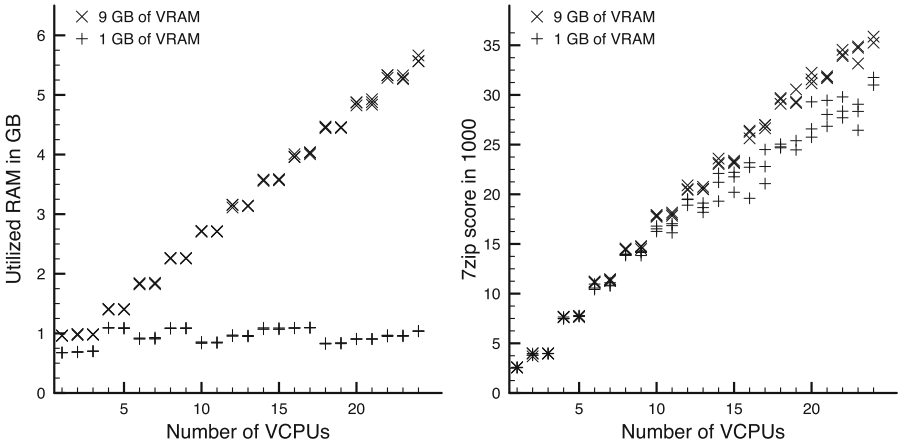
Figure 12b shows that when the VM executes PyBench, the VM process utilizes 270 MB of RAM at most. Although the VM is constraint in its RAM utilization, when it has less than 250 MB of VRAM, there is no correlation between the achieved PyBench score and the VM's VRAM, as the PyBench score does not increase.

Therefore, Fig. 12 shows that RAM, which is actively utilized by a VM (be it on startup or when executing an application), not necessarily impacts the VM's performance. In particular, even if the RAM utilized by a VM varies from 100 MB to 350 MB, the VM's Apache score, i.e., its ability to sustain concurrent server requests, only changed by 10%. For PyBench the score was entirely independent of the available RAM. This is particularly interesting, because not even a VM with 100 MB of VRAM showed decreased performance, while this is the minimum amount of RAM that avoids a kernel panic and even a VM that not executes any workload utilizes more, if possible.

3.5.2.2 VCPUs and Maximal RAM Utilization. The 7zip benchmark reveals an interesting dependency of VCPUs and RAM utilization (cf. Fig. 13). As Fig. 13a shows, for one to three VCPUs a VM executing the 7zip benchmark utilizes 1 GB of RAM and for every two additional cores the RAM utilization increases by 400 MB (the VM had 9 GB of VRAM).

The distinct pattern in which RAM is utilized gives reason to believe, that it is essential for performance. Therefore, Fig. 13b compares the 7zip scores

achieved by VMs with 1 and 9 GB of VRAM. As Fig. 13a shows, the more VCPUs a VM has, the more it will be constrained by only having 1 GB of VRAM, while 9 GB of VRAM not even constrain a VM with 24 VCPUs. In line with this observation, Fig. 13b shows that the difference between the 7zip scores achieved by VMs with 1 and 9 GB of VRAM grows with the number of VCPUs. However, the score difference is rather moderate compared to the large difference in terms of RAM utilization. In particular, a VM with 24 VCPUs utilizes more than 5 GB of RAM, if available. This is five times as much, as a VM with 1 GB of VRAM utilizes. However, the 7zip scores achieved by these VMs only differ by 15%.

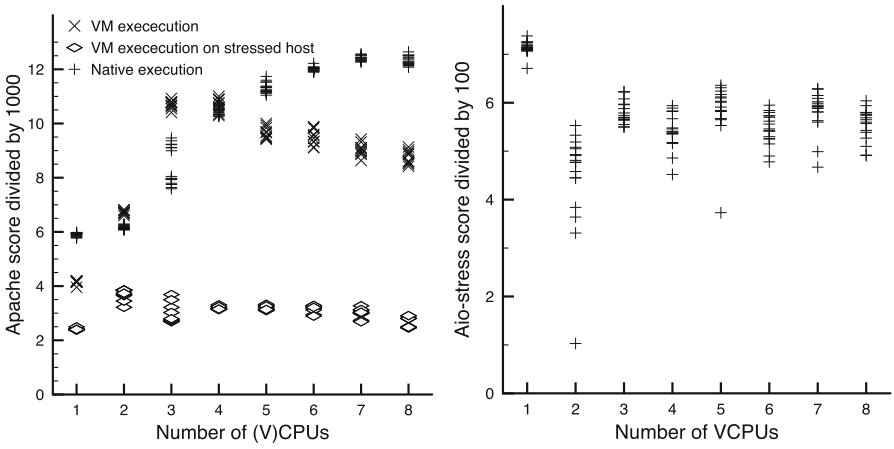


(a) RAM utilization (b) Performance

Fig. 13. RAM utilization and performance, depending on the number of VCPUs and amount of VRAM, of a VM executing the 7zip benchmark

3.5.2.3 Multi Core Penalty. Figure 14a plots the Apache scores achieved by a VM with 1 to 9 VCPUs, whereat 16 measurements per configuration were conducted. The figure shows that the best performance is achieved, when the VM has three or four VCPUs, while additional VCPUs linearly decrease the Apache score. As the figure depicts, up to three VCPUs significantly increase performance and four VCPUs perform equally well. However, adding additional VCPUs continuously decreases performance. This effect, which is termed *multi-core-penalty* occurred, independent of whether VCPUs were pinned to physical CPUs. Figure 14a also demonstrates that, while three VCPUs perform best for an unstressed host, two VCPUs perform best, when the host is stressed. Furthermore, the multi-core-penalty does not occur, when the benchmark is executed natively, i.e., directly on the host and not inside a VM. This shows that the it is caused by the virtualization layer. Despite the decrease of the Apache score with the number of VCPUs, the VM's utilization of CPU time increases with the number of VCPUs. For example, for the Apache benchmark it was found that for 9 VCPUs the

utilized CPU time is roughly twice as high as the CPU time utilized by one to three VCPUs (although the Apache score was significantly lower for 9 VCPUs).



(a) Apache scores achieved with different numbers of cores

(b) Aio-stress scores achieved with different numbers of VCPUs

Fig. 14. Two example of the multi-core-penalty

Figure 14b shows that the multi-core penalty also occurs for the aio-stress benchmark, where a VM with one VCPU constantly achieves a higher aio-stress score than any VM with more VCPUs. In particular, the aio-stress score of a VM with only one VCPU is on average a 30% higher than the aio-stress score of VMs with more VCPUs. However, unlike the Apache benchmark, the aio-stress score does not decrease with the number of VCPUs.

3.5.3 New Findings

Most work on data center resource allocation assumes that resources such as CPU and RAM are required in static or at least well defined ratios and that the resulting performance is clearly defined. The results of this section do not confirm these idealistic assumptions.

Section 3.5.2 did not find any significant effect of a VRAM on VM performance. Notably, even for workloads that seem to be RAM critical, as they utilize RAM in distinct patterns, or workloads running on VMs with just enough VRAM to avoid a kernel panic during boot, no significant effect was found. Even if a lack of RAM impedes performance, the impediment is minor compared to the amount of RAM that is missing (cf. Sect. 3.5.2). In contrast, a lack of RAM bandwidth significantly effects performance [61] but is rarely considered, when investigating data center fairness. Section 3.5.2 showed that the amount of RAM that is utilized by a VM may depend on the number of VCPUs. Section 3.5.2 presents the most counter-intuitive finding, which is that, when multi-core benchmarks

are executed inside a VM, the performance often decreases, when more VCPUs are added to the VM.

This section showed that it is a complex task to determine a class of utility functions that properly models the allocation of a node's PRs to VMs. However, a realistic class of utility functions would greatly aid cloud resource allocation, as it would allow to theoretically determine allocations that are practically more efficient. Therefore, positive results on this topic would also greatly aid the performance of cloud federations, as it would also allow to execute tasks in the cloud of a federation, that performs best for this task. Nonetheless, no work exists on this topic. This lack of work is caused by the topic's complexity. For example, resource dependencies vary over time, and depend on the workload that is executed inside a VM and the host's architecture. Also, the performance of a VM is determined by a combination of resources as diverse as CPU time, RAM, disk I/O, network access, CPU cache capacity, and memory bandwidth, where substitutabilities may or may not apply.

4 Cloud Federation for IoT

4.1 State-of-the-Art in IoT Cloud Research

The integration of IoT and clouds has been envisioned by Botta et al. [62] by summarizing their main properties, features, underlying technologies, and open issues. A solution for merging IoT and clouds is proposed by Nastic et al. [63]. They argued that system designers and operations managers faced numerous challenges to realize IoT cloud systems in practice, due to the complexity and diversity of their requirements in terms of IoT resources consumption, customization and runtime governance. They also proposed a novel approach for IoT cloud integration that encapsulated fine-grained IoT resources and capabilities in well-defined APIs in order to provide a unified view on accessing, configuring and operating IoT cloud systems, and demonstrated their framework for managing electric fleet vehicles.

Atzori et al. [64,65] examined IoT systems in a survey. They identified many application scenarios, and classified them into five application domains: transportation and logistics, healthcare, smart environments (home, office, plant), personal, social and futuristic domains. They described these domains in detail, and defined open issues and challenges for all of them. Concerning privacy, they stated that much sensitive information about a person can be collected without their awareness, and its control is impossible with current techniques.

Escribano [66] discussed the first opinion [67] of the Article 29 Data Protection Working Party (WP29) on IoT. According to these reports four categories can be differentiated: the first one is wearable computing, which means the application of everyday objects and clothes, such as watches and glasses, in which sensors were included to extend their functionalities. The second category is called the 'quantified self things', where things can also be carried by individuals to record information about themselves. With such things we can examine physical activities, track movements, and measure weight, pulse or other health indicators. The third one is home automation, which covers applications using

devices placed in offices or homes such as connected light bulbs, thermostats, or smoke alarms that can be controlled remotely over the Internet. They also mention smart cities as the fourth category, but they do not define them explicitly. They argue that sharing and combining data through clouds will increase locations and jurisdictions, where personal data resides. Therefore it is crucial to identify and realize which stakeholder is responsible for data protection. WP29 named many challenges concerning privacy and data protection, like lack of user control, intrusive user profiling and communication and infrastructure related security risks.

IoT application areas and scenarios have already been categorized, such as by Want et al. [68], who set up three categories: Composable systems, which are ad-hoc systems that can be built from a variety of nearby things by making connections among these possibly different kinds of devices. Since these devices can discover each other over local wireless connections, they can be combined to provide higher-level capabilities. Smart cities providing modern utilities could be managed more efficiently with IoT technologies. As an example traffic-light systems can be made capable of sensing the location and density of cars in the area, and optimizing red and green lights to offer the best possible service for drivers and pedestrians. Finally, resource conservation scenarios, where major improvements can be made in the monitoring and optimization of resources such as electricity and water.

4.2 MobIoTSim for Simulating IoT Devices

Cloud Federation can help IoT systems by providing more flexibility and scalability. Higher level decisions can be made on where to place a gateway service to receive IoT device messages, e.g. in order to optimize resource usage costs and energy utilization. Such complex IoT cloud systems can hardly be investigated in real world, therefore we need to turn to simulations.

The main purpose of MobIoTSim [69], our proposed mobile IoT device simulator, is to help cloud application developers to learn IoT device handling without buying real sensors, and to test and demonstrate IoT applications utilizing multiple devices. The structure of the application lets users create IoT environment simulations in a fast and efficient way that allows for customization.

MobIoTSim can simulate one or more IoT devices, and it is implemented as a mobile application for the Android platform. Sensor data generation of the simulated devices are random generated values in the range given by the user, or replayed data from trace files. The data sending frequency can also be specified for every device. The application uses the MQTT protocol to send data with the use of the Eclipse Paho opensource library. The data is represented in a structured JSON object compatible with the IBM IoT Foundation message format [70].

The basic usage of the simulator is to (i) connect to a cloud gateway, where the data is to be sent, (ii) create and configure the devices to be simulated and (iii) start the (data generation of the) required devices. These main steps are represented by three main parts of the application: the *Cloud settings*, the *Devices* and the *Device settings* screens. In the *Cloud settings* screen, the user

can set the required information about the targeted cloud, where the data will be received and processed. Currently there are two types of clouds supported: IBM Bluemix and MS Azure. For the IBM cloud we have two options: the Bluemix quickstart and the standard Bluemix IoT service. The Bluemix quickstart is a public demo application, it can visualise the data from a selected device. For a fast and easy setup (i.e. to try out the simulator) this type is recommended. The standard Bluemix IoT service type can be used if the user has a registered account for the Bluemix platform, and already created an IoT service. This IoT service can be used to handle devices, which have been registered before. The main part of the IoT service is an MQTT broker, this is the destination of the device messages, and it forwards them to the cloud applications. Such cloud applications can process the data, react to it or just perform some visualisation. The required configuration parameters for the standard Bluemix IoT service in MobIoTSim are: the Organization ID, which is the identifier of the IoT service of the user in Bluemix, and an authentication key, so that the user does not have to register the devices on the Bluemix web interface, and the command and event IDs, which are customizable parts of the used MQTT topics to send messages from the devices to the cloud and vice versa. MobIoTSim can register the created devices with these parameters automatically, by using the REST interface of Bluemix.

The *Devices* screen lists the created devices, where every row is a device or a device group. These devices can be started and stopped by the user at will, both together or separately for the selected ones. Some devices have the ability to display warnings and notifications sent back by a gateway. In this screen we can also create new devices or device groups. There are some pre-defined device templates, which can be selected for creation. These device templates help to create often used devices, such as a temperature sensor, humidity sensor or a thermostat. If the user selects a template for the base of the device, the message content and frequency will be set to some predefined values. The *Thermostat* template has a temperature parameter, it turns on by reaching a pre-defined low-level value and turns off at the high-level value. The On/Off state of the device is displayed all the time. It is possible to select the *Custom* template to configure a device in detail.

The new device creation and the editing of an existing one are made in the *Device settings* screen. The user can add more parameters to a device and can customize it with its own range. The range will be used to generate random values for the parameters. A device group is a group of devices with the same base template and they can be started and stopped together. If a device wants to send data to the Bluemix IoT service, it has to be registered beforehand. The registered devices have device IDs and tokens for authentication. The MobIoT-Sim application handles the device registration in the cloud with REST calls, so the user does not have to register the devices manually on the graphical web interface. There is an option to save the devices to a file and load them back to the application later. The device type attribute can be used to group devices. The simulation itself can also be saved, so the randomly generated data can be

replayed later many times. Even trace files from real world applications can be played from other sources, i.e. saved samples from the OpenWeatherMap public weather data provider [71]. The OpenWeatherMap monitors many cities and stores many parameters for them, including temperature, humidity, air pressure and wind speed. Using this trace loader feature, the simulation becomes closer to a real life scenario. In some cases, the user may want to send data to not just one but more cloud gateways at the same time. This is also possible by changing the organization ID attribute of a device to one of the already saved ones in the cloud settings.

We modified the Bluemix visualisation application to create a new private gateway to handle more than one device at the same time. In this way we can see the data from all devices in a real time chart. The node.js application subscribes to all device topics with the MQTT protocol, and waits for the data. In this revised gateway we use paging to overcome device management limitations (25 devices at a time). In order to enhance and better visualize many device data at the same time, we introduced device grouping for the chart generation.

To summarize, MobIoTSim together with the proposed gateways provide a novel solution to enable the simulation and experimentation of IoT cloud systems. Our future work will address extensions for additional thing and sensor templates, and will provide cases for scalability investigations involving multiple cloud gateways.

5 Summary

In this chapter we have reported activities of the COST IC1304 ACROSS European Project corresponding to traffic management for Cloud Federation. In particular, we have provided survey of discussed CF architectures and corresponding standardization activities, we have proposed comprehensive multi-level model for traffic management for CF together with proposed solutions for each level. The effectiveness of these solutions were verified by simulation and analytical methods. The proposed levels are: Level 5 - Strategies for building CF, Level 4 - Network for CF, Level 3 - Service specification and provision, Level 2 - Service composition and orchestration, Level 1 - Task service in cloud resources. Finally, we have presented specialized simulator for testing CF solution in IoT environment.

References

1. Schubert, L., Jeffery, K.: Advances in Clouds - Research in Future Cloud Computing, Report from the Cloud Computing Expert Working Group Meeting. Cordis (Online), BE: European Commission (2012). http://cordis.europa.eu/fp7/ict/ssai/docs/future-cc-2may-finalreport-experts.pdf
2. Grozev, N., Buyya, R.: Inter-cloud architectures and application brokering: taxonomy and survey. Softw. Pract. Exper. (2012). https://doi.org/10.1002/spe.2168

3. Celesti, A., Tusa, F., Villari, M., Puliafito, A.: How to enhance cloud architectures to enable cross-federation. In: Proceedings of the 3rd International Conference on Cloud Computing (CLOUD 2010), Miami, Florida, USA, pp. 337–345. IEEE (2010)
4. Bernstein, D., Ludvigson, E., Sankar, K., Diamond, S., Morrow, M.: Blueprint for the intercloud - protocols and formats for cloud computing interoperability. In: Proceedings of the Fourth International Conference on Internet and Web Applications and Services, pp. 328–336 (2009)
5. Marosi, A.C., Kecskemeti, G., Kertesz, A., Kacsuk, P.: FCM: an architecture for integrating IaaS cloud systems. In: Proceedings of the Second International Conference on Cloud Computing, GRIDs, and Virtualization (Cloud Computing 2011), IARIA, pp. 7–12, Rome, Italy (2011)
6. International Telecommunication Union (ITU-T): Framework of Inter-Could Computing (2014)
7. Internet Engineering Task Force (IETF): Working group on Content Delivery Network Interconnection (CDNI) (2011)
8. National Institute of Standards and Technology [NIST]: U.S. Dept. of Commerce, NIST Cloud Computing Standards Roadmap, Spec. Publ. 500–291 (2013)
9. Institute of electrical and electronics engineering (IEEE): Inter-cloud working group, Standard for Intercloud Interoperability and Federation (SIIF) (2017)
10. Darzanos, G., Koutsopoulos, I., Stamoulis, G.D.: Economics models and policies for cloud federations. In: 2016 IFIP Networking Conference (IFIP Networking) and Workshops, Vienna, pp. 485–493 (2016). https://doi.org/10.1109/IFIPNetworking.2016.7497246
11. Samaan, N.: A novel economic sharing model in a federation of selfish cloud providers. IEEE Trans. Parallel Distrib. Syst. **25**(1), 12–21 (2014). https://doi.org/10.1109/TPDS.2013.23
12. Kleinrock, L.: Queueing Systems Volume 1: Theory, p. 103. Wiley, Hoboken (1975). ISBN 0471491101
13. Carlini, E., Coppola, M., Dazzi, P., Ricci, L., Righetti, G.: Cloud federations in contrail. In: Alexander, M., et al. (eds.) Euro-Par 2011. LNCS, vol. 7155, pp. 159–168. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29737-3_19
14. Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., Zolla, J., Hölzle, U., Stuart, S., Vahdat, A.: B4: experience with a globally-deployed software defined WAN. In: ACM SIGCOMM 2013 Conference, New York, USA (2013)
15. Yen, J.Y.: Finding the K shortest loopless paths in a network. Manag. Sci. JSTOR **17**(11), 712–716 (1971). www.jstor.org/stable/2629312
16. Aljazzar, H., Leue, S.: K*: a heuristic search algorithm for finding the k shortest paths. Artif. Intell. **175**(18), 2129–2154 (2011). https://doi.org/10.1016/j.artint.2011.07.003. ISSN 0004–3702
17. Puleri, M., Sabella, R.: Cloud robotics: 5G paves the way for mass-market autmation. In: Charting the Future of Innovation, 5th edn., vol. 93, Ericsson, Stockholm (2016)
18. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the Internet of Things. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, pp. 13–16. ACM (2012). https://doi.org/10.1145/2342509.2342513

19. Al-Muhtadi, J., Campbell, R., Kapadia, A., Mickunas, M.D., Yi, S.: Routing through the mist: privacy preserving communication in ubiquitous computing environments. In: Proceedings 22nd International Conference on Distributed Computing Systems, pp. 74–83 (2002). https://doi.org/10.1109/ICDCS.2002.1022244. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1022244
20. ISO/IEC-25010: Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models, Standard, International Organization for Standardization, Geneva, CH, March 2010
21. Spinnewyn, B., Latré, S.: Towards a fluid cloud: an extension of the cloud into the local network. In: Latré, S., Charalambides, M., François, J., Schmitt, C., Stiller, B. (eds.) AIMS 2015. LNCS, vol. 9122, pp. 61–65. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20034-7_7
22. Camati, R., Calsavara, A., Lima Jr., L.: Solving the virtual machine placement problem as a multiple multidimensional Knapsack problem. In: ICN 2014, no. c, pp. 253–260 (2014). https://www.thinkmind.org/download.php?articleid=icn_2014_11_10_30065
23. Xu, J., Fortes, J.A.B.: Multi-objective virtual machine placement in virtualized data center environments. In: 2010 IEEE/ACM International Conference on \& International Conference on Cyber, Physical and Social Computing (CPSCom), GREENCOM-CPSCOM 2010, IEEE Computer Society, Washington, DC, USA, pp. 179–188 (2010). https://doi.org/10.1109/GreenCom-CPSCom.2010.137
24. Ren, Y., Suzuki, J., Vasilakos, A., Omura, S., Oba, K.: Cielo: an evolutionary game theoretic framework for virtual machine placement in clouds. In: Proceedings - 2014 International Conference on Future Internet of Things and Cloud, FiCloud 2014, pp. 1–8 (2014). https://doi.org/10.1109/FiCloud.2014.11
25. Moens, H., Truyen, E., Walraven, S., Joosen, W., Dhoedt, B., De Turck, F.: Cost-effective feature placement of customizable multi-tenant applications in the cloud. J. Netw. Syst. Manag. **22**(4), 517–558 (2014). https://doi.org/10.1007/s10922-013-9265-5
26. Fischer, A., Botero, J.F., Beck, M.T., De Meer, H., Hesselbach, X.: Virtual network embedding: a survey. IEEE Commun. Surv. Tutor. **15**(4), 1888–1906 (2013). https://doi.org/10.1109/SURV.2013.013013.00155. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6463372
27. Moens, H., Hanssens, B., Dhoedt, B., De Turck, F.: Hierarchical network-aware placement of service oriented applications in clouds. In: IEEE/IFIP NOMS 2014 - IEEE/IFIP Network Operations and Management Symposium: Management in a Software Defined World, pp. 1–8 (2014). https://doi.org/10.1109/NOMS.2014.6838230
28. Cheng, X., Su, S., Zhang, Z., Wang, H., Yang, F., Luo, Y., Wang, J.: Virtual network embedding through topology-aware node ranking. ACM SIGCOMM Comput. Commun. Rev. **41**(2), 38 (2011). https://doi.org/10.1145/1971162.1971168
29. Zhu, Y., Ammar, M.: Algorithms for assigning substrate network resources to virtual network components. In: Proceedings - IEEE INFOCOM, pp. 1–12 (2006). https://doi.org/10.1109/INFOCOM.2006.322
30. Ajtai, M., Alon, N., Bruck, J., Cypher, R., Ho, C., Naor, M., Szemeredi, E.: Fault tolerant graphs, perfect hash functions and disjoint paths. In: Proceedings, 33rd Annual Symposium on Foundations of Computer Science, pp. 693–702 (1992). https://doi.org/10.1109/SFCS.1992.267781. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=267781

31. Mihailescu, M., Sharify, S., Amza, C.: Optimized application placement for network congestion and failure resiliency in clouds. In: 2015 IEEE 4th International Conference on Cloud Networking, CloudNet 2015, pp. 7–13 (2015). https://doi.org/10.1109/CloudNet.2015.7335272
32. Csorba, M.J., Meling, H., Heegaard, P.E.: Ant system for service deployment in private and public clouds. In: Proceeding of the 2nd Workshop on Bio-inspired Algorithms for Distributed Systems - BADS 2010, p. 19. ACM (2010). https://doi.org/10.1145/1809018.1809024. http://portal.acm.org/citation.cfm?doid=1809018.1809024
33. Khan, M.M.A., Shahriar, N., Ahmed, R., Boutaba, R.: SiMPLE: survivability in multi-path link embedding. In: Proceedings of the 11th International Conference on Network and Service Management, CNSM 2015, pp. 210–218 (2015). https://doi.org/10.1109/CNSM.2015.7367361
34. Chowdhury, S., Ahmed, R., Alamkhan, M.M., Shahriar, N., Boutaba, R., Mitra, J., Zeng, F.: Dedicated protection for survivable virtual network embedding. In: IEEE Transactions on Network and Service Management, p. 1 (2016). https://doi.org/10.1109/TNSM.2016.2574239. http://ieeexplore.ieee.org/document/7480798/
35. Jayasinghe, D., Pu, C., Eilam, T., Steinder, M., Whalley, I., Snible, E.: Improving performance and availability of services hosted on IaaS clouds with structural constraint-aware virtual machine placement. In: Proceedings - 2011 IEEE International Conference on Services Computing, SCC 2011, pp. 72–79. IEEE (2011). https://doi.org/10.1109/SCC.2011.28
36. Wang, W., Chen, H., Chen, X.: An availability-aware virtual machine placement approach for dynamic scaling of cloud applications. In: Proceedings - IEEE 9th International Conference on Ubiquitous Intelligence and Computing and IEEE 9th International Conference on Autonomic and Trusted Computing, UIC-ATC 2012, pp. 509–516 (2012). https://doi.org/10.1109/UIC-ATC.2012.31
37. Yeow, W.-L., Westphal, C., Kozat, U.: Designing and embedding reliable virtual infrastructures. In: Proceedings of the Second ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures - VISA 2010, vol. 41(2), p. 33 (2010) . arXiv:1005.5367. https://doi.org/10.1145/1851399.1851406. http://portal.acm.org/citation.cfm?doid=1851399.1851406
38. Laskey, K.B., Laskey, K.: Service oriented architecture. Wiley Interdisc. Rev. Comput. Stat. **1**(1), 101–105 (2009). https://doi.org/10.1002/wics.8
39. Spinnewyn, B., Braem, B., Latre, S.: Fault-tolerant application placement in heterogeneous cloud environments. In: Proceedings of the 11th International Conference on Network and Service Management, CNSM 2015, pp. 192–200. IEEE (2015). https://doi.org/10.1109/CNSM.2015.7367359
40. Spinnewyn, B., Mennes, R., Botero, J.F., Latre, S.: Resilient application placement for geo-distributed cloud networks. J. Netw. Comput. Appl. **85**(1), 14–31 (2017). https://doi.org/10.1016/j.jnca.2016.12.015
41. Canfora, G., Di Penta, M., Esposito, R., Villani, M.L.: An approach for QoS-aware service composition based on genetic algorithms. In: Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, pp. 1069–1075. ACM (2005)
42. Yu, T., Zhang, Y., Lin, K.J.: Efficient algorithms for web services selection with end-to-end QoS constraints. ACM Trans. Web (TWEB) **1**, 6 (2007). ACM
43. Canfora, G., Di Penta, M., Esposito, R., Villani, M.L.: A framework for QoS-aware binding and re-binding of composite web services. J. Syst. Softw. **81**, 1754–1769 (2008). Elsevier

44. Zeng, L., Lingenfelder, C., Lei, H., Chang, H.: Event-driven quality of service prediction. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 147–161. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89652-4_14
45. Leitner, P.: Ensuring cost-optimal SLA conformance for composite service providers. ICSOC/ServiceWave 2009. Ph.D. symposium, p. 49 (2009)
46. Cardellini, V., Casalicchio, E., Grassi, V., Lo Presti, F.: Adaptive management of composite services under percentile-based service level agreements. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 381–395. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17358-5_26
47. Gao, A., Yang, D., Tang, S., Zhang, M.: Web service composition using Markov decision processes. In: Fan, W., Wu, Z., Yang, J. (eds.) WAIM 2005. LNCS, vol. 3739, pp. 308–319. Springer, Heidelberg (2005). https://doi.org/10.1007/11563952_28
48. Živković, M., Bosman, J.W., van den Berg, J.L., van der Mei, R.D., Meeuwissen, H.B., Núnez-Queija, R.: Run-time revenue maximization for composite web services with response time commitments. In: 2012 IEEE 26th International Conference on Advanced Information Networking and Applications (AINA), pp. 589–596. IEEE (2012)
49. Doshi, P., Goodwin, R., Akkiraju, R., Verma, K.: Dynamic workflow composition using Markov decision processes. Int. J. Web Serv. Res. **2**, 1–17 (2005)
50. Choudhury, G.L., Houck, D.J.: Combined queuing and activity network based modeling of sojourn time distributions in distributed telecommunication systems. In: Labetoulle, J., Roberts, J.W. (eds.) The Fundamental Role of Teletraffic in the Evolution of Telecommunications Networks, Proceedings ITC, vol. 14, pp. 525–534 (1994)
51. Gosavi, A.: Reinforcement learning: a tutorial survey and recent advances. INFORMS J. Comput. **21**, 178–192 (2009)
52. Barto, A.G., Mahadeva, S.: Recent advances in hierarchical reinforcement learning. Discrete Event Dyn. Syst. **13**, 341–379 (2004). https://doi.org/10.1023/A:1022140919877
53. Zheng, H., Zhao, W., Yang, J., Bouguettaya, A.: QoS analysis for web service composition. In: 2009 IEEE International Conference on Services Computing, pp. 235–242. IEEE (2009)
54. Preist, C.: A conceptual architecture for semantic web services. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 395–409. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30475-3_28
55. Bosman, J.W., van den Berg, J.L., van der Mei, R.D.: Real-time QoS control for service orchestration. In: 27-th International Teletraffic Congress, Ghent, Belgium (2015)
56. Poullie, P., Bocek, T., Stiller, B.: A survey of the state-of-the-art in fair multi-resource allocations for data centers. IEEE Trans. Netw. Serv. Manag. **15**(1), 169–183 (2017). TNSM 2017
57. Bellard, F.: QEMU, a fast and portable dynamic translator. In: Annual Conference on USENIX Annual Technical Conference, ATEC 2005, p. 41, Anaheim, CA, USA (2005)
58. Selenic Consulting: smem memory reporting tool. https://www.selenic.com/smem/. Accessed 7 Feb 2017
59. Phoronix Media: Phoronix test suite (2017). http://www.phoronix-test-suite.com. Accessed 18 Jan 2017

60. Poullie, P.: Decentralized multi-resource allocation in clouds. Dissertation, University of Zurich, Zurich, Switzerland, September 2017
61. Gruhler, A.L.: Investigation of resource reallocation capabilities of KVM and OpenStack. Bachelor Thesis, Universität Zürich, Zurich, Switzerland, August 2015. https://files.ifi.uzh.ch/CSG/staff/poullie/extern/theses/BAgruhler.pdf
62. Botta, A., de Donato, W., Persico, V., Pescape, A.: On the integration of cloud computing and Internet of Things. In: The 2nd International Conference on Future Internet of Things and Cloud (FiCloud-2014), August 2014
63. Nastic, S., Sehic, S., Le, D., Truong, H., Dustdar, S.: Provisioning software-defined IoT cloud systems. In: The 2nd International Conference on Future Internet of Things and Cloud (FiCloud-2014), August 2014
64. Atzori, L., Iera, A., Morabito, G.: The Internet of Things: a survey. Comput. Netw. **54**(15), 2787–2805 (2010)
65. Farris, I., Militano, L., Nitti, M., Atzori, L., Iera, A.: MIFaaS: a Mobile-IoT-Federation-as-a-Service model for dynamic cooperation of IoT cloud providers. Future Gene. Comp. Syst. **70**, 126–137 (2017)
66. Escribano, B.: Privacy and security in the Internet of Things: challenge or opportunity. In: OLSWANG, November 2014. http://www.olswang.com/me-dia/48315339/privacy_and_security_in_the_iot.pdf
67. Opinion 8/2014 on the on Recent Developments on the Internet of Things, October 2014. http://ec.europa.eu/justice/data-protection/article-29/documentation/opinion-recommendation/files/2014/wp223_en.pdf
68. Want, R., Dustdar, S.: Activating the Internet of Things. Computer **48**(9), 16–20 (2015)
69. Pflanzner, T., Kertesz, A., Spinnewyn, B., Latre, S.: MobIoTSim: towards a mobile IoT device simulator. In: 2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW), pp. 21–27 (2016)
70. IBM IoT Foundation message format. https://docs.internetofthings.ibmcloud.com/gateways/mqtt.html#/managed-gateways#managed-gateways. Accessed Mar 2017
71. OpenWeatherMap. http://www.openweathermap.org. Accessed Mar 2017