Cloud Service Brokerage and Service Arbitrage for Container-Based Cloud Services

Ruediger Schulze(✉)

IBM Germany Research and Development GmbH,
Schoenaicher Str. 220, 71034 Boeblingen, Germany
ruediger.schulze@de.ibm.com

Abstract. By exploiting the portability of application containers, platform- and software-as-a-service providers receive the flexibility to deploy and move their cloud services across infrastructure services delivered by different providers. The aim of this research is to apply the concepts of cloud service brokerage to container-based cloud services and to define a method for service arbitrage in an environment with multiple infrastructure-as-a-service (IaaS) providers. A new placement method based on constraint programming is introduced for the optimised deployment of containers across multiple IaaS providers. The benefits and limitations of the proposed method are discussed and the efficiency of the method is evaluated based on multiple deployment scenarios.

1 Introduction

Container-based applications are on the rise. Docker [1] as an open platform for application containers has become very popular since its first release in June 2014 and a complete ecosystem of supporting tools has grown up. With Docker, developers receive the option to package applications and their dependencies into self-contained images that can run as containers on any server. Multiple large Cloud Service Providers (CSP) have embraced Docker as container technology and announced alliances with the Docker company. Container-based applications are highly portable and independent of the hosting provider and hardware.

Cloud services such as Platform-as-a-services (PaaS) and Software-as-a-services (SaaS) have usually a topology that aligns with the components of a multi-tier architecture, including tiers for presentation, business logic and data access. The topology of the cloud service describes the structure of the IT service delivered by the CSP, the components of the service and the relationships between them. Cloud services designed for the use in enterprises have multiple Quality of Service (QoS) requirements such as High Availability (HA) and Disaster Recovery (DR) targets, performance and scalability requirements, and require compliance to security standards and data location regulations. The requirements for HA, DR and horizontal scaling are the drivers to design and deploy

M. Luo and L.-J. Zhang (Eds.): CLOUD 2018, LNCS 10967, pp. 97–111, 2018.
https://doi.org/10.1007/978-3-319-94295-7_7

the cloud services in multi-node topologies which may span multiple availability zones and geographical sites.

PaaS and SaaS providers in public or private cloud environments receive with Docker the option to deliver their cloud services using container-based applications or micro-services. The portability of the application containers enables them to easily move cloud services between IaaS providers and to locations where, for instance, the customer's data resides, the best SLA is achieved or where the hosting is most cost-effective. PaaS and SaaS providers will benefit from employing brokerage services that allow them to choose from a variety of IaaS providers and to optimise the deployment of their cloud services with respect to the QoS requirements, distribution and cost. By using a Cloud Service Broker (CSB), PaaS and SaaS providers will be enabled to structure their offering portfolio with additional flexibility and to quickly respond to new or changing customer requirements. New options arise from the easy deployment of the containers across the multiple IaaS providers. A PaaS or SaaS provider may deploy a container to a new geographic location by selecting a different IaaS provider when the primary one is not available there. A globally delivered cloud service may be provided by using resources from multiple IaaS providers. DR use cases may be realized by selecting a different provider for the backup site. Load-balancing and scalability built into the cloud service will allow to gradually migrating the service from one provider to another one with no downtime by simply moving the containers to the new provider.

Cloud service brokerage for application containers requires to define a new method for the optimised placement of the containers on IaaS resources of multiple providers. The method has to take the attributes of the containers and the IaaS resources into account, and honour the QoS requirements of enterprise-ready cloud services. The initial placement of the containers has to follow the specification of the topology of the cloud service. Aside of the attributes used for rating IaaS providers, the CSB has to consider container related attributes such as the built-in container support of the IaaS providers, the packaging of containers in virtual machines and the clustering of containers. The optimisation of the infrastructure resources of the container-based cloud services has to be without impact and visibility to the cloud service consumers. Access to user data and network connectivity to the cloud services have to be handled and delivered uninterrupted, and without the need to reconfigure clients.

A CSB for application containers may use a constraint programming-based engine to make the placement decision about the optimal IaaS provider. The engine will use as input the requirements of the container and information about each of the available IaaS providers. The objective of this research is to define a method for service arbitrage that allows for the optimised placement of containers in a cloud environment with multiple IaaS providers and to demonstrate the feasibility of the proposed method.

2 Background Research

2.1 Cloud Service Broker

The definition of a cloud broker as introduced by NIST is widely referenced by other authors, e.g., in [2–5]. In the NIST CCRA [6], a cloud broker is described as an "entity that manages the use, performance and delivery of cloud services and negotiates relationships between cloud providers and cloud consumers." The cloud broker is an organisation that serves as a third-party entity as a centralised coordinator of cloud services for other organisations and enables users to interact through a single interface with multiple service providers [2,7]. Gartner [8] describes cloud service brokerage as "an IT role and business model in which a company or other entity adds value to one or more (public or private) cloud services on behalf of one or more consumers of that service via three primary roles including aggregation, integration and customisation brokerage."

NIST [6] and Gartner [9] define three categories of services that can be provided by a CSB. *Service intermediation* enables a CSB to enhance a service by improving some specific capability and providing value-added services to cloud consumers. Service intermediation is responsible for service access and identity management, performance reporting, enhanced security, service pricing and billing. A CSB uses *service aggregation* to combine and integrate multiple services into one or more new services while ensuring interoperability. The CSB is responsible for the integration and consolidation of data across multiple service providers, and ensures the secure movement of the data between the cloud consumer and the cloud providers. The key aspect of the service aggregation is to ease service selection and present services from separate providers as a unique set of services to the cloud service consumer. *Service arbitrage* is the process of determining the best CSP. The CSB has the flexibility to choose a service from multiple providers and can, for example, use a credit-scoring service to measure and select a provider with the best score. Service arbitrage adds flexibility and choice to service aggregation as the aggregated services are not fixed.

Six key attributes of CSB are derived in [7] mainly based on the categories defined by NIST [6] and Gartner [8,9], and used for evaluation of existing CSBs: *intermediation*, *aggregation*, *arbitrage*, *customisation*, *integration* and *standardisation*. According to [10], integration is focused on creating an unified, common system of services by integrating private and public clouds or bridging between CSPs. Customisation refers to the aggregation and integration with other value-added services, including the creation of new original services [10]. Both integration and customisation are closely interlinked with service aggregation and intermediation, and in-fact it is difficult to find distinct definitions of these attributes in the literature. Standardisation among the CSB mechanisms and across the cloud services of different providers enables interoperability, and supports the process of service selection by a CSB.

A comprehensive model of a CSB architecture is described in [2]. The CSB environment is built of the same components as the management platform for a single cloud but additional complexity is introduced into the system by the

requirement to support multiple CSPs. Bond [2] distinguishes between the functions of the Cloud Broker Portal, the Cloud Broker Operations and Management and the multiple CSPs, and aligns them in a layered model of a vendor-agnostic CSB architecture. Governance is introduced in addition as a set of functions which are orthogonal to the layers of the model and have to be realised for all functional components of the architecture.

A taxonomy of brokering mechanisms is given in [12]. *Externally managed* brokers are provided off-premise by a centralized entity, for instance, by a third-party cloud provider or SaaS offering. *Externally managed* brokers are transparent for applications using the services provided by the broker. *Directly managed* brokers are incorporated into an application, have to keep track of the application's performance and be built to meet the availability and dependability requirements of the applications. *Externally managed* brokers can be classified into SLA-based and trigger-action brokers. In case of *SLA-based broker*, a cloud user specifies the brokering requirements of a SLA in form of constraints and objectives, and the CSB finds the most suitable cloud service by taking into account the user requirements specified by the SLA. For trigger-action broker, a cloud user specifies a set of triggers and associates actions with them. A trigger becomes active when a predefined condition is meet, e.g., the threshold of a specific metric is exceeded, and the associated action is executed. Actions can be, for instance, scale-out and provisioning activities, and may include bursting into a public cloud when there is a spike in the demand for computing capacity.

In the context of the interoperability of clouds, the following challenges are described in [13] and applied here to CSBs. In an environment with multiple cloud service providers, each provider is expected to have its own SLA management mechanism. A CSB has to establish federation of the SLAs from each CSP in order to set up and enforce a global SLA. Methods and protocols for the negotiation of dynamic and flexible SLAs between the CSB and the multiple CSPs are required. Another important issue is the enforcement of the SLAs in environments with conflicting policies and goals, e.g., a CSB may offer a service with a SLA for HA, while none of the providers are willing to offer such a service. In addition to the SLA, there can be a Federation-Level Agreement that defines rules and conditions between the CSB and the CSPs, e.g., about pools of resources and the QoS such as the minimum expected availability. The CSB has to establish functions for matching the guaranteed QoS of cloud services offered by the CSPs with the QoS requirements of the end-user, and for monitoring that the promised QoS and SLA is provided to the cloud service consumer. The dependencies of a CSP to other providers have to be considered by the CSB as well. The QoS of a higher-layered service can be affected in cases when the CSP of the service itself uses external services. If one of the providers of the lower-layered services is not functioning properly, the performance of the higher-layered service may be affected and impact finally the SLA agreed by the CSB. The CSB has to guarantee the security, confidentiality and privacy of the data processed by the services provided. Within the country where the services are delivered, the CSB must comply with the legislation and laws concerning

the privacy and security of the data. Therefore, the CSB has to implement geolocation and legislation awareness policies and enforce compliance with those policies. As part of the SLA management, services of specific providers can be avoided or agreement can be made that placing data outside of a given country is prohibited.

The results of a survey of CSB projects are described in [14]. The authors consider four categories of CSB technologies: *CSBs for performance* to address issues of cloud performance comparison and prediction, *CSBs for application migration* which provide decision support when moving applications to the cloud, *theoretical models for CSBs* which describe purely theoretical and mathematical techniques and *data for CSBs* that summarises providers of data and metrics available for use by CSBs. A comprehensive list of commercial CSB projects is given in [10]. Recent research about CSBs has a significant focus on service arbitration across numerous CSPs, in particular on optimising the allocation of resources from different IaaS providers. The use of arbitration engines enables CSBs to automatically determine the best CSP and service for any given customer order. The attributes considered in the optimisation process vary depending on the proposed method. Typically attributes for rating IaaS providers are: the supported operating systems and configurations, geographical location, costs and rates, bandwidth and performance, SLA terms, legalisation and security, compliance and audit [15,16].

The placement of VMs in cloud and virtual environments is a critical operation as it has an direct impact on the performance, resource utilisation, power-consumption and cost. The subject of VM placement is widely discussed in the research literature. Detailed reviews of the current VM placement algorithms can be found in [17,18]. According to [18], the mono-objective, multi-objective solved as mono-objective and pure multi-objective approaches can be distinguished with respective optimisation objectives. *Mono-objective methods* are designed for the optimisation of a single objective or the individual optimisation of more objective functions, but one at a time. For *multi-objective solved as mono-objective approaches*, multiple objective functions are combined into a single objective function. The weighted sum method is used most often by this approach. This method defines one objective function as the linear combination of multiple objectives. A disadvantage of this approach is that it requires knowledge about the correct combination of the objective functions – which is not always available. Pure multi-objective approaches have a vector of multiple objective functions that are optimised. Only a small number of methods are described in the literature which use a pure multi-objective approach for VM placement.

A broad range of different VM placement schemes (18 different in total) are analysed in [17]. Constraint programming based VM placement is described as one of the considered placement schemes. The following classification of the placement schemes is proposed: *Resource-aware VM placement schemes* consider the infrastructure resource requirements of the VMs are considered in the placement decisions by these schemes. Efficient resource-aware placement tries to optimally place VMs on the hosts, so that the overall resource utilisation

is maximised. Most of the schemes consider CPU and memory resources, some network resources, and a minor number includes the device or disk I/O, or try to minimise the number of active hosts. Designed for the use by the CSPs, *power-aware VM placement schemes* try to make cloud data centres more efficient and to reduce the power consumption in order to enable green cloud computing. The objective of these schemes is to reduce the number of active host, networking and other data center components. The methods include VM consolidation and packaging of VMs on the same host or in the same rack, and powering off not needed VMs, hosts and network components. The attributes considered by the power-aware schemes include the CPU utilisation (e.g., based on the states: idle, average, active and over utilised), the server power usage and the host status (running, ready, sleep, off), costs for network routing and data center power usage, and the distance between VMs. *Network-aware VM placement schemes* try to reduce the network traffic or try to distribute network traffic evenly in order avoid congestion. The placement schemes allocate the VMs with more or extensive communication on the same host, to the same switch and rack, or within the same data center in order to reduce the network traffic within the data center and across data centres. Most common is the consideration of the traffic between VMs by the network-aware VM placement schemes, some evaluate the traffic between the hosts and selected schemes try to minimise the transfer time between the VMs and data, and the distance between the VMs. *Cost-aware VM placement schemes* try to reduce the costs for the CSPs while considering the QoS of the cloud services and honouring the SLAs. The schemes use different types of costs as attributes, such as the VM, physical machine, cooling and data center costs, and the distance between the VMs and the clients.

Conceptually a similar approach is taken in [18] with the classification of the objective functions. Based on the study of 56 different objective functions, the classification into five groups of objective functions is described: *Energy Consumption Minimisation*, *Network Traffic Minimisation*, *Economical Costs Optimisation*, *Performance Maximisation* and *Resource Utilisation Maximisation*. Most of the publications are focused on single-cloud environments, i.e. for use by CSPs. Seven of the methods are suitable for multi-cloud environments, i.e. use multiple cloud computing data centres from one or more CSP. Only two articles take a broker-oriented approach.

Different methods are employed by CSBs for rating IaaS providers, e.g., genetics algorithms [19,20] and rough sets [16,21]. Multiple projects propose CSBs which take advantage of the different price options such as for on-demand, reservation and spot instances [22], examples can be found in [23–26]. There are a couple of academic and open-source implementations of CSBs, e.g., STRATOS [27], QBROKAGE [19] and CompatibleOne [28].

2.2 Constraint Programming

Constraint programming is a form of declarative programming which uses variables and their domains, constraints and objective functions in order to solve

a given problem. The purpose of constraint programming is to solve constraint satisfaction problems as defined in [30,31]:

Definition 1 (Constraint Satisfaction Problem). *A Constraint Satisfaction Problem $\mathcal{P}$ is a triple $\mathcal{P} = (X, D, C)$ where X is an n-tuple of variables $X = (x_1, x_2, \ldots, x_n)$, D is a corresponding n-tuple of domains $D = (D_1, D_2, \ldots, D_n)$ such that $x_i \in D_i$, C is a t-tuple of constraints $C = (C_1, C_2, \ldots, C_t)$.*

The domain D_i of a variable x_i is a finite set of numbers, and can be continuous or of a discrete set of values. In order to describe a constraint satisfaction problem $\mathcal{P}$, a finite sequence of variables with their respective domains is used together with a finite set of constraints. A constraint over a sequence of variables is a subset of the Cartesian product of the variables' domains in the scope of the constraint.

Definition 2 (Constraints). *$C = (C_1, C_2, \ldots, C_t)$ is the set of constraints. A constraint C_j is a pair $\left(R_{S_j}, S_j\right)$ where R_{S_j} is a relation on the variables in $S_j = scope(C_j)$, i.e. the relation R_{S_j} is a subset of the Cartesian product $D_1 \times \ldots \times D_m$ of the domains $D_1, D_2, \ldots, D_m$ for the variables in S_j.*

A solution of the Constraint Satisfaction Problem $\mathcal{P}$ is defined as follows:

Definition 3 (Solution of $\mathcal{P}$). *A solution to the Constraint Satisfaction Problem $\mathcal{P}$ is a n-tuple $A = (a_1, a_2, \ldots, a_n)$ where $a_i \in D_i$ and each C_j is satisfied in that R_{S_j} holds the projection of A onto the scope of S_j.*

The definition of multiple global constraints such as the `alldifferent` constraint is described in the literature. The constraint `alldifferent` requires that the variables $x_{1,2}, \ldots, x_n$ take all different values. An overview of the most popular global constraints is given in [32].

Several publications focus on the use of CP-based cloud selection and VM placement methods. A method for cloud service match making based on QoS demand is introduced in [33]. CP is a convenient method for optimising the placement of VMs, as placement constraints can be directly expressed by variables representing the assignment of the VMs to the hosts and the allocation of resources for the VMs placed on each host. Resource-aware VM placement schemes are presented in [34–36]. A combined CP and heuristic algorithm is utilised in [35]. Special focus is put on fault tolerance and HA in [36]. In [37], the CP-based, open source VM scheduler BtrPlace [38] is used to exhibit SLA violations for discrete placement constraints, as these do not consider interims states of a reconfiguration process. As consequence, BtrPlace is extended with a preliminary version of continuous constraints and it is proved that these remove the temporary violation and improve the reliability. Power-aware methods are discussed in [39,40].

3 Method for Service Arbitrage

Aims of this research is to define a method for optimising the deployment of container-based applications across different CSPs. The objective is to find for

a given Docker container c the optimal host h. A host h is a virtual or physical machine that meets the requirements of the container c and is delivered by an IaaS provider. The optimisation problem to be solved can be described as transformation of a container domain $\mathbb{C}$ into a host domain $\mathbb{H}$:

$$f : \mathbb{C} \rightarrow \mathbb{H},\ c \mapsto h,\ \textit{where}\ c \in \mathbb{C}\ \textit{and}\ h \in \mathbb{H}. \tag{1}$$

The requirements of a container c are expressed as vector $r_c = (r_{c,1}, r_{c,i} \ldots, r_{c,n})$. Each attribute $r_{c,i}$ is an element or subset of a domain R_i with a finite number of elements. Likewise, the attributes of a host h are described by the a vector $a_h = (a_{h,1}, a_{h,j}, \ldots, a_{h,m})$ for which each attribute $a_{h,j}$ belongs to a domain A_j and $A = (A_1, A_j, \ldots, A_m)$. In order to solve the optimisation problem, the requirements $r_{c,i}$ of the containers and the attributes $a_{h,j}$ of the hosts have to be considered. As method for finding the optimal host h for a given container c, a CP model is used. As per Definition 1, a constraint satisfaction problem $\mathcal{P}$ is defined as the triple $\mathcal{P} = (X, D, C)$. The objective of the CP model is to provide solutions for the container placement problem $\mathcal{P}_{placement}$. The variables X and the corresponding domains D of the problem $\mathcal{P}_{placement}$ are defined by the attribute domains A of the hosts $\mathbb{H}$, i.e. $D = A$ and $X = (a_1, a_j, \ldots, a_m)$ where $a_j \in A_j$. Provided that the function *index* returns the index set I of any given set S, so that

$$\textit{index} : S \rightarrow I,\ s_i \mapsto i = \textit{index}(s), \tag{2}$$

then can be defined for the variables and domains of the host attributes the following:

$$\begin{aligned}
\textit{provider}: \quad & a_1 \in A_1 \text{ and } A_1 = \textit{index}(\mathbb{P}) \text{ where} && (3)\\
& \mathbb{P} = \{\text{AWS}, \text{DigitalOcean}, \text{Azure}, \text{SoftLayer}, \text{Packet}\} \\
\textit{host type}: \quad & a_2 \in A_2 \text{ and } A_2 = \textit{index}(\mathbb{T}) \text{ where} && (4)\\
& \mathbb{T} = \{\text{physical}, \text{virtual}\} \\
\textit{region}: \quad & a_3 \in A_3 \text{ and } A_3 = \textit{index}(\mathbb{R}) \text{ where} && (5)\\
& \mathbb{R} = \{\text{Australia}, \text{Brazil}, \text{Canada}, \text{France}, \text{Germany}\} \\
& \mathbb{R} = \mathbb{R} \cup \{\text{Great Britain}, \text{Hongkong}, \text{India}, \text{Ireland}\} \\
& \mathbb{R} = \mathbb{R} \cup \{\text{Italy}, \text{Japan}, \text{Mexico}, \text{Netherlands}, \} \\
& \mathbb{R} = \mathbb{R} \cup \{\text{Singapore}, \text{California}, \text{Iowa}, \text{New Jersey}\} \\
& \mathbb{R} = \mathbb{R} \cup \{\text{New York}, \text{Oregon}, \text{Texas}, \text{Washington}\} \\
& \mathbb{R} = \mathbb{R} \cup \{\text{Virginia}\} && (6)
\end{aligned}$$

$$\textit{data centres}: \quad a_4 \in A_4 \text{ and } A_4 = \{1, .., 52\} \tag{7}$$
$$\textit{availability zone}: \quad a_5 \in A_5 \text{ and } A_5 = \{0, .., 5\} \tag{8}$$
$$\textit{cpu}: \quad a_6 \in A_6 \text{ and } A_6 = \{1, .., 40\} \tag{9}$$
$$\textit{memory (GB)}: \quad a_7 \in A_7 \text{ and } A_7 = \{1, .., 488\} \tag{10}$$
$$\textit{disk size (GB)}: \quad a_8 \in A_8 \text{ and } A_8 = \{1, .., 48000\} \tag{11}$$
$$\textit{disk type}: \quad a_9 \in A_9 \text{ and } A_9 = \textit{index}(\mathbb{D}) \text{ where} \tag{12}$$
$$\mathbb{D} = \{\text{HDD}, \text{SSD}\}$$
$$\textit{private}: \quad a_{10} \in A_{10} \text{ and } A_{10} = \{0, 1\} \tag{13}$$
$$\textit{optimised}: \quad a_{11} \in A_{11} \text{ and } A_{11} = \textit{index}(\mathbb{O}) \text{ where} \tag{14}$$
$$\mathbb{O} = \{\text{compute}, \text{memory}, \text{gpu}, \text{storage}\}$$
$$\mathbb{O} = \mathbb{O} \cup \{\text{network}, \text{none}\}$$
$$\textit{cost}: \quad a_{12} \in A_{12} \text{ and } A_{12} = \{1, .., 99999\} \tag{15}$$

In order to describe the constraints C, the requirements $r_c = (r_{c,1}, r_{c,i} \ldots, r_{c,n})$ of a container c have to be detailed first. The properties *ha_scale* and *dr_scale* are introduced to address the requirements for HA and DR. The requirement *ha_scale* describes how many containers have to be deployed across the data centres or available zones within one region and *dr_scale* is the number of containers that have to be deployed across multiple regions of the same or multiple providers in order to achieve protection against a disaster. The *op_factor* $r_{c,9}$ allows to request a larger host which can run $r_{c,9}$ instances of the container c. The *price limit* ($ 0.0001 per hour) specifies the maximum cost of host per hour which must not be exceeded. The attribute *private* $r_{c,11}$ allows to request to place the container c on a dedicated host.

$$\textit{host type}: \quad r_{c,1} \in R_1 \text{ and } R_1 = A_2 \tag{16}$$
$$\textit{region}: \quad r_{c,2} \in R_2 \text{ and } R_2 = A_3 \tag{17}$$
$$\textit{cpu}: \quad r_{c,3} \in R_3 \text{ and } R_3 = A_6 \tag{18}$$
$$\textit{memory (GB)}: \quad r_{c,4} \in R_4 \text{ and } R_4 = A_7 \tag{19}$$
$$\textit{disk size (GB)}: \quad r_{c,5} \in R_5 \text{ and } R_5 = A_8 \tag{20}$$
$$\textit{disk type}: \quad r_{c,6} \in R_6 \text{ and } R_6 = A_9 \tag{21}$$
$$\textit{ha\_scale}: \quad r_{c,7} \in R_7 \text{ and } R_7 = \{1, .., 5\} \tag{22}$$
$$\textit{dr\_scale}: \quad r_{c,8} \in R_8 \text{ and } R_8 = \{1, .., 5\} \tag{23}$$
$$\textit{op\_factor}: \quad r_{c,9} \in R_9 \text{ and } R_9 = \{1, .., 10\} \tag{24}$$
$$\textit{price limit}: \quad r_{c,10} \in R_{10} \text{ and } R_{10} = A_{12} \tag{25}$$
$$\textit{private}: \quad r_{c,11} \in R_{11} \text{ and } R_{11} = A_{10} \tag{26}$$
$$\textit{optimized}: \quad r_{c,12} \in R_{12} \text{ and } R_{12} = A_{11} \tag{27}$$
$$\textit{image}: \quad r_{c,13} \in R_{13} \text{ and } R_{13} = \mathbb{I} \text{ where} \tag{28}$$
$$\mathbb{I} \text{ is the set of Docker images}$$

In order to allow the placement of a container c either on an existing host or a new host, the domain $\mathbb{H}$ to be searched is defined as the union of the host templates T, i.e. the set of hosts which can be provisioned, and the already provisioned hosts H:

$$\mathbb{H} = T \cup H \tag{29}$$

wherein T is the superset of the templates t^p from all providers $\mathbb{P}$:

$$T = \bigcup_{p \in \mathbb{P}} T^p \text{ and } t^p \in T^p \tag{30}$$

The template t^p is described by the attribute vector a_t^p and domains A^p:

$$a_t^p = (a_{t,1}^p, a_{t,j}^p, \ldots, a_{t,m}^p) \text{ and } a_{t,j}^p \in A_j^p \tag{31}$$

$$A^p = A_1^p, A_j^p, \ldots, A_m^p \text{ and } p \in \mathbb{P} \tag{32}$$

The set of already provisioned hosts H is the union of the deployed hosts h^p at all providers $\mathbb{P}$:

$$H = \bigcup_{p \in \mathbb{P}} H^p \text{ and } h^p \in H^p \tag{33}$$

Provided that the host h^p is provisioned using the template h^p and that $\mathbb{C}_h$ denotes the set of containers deployed on the host h, the available resources on the host h^p can the be determined as follows:

$$cpu: \quad a_{h,6} = a_{t,6}^p - \sum_{c \in \mathbb{C}_h} r_{c,3} \tag{34}$$

$$memory(GB): \quad a_{h,7} = a_{t,7}^p - \sum_{c \in \mathbb{C}_h} r_{c,4} \tag{35}$$

$$disksize(GB): \quad a_{h,8} = a_{t,8}^p - \sum_{c \in \mathbb{C}_h} r_{c,5} \tag{36}$$

$$cost: \quad a_{h,12} = \frac{a_{t,12}^p}{|\mathbb{C}_h|} \tag{37}$$

The variables X represent the attributes of a single host. In order to respond to the requirements for HA and DR, multiple containers have to be placed on k anti-collocated hosts. $X_{e,f}$ denotes the variables and $a_j^{e,f}$ the attributes required to describe the k hosts $h_{e,f}$:

$$k = dr\_scale \cdot ha\_scale \tag{38}$$

$$X_{e,f} = (a_1^{e,f}, a_j^{e,f}, \ldots, a_m^{e,f}) \text{ where } a_j^{e,f} \in A_j \tag{39}$$

$$1 \leq e \leq dr\_scale \tag{40}$$

$$1 \leq f \leq ha\_scale \tag{41}$$

The constraints C can then be defined as follows:

$$hosts: \quad C_1 : \vee \Big(\wedge \big(a_j^{e,f} = a_{h,j}\big)\Big),\ \forall h \in \mathbb{H} \text{ and } 1 \leq j \leq 12 \tag{42}$$

$$host\ type: \quad C_2 : \begin{cases} a_2^{e,f} = r_{c,1}, & \text{if } r_{c,1} \geq 0 \\ true, & \text{otherwise} \end{cases} \tag{43}$$

$$region: \quad C_3 : \begin{cases} a_3^{e,f} = r_{c,2}, & \text{if } r_{c,2} \geq 0 \\ true, & \text{otherwise} \end{cases} \tag{44}$$

$$cpu: \quad C_4 : a_6^{e,f} \geq \big(r_{c,3} \cdot r_{c,9}\big) \tag{45}$$

$$memory: \quad C_5 : a_7^{e,f} \geq \big(r_{c,4} \cdot r_{c,9}\big) \tag{46}$$

$$disk\ size: \quad C_6 : a_8^{e,f} \geq \big(r_{c,5} \cdot r_{c,9}\big) \tag{47}$$

$$disk\ type: \quad C_7 : \begin{cases} a_9^{e,f} = r_{c,6}, & \text{if } r_{c,6} \geq 0 \\ true, & \text{otherwise} \end{cases} \tag{48}$$

$$price\ limit: \quad C_8 : \begin{cases} a_{12}^{e,f} \leq r_{c,10}, & \text{if } r_{c,10} \geq 0 \\ true, & \text{otherwise} \end{cases} \tag{49}$$

$$private: \quad C_9 : \begin{cases} a_{10}^{e,f} = r_{c,11}, & \text{if } r_{c,11} \geq 0 \\ true, & \text{otherwise} \end{cases} \tag{50}$$

$$optimised: \quad C_{10} : \begin{cases} a_{11}^{e,f} = r_{c,12}, & \text{if } r_{c,12} \geq 0 \\ true, & \text{otherwise} \end{cases} \tag{51}$$

$$HA: \quad C_{11} : \Big(\big(a_5^{e,f} \neq a_5^{e,g}\big) \wedge \big(a_4^{e,f} = a_4^{e,g}\big) \wedge \big(az_{enabled} = 1\big)\Big) \vee \Big(\big(a_4^{e,f} \neq a_4^{e,g}\big) \wedge \big(a_3^{e,f} = a_3^{e,g}\big)\Big) \tag{52}$$

where $1 \leq g \leq ha\_scale$ and $f \neq g$

$$DR: \quad C_{12} : \big(a_3^{d,f} \neq a_3^{e,g}\big) \tag{53}$$

where $1 \leq d \leq dr\_scale$ and $d \neq e$

The property $az_{enabled}$ indicates that the used technology generally supports the deployment of hosts into availability zones. The objective function f_{cost} for minimising the cost across the k hosts is defined as follows:

$$f_{cost} = minimise \sum a_{12}^{d,f}. \tag{54}$$

4 Results

In order to verify the effectiveness of the proposed method for service arbitrage, a series of deployment scenarios was executed with the objective to verify that the most cost-effective host for a given configuration is chosen for deployment. The verification was performed in an environment with five different IaaS providers.

In total, 3587 host templates with different server configurations in 22 regions and 52 data centres were given to the CP model as input. Actual prices from the CSPs were used for each of the configurations. The CP model was implemented using NumberJack [41] and the CP solver "Mistral" [42]. The execution of the test scenarios showed that the used CP solver is able to find optimal solutions for the CP model. The CP solver returns solutions in a reasonable time when only a single container has to be placed. The runtime of the CP solver increases significantly when multiple containers have to placed across different locations for HA and DR. In this case, the number of variables and constraints supplied to the CP model increase and the objective function becomes more complex. In order to validate if better performance results can be achieved with another CP solver supported by NumberJack, the "MiniSat" solver [43] was used. The test execution with "MiniSat" showed an increased CPU utilization on the hosting server and significant longer runtime. All further tests were executed using the "Mistral" solver afterwards. By applying a lower price limit to a deployment scenario with multiple containers, it was possible to obtain an optimal solution quicker. With the price limit applied, the initial number of host templates can be already reduced before the actual constraints are added to the CP model. The deployment scenarios with multiple containers shows as well that regions on different continents may be selected by the CP solver as optimal solution, e.g. Europe and North America. This solution may not be suitable for business use in all cases, e.g., when legal restrictions apply. Additional locality constraints or an objective function for minimising the distance between regions may be added to the CP model in future.

5 Summary and Conclusions

An important aspects of the proposed CP model is that the emphasis is not on rating the CSPs but the particular hosts for their capability to run a specific container, so that the CSP becomes only an attribute of the host. The proposed CP model was validated based on test data with host templates from five IaaS providers, 22 regions and 52 data centres. In this experimental research it is shown that the proposed CP model is capable to find the optimal placement for containers also in complex environments, and that HA and DR topologies of applications can be realised. It is further shown that the CP solver "Mistral" [42] used by NumberJack runs stable for large CP models. The duration of the process for finding the optimal solution increases significantly when multiple containers have to placed across different locations for HA and DR. Hence, the practical use of the proposed CP model in a production environment is not possible. Further research is required to reduce the complexity inherited from the input attributes before the actual CP model is constructed. Other CP solvers may be evaluated and the integration of rule-based algorithms such as Rete [44] into the CSB framework can be investigated. The objective of the integration with a rule-based approach will be to limit the number of CP variables and constraints to only the ones which are valid for a given request, so that the CP

solver can find a solution to the objective function within a few seconds. The advantage of the combination of the two approaches will be that the rule-based algorithm can provide a reduction of the solution space, while the CP solver is still used to find the best solution for the given objective function.

Aside of the runtime aspect, the CP model can be extended in future in various ways. The CP model uses a mono-objective function to minimise the cost. A multi-objective approach may take additional performance attributes into consideration and allow to maximise the service performance while providing the most cost-effective price. Such performance attributes could be gathered from monitoring data of the containers during runtime or be collected from IaaS benchmarking services like CloudHarmony [29]. The data model related the CP model is centred around host templates which represent virtual or physical servers. CSPs offer compute, storage and network resources today as independent services with various, flexible options for selection. The CP model may be extended to allow for better consumption and distinction of those services in the placement decisions. In addition, the CP model may be further extended to honour region specific prices, and optionally to take price differences for reservation, on-demand and spot instances into account. Further extension of the CP model can be done to support node clusters with multiple nodes and services with multiple containers.

With the proposed CP model, a first brokerage solution using service arbitrage for containers is provided. The underlying concepts were successful verified and allow for future research and development in this area.

References

1. What is Docker? https://www.docker.com/what-docker
2. Bond, J.: The Enterprise Cloud: Best Practices for Transforming Legacy IT. O'Reilly Media Inc., Newton (2015)
3. Amato, A., Di Martino, B., Venticinque, S.: Cloud brokering as a service. In: 2013 Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), pp. 9–16. IEEE (2013)
4. Park, J., An, Y., Yeom, K.: Virtual cloud bank: an architectural approach for intermediating cloud services. In: 2015 16th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), pp. 1–6. IEEE (2015)
5. Khanna, P., Jain, S., Babu, B.: BroCUR: distributed cloud broker in a cloud federation: brokerage peculiarities in a hybrid cloud. In: 2015 International Conference on Computing, Communication and Automation (ICCCA), pp. 729–734. IEEE (2015)
6. Liu, F., Tong, J., Mao, J., Bohn, R., Messina, J., Badger, L., Leaf, D.: NIST cloud computing reference architecture. NIST Spec. Publ. **500**(2011), 292 (2011)
7. Mostajeran, E., Ismail, B.I., Khalid, M.F., Ong, H.: A survey on sla-based brokering for inter-cloud computing. In: 2015 Second International Conference on Computing Technology and Information Management (ICCTIM), pp. 25–31. IEEE (2015)
8. Gartner: Cloud service brokerage. Gartner Research (2016). http://www.gartner.com/it-glossary/cloud-services-brokerage-csb

9. Gartner: Gartner says cloud consumers need brokerages to unlock the potential of cloud services. Gartner Research, July 2009. http://www.gartner.com/newsroom/id/1064712
10. Guzek, M., Gniewek, A., Bouvry, P., Musial, J., Blazewicz, J.: Cloud brokering: current practices and upcoming challenges. IEEE Cloud Comput. **2**, 40–47 (2015)
11. What is ITIL best practice? https://www.axelos.com/best-practice-solutions/itil/what-is-itil
12. Grozev, N., Buyya, R.: Inter-cloud architectures and application brokering: taxonomy and survey. Softw. Pract. Exp. **44**(3), 369–390 (2014)
13. Toosi, A.N., Calheiros, R.N., Buyya, R.: Interconnected cloud computing environments: challenges, taxonomy, and survey. ACM Comput. Surv. (CSUR) **47**(1), 7 (2014)
14. Barker, A., Varghese, B., Thai, L.: Cloud services brokerage: a survey and research roadmap. In: 2015 IEEE 8th International Conference on Cloud Computing (CLOUD), pp. 1029–1032. IEEE (2015)
15. Ngan, L.D., Tsai Flora, S., Keong, C.C., Kanagasabai, R.: Towards a common benchmark framework for cloud brokers. In: 2012 IEEE 18th International Conference on Parallel and Distributed Systems (ICPADS), pp. 750–754. IEEE (2012)
16. Tiwari, A., Nagaraju, A., Mahrishi, M.: An optimized scheduling algorithm for cloud broker using adaptive cost model. In: 2013 IEEE 3rd International Advance Computing Conference (IACC), pp. 28–33. IEEE (2013)
17. Masdari, M., Nabavi, S.S., Ahmadi, V.: An overview of virtual machine placement schemes in cloud computing. J. Netw. Comput. Appl. **66**, 106–127 (2016)
18. Lopez-Pires, F., Baran, B.: Virtual machine placement literature review. arXiv preprint arXiv:1506.01509 (2015)
19. Anastasi, G.F., Carlini, E., Coppola, M., Dazzi, P.: QBROKAGE: a genetic approach for QoS cloud brokering. In: 2014 IEEE 7th International Conference on Cloud Computing (CLOUD), pp. 304–311. IEEE (2014)
20. Kessaci, Y., Melab, N., Talbi, E.-G.: A Pareto-based genetic algorithm for optimized assignment of VM requests on a cloud brokering environment. In: 2013 IEEE Congress on Evolutionary Computation (CEC), pp. 2496–2503. IEEE (2013)
21. Srivastava, N.K., Singh, P., Singh, S.: Optimal adaptive CSP scheduling on basis of priority of specific service using cloud broker. In: 2014 9th International Conference on Industrial and Information Systems (ICIIS), pp. 1–6. IEEE (2014)
22. Amazon EC2 pricing. Amazon Web Services. https://aws.amazon.com/ec2/pricing/
23. Wang, W., Niu, D., Liang, B., Li, B.: Dynamic cloud instance acquisition via iaas cloud brokerage. IEEE Trans. Parallel Distrib. Syst. **26**(6), 1580–1593 (2015)
24. Liu, K., Peng, J., Liu, W., Yao, P., Huang, Z.: Dynamic resource reservation via broker federation in cloud service: a fine-grained heuristic-based approach. In: 2014 IEEE Global Communications Conference (GLOBECOM), pp. 2338–2343. IEEE (2014)
25. Nesmachnow, S., Iturriaga, S., Dorronsoro, B.: Efficient heuristics for profit optimization of virtual cloud brokers. IEEE Comput. Intell. Mag. **10**(1), 33–43 (2015)
26. Vieira, C., Bittencourt, L., Madeira, E.: A scheduling strategy based on redundancy of service requests on iaas providers. In: 2015 23rd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), pp. 497–504. IEEE (2015)
27. Pawluk, P., Simmons, B., Smit, M., Litoiu, M., Mankovski, S.: Introducing STRATOS: a cloud broker service. In: 2012 IEEE Fifth International Conference on Cloud Computing, pp. 891–898. IEEE (2012)

28. Yangui, S., Marshall, I.-J., Laisne, J.-P., Tata, S.: CompatibleOne: the open source cloud broker. J. Grid Comput. **12**(1), 93–109 (2014)
29. Cloud Harmony - Transparency for the Cloud. Gartner, Inc. https://cloudharmony.com
30. Freuder, E.C., Mackworth, A.K.: Constraint satisfaction: an emerging paradigm. Handb. Constr. Program. **2**, 13–27 (2006)
31. Apt, K.: Principles of Constraint Programming. Cambridge University Press, Cambridge (2003)
32. Bockmayr, A., Hooker, J.N.: Constraint programming. Handb. Oper. Res. Manag. Sci. **12**, 559–600 (2005)
33. Zilci, B., Slawik, M., Kupper, A.: Cloud service matchmaking using constraint programming. In: 2015 IEEE 24th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), pp. 63–68. IEEE (2015)
34. Zhang, L., Zhuang, Y., Zhu, W.: Constraint programming based virtual cloud resources allocation model. Int. J. Hybrid Inf. Technol. **6**(6), 333–334 (2013)
35. Zhang, Y., Fu, X., Ramakrishnan, K.: Fine-grained multi-resource scheduling in cloud datacenters. In: 2014 IEEE 20th International Workshop on Local & Metropolitan Area Networks (LANMAN), pp. 1–6. IEEE (2014)
36. Bin, E., Biran, O., Boni, O., Hadad, E., Kolodner, E.K., Moatti, Y., Lorenz, D.H.: Guaranteeing high availability goals for virtual machine placement. In: 2011 31st International Conference on Distributed Computing Systems (ICDCS), pp. 700–709. IEEE (2011)
37. Dang, H.T., Hermenier, F.: Higher SLA satisfaction in datacenters with continuous VM placement constraints. In: Proceedings of the 9th Workshop on Hot Topics in Dependable Systems, p. 1. ACM (2013)
38. BtrPlace - An Open-Source Flexible Virtual Machine Scheduler. http://www.btrplace.org
39. Van, H.N., Tran, F.D., Menaud, J.-M.: Performance and power management for cloud infrastructures. In: 2010 IEEE 3rd International Conference on Cloud Computing (CLOUD), pp. 329–336. IEEE (2010)
40. Dupont, C., Giuliani, G., Hermenier, F., Schulze, T., Somov, A.: An energy aware framework for virtual machine placement in cloud federated data centres. In: 2012 Third International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet (e-Energy), pp. 1–10. IEEE (2012)
41. NumberJack - A Python Contraint Programming platform. http://numberjack.ucc.ie
42. Mistral. http://homepages.laas.fr/ehebrard/mistral.html
43. The MiniSat Page. http://minisat.se
44. Forgy, C.L.: Rete: a fast algorithm for the many pattern/many object pattern match problem. Artif. Intell. **19**(1), 17–37 (1982)