Rule-Based Resource Matchmaking for Composite Application Deployments across IoT-Fog-Cloud Continuums

Josef Spillner, Panagiotis Gkikopoulos
Zurich University of Applied Sciences
Winterthur, Switzerland
Email: {josef.spillner,pang}@zhaw.ch

Alina Buzachis, Massimo Villari
MIFT Department, University of Messina
Messina, Italy
Email: {abuzachis,mvillari}@unime.it

***Abstract*—Where shall my new shiny application run? Hundreds of such questions are asked by software engineers who have many cloud services at their disposition, but increasingly also many other hosting options around managed edge devices and fog spectrums, including for functions and container hosting (FaaS/CaaS). Especially for composite applications prevalent in this field, the combinatorial deployment space is exploding. We claim that a systematic and automated approach is unavoidable in order to scale functional decomposition applications further so that each hosting facility is fully exploited. To support engineers while they transition from cloud-native to continuum-native, we provide a rule-based matchmaker called RBMM that combines several decision factors typically present in software description formats and applies rules to them. Using the MaestroNG orchestrator and OsmoticToolkit, we also contribute an integration of the matchmaker into an actual deployment environment.**

***Index Terms*—Cloud Computing, Osmotic Computing, Fog Computing, Deployment, Matchmaking**

1. Introduction

Continuum applications are widely considered as the next evolutionary step of cloud applications [1], by departing from the notion of single or even multiple clouds and instead incorporating other compute facilities such as data-generating nodes (mobile devices, IoT sensor nodes) and intermediaries (edges, fogs) [2].

In such continuums, various deployment patterns can be identified. Fig. 1 shows four such patterns, out of a lot more that are already used in industrial applications: Pipelining follows the flow of data mostly for analytics purposes; multiplexing considers the parallel use of multi-cloud services for increasing availability or security; offloading adaptively switches on a cloud on demand; and switching optimises the connection latency by choosing between intermediary and direct cloud access depending on the situation of each link.

Instead of hardcoding where each part of a composite application runs, it is possible and indeed desired to automate that process and specify requirements along with desired patterns programmatically. This requires transferring

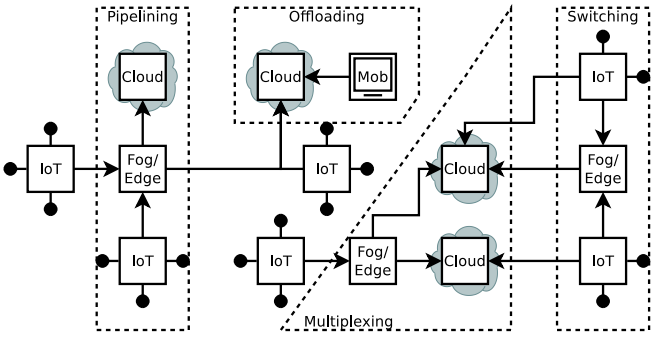

Figure 1. Application deployments across continuums

the knowledge from humans, who would do the hardcoding, to knowledge bases and tools that understand the characteristics of application parts, resources along the continuum, and other *decision factors*. Based on this knowledge, matchmaking can be performed to solve the assignment problem and yield suitable deployment instructions that fulfil all hard constraints and moreover consider soft preferences.

In this paper, we define application and resource models, then categorise the associated decision factors in the RBMM model. To increase automation, we present suitable acquisition techniques for each, complemented by propagation, skipping, deployment and accumulation rules. Subsequently, we present simple matchmaking algorithms to yield suitable deployment topologies (Sects. 2–3). To increase the applicability of the work, we implement the matchmaking along with a composite application deployment scenario (Sect. 4), where the OsmoticToolkit is taken into consideration as a practical demonstrator, before diving into related works and concluding our work (Sects. 5–6).

2. Models, Decision Factors and Rules

2.1. Definitions and Models

We define a composite application A to consist of a number of software artefacts a_i which are loosely coupled and instantiated as application execution units, or parts, with certain scaling factors, i.e. $A = \{s_0 \times a_0, s_1 \times a_1, \ldots\}$.

A continuum resource collection R consists of independent resources r_i whose owners or operators can differ, leading to further differences in location as well as technical characteristics including the resource level (infrastructure, platform, middleware).

Both artefacts and resources have certain properties, although not all of them are guaranteed to be explicitly expressed in machine-readable descriptions. Sometimes, they are also loosely expressed, for example a runtime environment $java$ without corresponding version number. Therefore, we assume those *deployment factors* to contain a measure of uncertainty, i.e. $F = \{u_0 \times f_0, u_1 \times f_1, \ldots\}$. In component notation, resource factors are *offered* whereas artefact factors are *required*. Some factors only exist for either artefacts or resources, while others exist for both; this is expressed by the *factor scope* (A, R or both).

A deployment plan (assignment) is the projection $A \mapsto_{C+P} R_{used} \subseteq R$ under a set of conditions C and a set of preferences P. Conditions must be fulfilled (e.g. sufficient memory to run the application, monthly cost not more than a certain limit) whereas preferences are used to determine the winning resource combination out of several that fulfil the conditions (e.g. smallest possible latency). Multiple preferences can be combined by weights, i.e. $P = \{w_0 \times p_0, w_1 \times p_1, \ldots\}$. All conditions and preferences are expressed with application-specific rules (Ψ) referencing arbitrary deployment factors F and applying to any pair (a_i, r_i).

Our model is limited by not taking data dependencies or workflows into account. Specifically, we assume for simplicity that any application part can generate data and transmit it freely to any other part. We acknowledge this limitation while claiming that even the simplified model advances systematic deployment methodologies beyond current deployment tool designs for clouds and continuums. On the other hand, the model is flexible by allowing to skip certain factors so that a subsequent matchmaker or deployment tool can perform further micro-optimisation. These skipping rules, together with the deployment rules as well as propagation and accumulation rules, lead to a highly flexible approach that fits multiple deployment patterns and topologies.

2.2. Decision Factors

Deployment processes are constrained and influenced by decision factors that are inherent to any application part or resource, or even both, as well as to the composite application as a whole. The precise definition of these factors depends on the operational environment and primarily on the ability to automate the collection of factors. We propose the following common categories of hosting-related factors to control application deployments in continuums:

1) Infrastructure. Application parts are only deployed if their requirements in terms of low-level computing hardware if fulfilled.
2) Platform. For higher-level deployments, operating systems as well as language runtimes and their version along with managed backend services are important.
3) Connectivity. For latency-sensitive or bandwidth-intensive services, the connectivity to end users or other resources in the continuum is significant.
4) Security. Vulnerabilities and protection levels further influence deployments.
5) Economics. Pricing models, plans and tariffs must match before a deployment is considered valid.

Furthermore, we foresee support external context information, such as date, time and system load, but omit their handling from our current approach. Table 1 contains an exemplary set of suitable decision factors. All values are restricted by a (typed) value space and may be complemented with physical units for numerical values. The selection of factors may be extended and customised depending on the application domain or specific infrastructure topologies.

TABLE 1. SUBSET OF DEPLOYMENT DECISION FACTORS

Scope	Name	Values (examples)
A, R	memory	128 MiB, 2 GiB
A, R	runtime	python:3, java
A, R	latency	5 ms
A, R	duration	900 s
A, R	zone	intranet, dmz, internet
A	vulnerability	backdoor, CVE-477
A	consistency	true, false
A	complexity	high, medium, low
A	port	9233
R	country	gb, cn
R	trust	high, low
R	billing	monthly, pay-per-use, free
R	gpu	true, false

2.3. Acquisition Techniques

Acquisition of accurate and up-to-date values of decision factors is defined as the automated process of building up the knowledge on both software applications, ranging from the source code level to packaged artefacts ready for deployment, and computing resources.

2.3.1. Software artefacts. To acquire metrics from composite application units, we rely on the Microservice Artefact Observatory (MAO) [3] that is able to perform static and dynamic assessment on a number of artefacts, including Docker containers. MAO is a federated data management system that schedules and orchestrates individual tools that acquire metrics for different software artefacts. Multiple tools can acquire different metrics for the same type of artefact, creating a more detailed overview of the artefact's quality and properties. The result is an aggregated knowledge base of software artefact information. Data can be contributed or validated by any member node of the federation.

2.3.2. Computing resources. Researchers have proposed automated approaches to scrape key characteristics of cloud

services from provider websites as a means to automate the acquisition process [4]. Similarly, CloudPick acknowledges the variety in cross-cloud service selection and contributes a translator component for automatic semantic enrichment [5]. We point out that certain subjective factors, such as trust in a resource provider, needs to be modelled manually.

2.4. Rules

Rules (Ψ) applying to factors are composed of propagation rules (Ψ_π), skipping rules (Ψ_σ) deployment rules (Ψ_δ) and accumulation rules (Ψ_α). They are applied in this order: First, propagation rules use invariants to complement missing factors or change existing factors in application compositions and resource sets. On this basis, skipping rules temporarily hide factors – marking them to be skipped during processing – so that they remain intact in the output mapping and serve as input for further post-processing. Then, matchmaking is performed with deployment rules, and all successful assignments imply the use of accumulation rules to adjust post-deployment resource characteristics. In case an assignment is reverted, for instance through backtracking, the accumulation rules are executed in inverse order to roll back forecasted resource modifications.

2.4.1. Propagation rules. By considering a hierarchical application and resource model, it is possible to specify which factors at lower levels invariably influence those at higher levels, and vice-versa, as well as lower level siblings (up-, down- and side-propagation). The hierarchy can have multiple levels, although on the application side many composition formats adopted in industry only support two levels. Resources can however have sub-resources, for instance a VM instance offering both CPU and GPU computing access. Through propagation, further efficiency gains can be achieved when only modelling some of the factors that have to be modelled manually or whose automated acquisition consumes a lot of time. For instance, a software composition affected by a security vulnerability in one constituent unit is considered itself tainted, representing an up-propagation, whereas another component in the same composition remains unaffected by itself. Specifically, the following propagation rules, including two trivial ones, are useful in continuums and need to be expressible as preprocessing step in matchmaking.

1) Replication. All factors in a trivially apply to all other instances of the same artefact. This applies to all static factors as well as, assuming they share the same resource, to dynamic runtime-related ones such as maximum task processing rate. However, in our work we apply these rules before deployment and thus do not consider dynamic factors.
2) Subsumption. The resource needs of A are trivially defined as the conjunction of all resource needs of the constituent a including scaling factors (up).
3) Bounding. The upper bound of latency in a is mirrored in A (up).
4) Tainting. Any quality deficiency or security vulnerability in a is mirrored in A (up), and any trust level in r is mirrored in the subset of R that shares the same operator (side).

2.4.2. Skipping rules. A potential use case of matchmaking is generating a subset of valid deployment mappings, and further post-processing it with another matchmaking or optimisation tool that might use different algorithms for achieving increased output precision. Skipping rules mark attributes such as CPU or memory needs by the application, and the corresponding offered capacity by the resources, effectively leading to them being skipped during the matchmaking. Afterwards, they get reinstated on the resulting mapping. Possible skipping rules are:

1) Context. Skip CPU and memory factors, deferring these technical details to later, and instead perform matchmaking primarily on context factors.
2) Feasibility. Pre-check whether a deployment is technically feasible at all by skipping non-technical factors such as trust, country or geolocation.

2.4.3. Deployment rules. These rules set constraints on where each application part a can be deployed. We require the deployment rules to express the following scenarios:

1) An application processing sensitive personal information shall not be deployed in hosting locations whose jurisdiction does not support certain minimum guarantees on privacy.
2) An application subject to a vulnerability shall only be deployed into the demilitarised zone (DMZ), not in the internal network behind the firewall.
3) Any application part a needs to be deployed into a resource with sufficient memory. For latency-sensitive applications, the entire application A needs to fit within one resource.

2.4.4. Accumulation rules. Due to resource sharing and resource utilisation in general, each deployment leads to some changes in factors. We differentiate between constant factors unimpeded by any deployment (e.g. location of a datacentre) and those changing their values according to accumulation rules, for instance, available memory being reduced by any running application. More accurately, we assume that resource access is either unlimited, shared, or exclusive.

A concrete set of rules might look as follows:

1) The amount of free memory in r is reduced by the memory claimed by a (shared).
2) The range of free port numbers in r is reduced by any allocated port in a (shared).
3) A GPU available as sub-resource in r is occupied by any a claiming to perform GPU computing (exclusive).

The distinction into resource access models influences the permissible algorithms. Under the assumption that resources are infinite ($|R| = \infty$) or largely available beyond

what can possibly be consumed by A, as in most clouds, a simple combinatorial matchmaking can be performed. Otherwise, a complex assignment and satisfiability problem needs to be solved. For constrained devices, we propose a depth-first recursive tree search where after each candidate assignment its validity is determined by successful matchmaking of the remaining subtree, otherwise rolled back.

3. Rule-Based and Weighted Matchmaking Concept

The matchmaking process guarantees that if a deployment of A including its constituent parts, e.g. microservices, to R is possible, a valid deployment plan is returned.

3.1. Design and Architecture

RBMM's matchmaking component operates as a service to be queried by deployment tools after scanning the composition description and artefacts to be deployed. Fig. 2 outlines the process of acquiring the factors through automated scanning (acquirer tools) and manual curation, creating an instance of the models of A and R, and submitting them to the matchmaker to yield a resource-aware deployment plan.

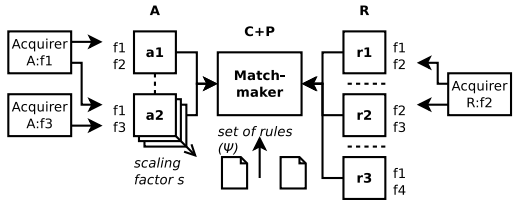

Figure 2. Matchmaking based on collected factors, rules and constraints

3.2. Matchmaking Algorithms

The goal of the matchmaking process is to achieve an optimal deployment by creating, rating and ranking all possible assignment combinations of $A \mapsto R$. We briefly describe an exemplary *fast combinatorial algorithm* assuming infinite resource availability and a more thorough *recursive tree search* assuming finite resources. These algorithms are suitable for simple scenarios but would be replaced with more capable ones in actual deployment systems.

3.2.1. Combinatorial algorithm. The iterative combinatorial algorithm attempts to perform a mapping of all resources on all application parts. For any part a mapping has been found and validated according to deployment rules, the accumulation rules are applied. As further resources are then skipped, the algorithm complexity is approximately $O(n \times \frac{n}{2})$ for $|A| = |R| = n$.

3.2.2. Tree search algorithm. In the recursive tree search, again for each application part a mapping is attempted. Any successful mapping of a_i applies the accumulation rules, followed by a recursive invocation of $A \setminus a_i$, i.e. the set of application parts without the one already mapped. In case the invocation returns a valid result, it is proven that all application parts have been mapped successfully. Otherwise, the accumulation is reversed and the next resource is mapped for a_i. The complexity is approximately $O(n \times \frac{(n-1)^2}{2})$.

4. Implementation

4.1. Acquisition Tools

We have implemented and integrated MAO to automatically produce non-functional property metrics for Docker containers, Docker compositions, serverless applications packaged as Serverless Application Model (SAM), and various other formats typically used in cloud and fog software. In contrast, we have not implemented the automatic acquisition of metrics for resources but acknowledge the existence of the aforementioned approaches to do so.

For the example of Docker images, there are currently two separate factor acquirer tools in production and others under development. One collects public image metadata from Docker Hub, including supported system architectures and artefact size. The other uses the Clair scanner to scan images, producing a report on the number and severity of security issues according to Common Vulnerability and Exposures (CVE) present in the image. These reports are retrieved from the knowledge base and compiled into a merged tree of image factors. Listing 1 shows a typical output of MAO for a single Docker container retrieved from Docker Hub, determining which resources this image can be deployed to and what security constraints apply to it. The combined tree format allows for direct feeding into the matchmaker.

Listing 1. Source of application factor acquisition

```
{ "image": "docker.io/library/mongo:4.2",
  "architecture": "amd64",
  "features": "",
  "variant": null,
  "digest": "sha256:93f3dc8491f23d507...",
  "os": "linux",
  "os_features": "",
  "os_version": null,
  "size": 164677487,
  "CVEs": {
    "Medium": 12,
    "Low": 25,
    "Negligible": 11 } }
```

4.2. Matchmaker Library

First, we implemented the matchmaking algorithms as a Python library, and complemented it with a test tool to synthetically generate applications, resources and rules.

In an experiment with 10'000 application parts and the same number of resources, i.e. 100 million possible combinations, around 488 million factor comparisons were generated. On a single-core Intel i7 processor with 2.60 GHz,

using only deployment and accumulation rules, the iterative combinatorial matchmaking took 3.3 s.

For a more modest scenario with 200 application parts and resources, i.e. 40'000 possible combinations and 175'275 factor comparisons, the combinatorial matchmaking finished in less than 0.05 s. For this scenario, the recursive algorithm implementation became feasible and finished in 80.1 s.

4.3. Emulator Integration

To demonstrate the practical usefulness of RBMM, we integrated the resource-aware deployment of applications to continuums with OsmoticToolkit, an emulation environment to create and interconnect container-based deployments with support for Osmotic Computing. The research area of Osmotic Computing falls into the accomplishment of a new paradigm able to deal with a scalable, interoperable, configurable solution for delivering IoT applications in complex, heterogeneous and dynamic computing environments. In particular, the paradigm [6] looks at the opportunistic management of IoT MicroELements (MELs) to improve QoS and networking management, security, interoperability, and efficiency of next-generation IoT applications. The OsmoticToolkit is the emulation system able to design and test workflows based on MELs in particular conditions where the edge does not have computation and permanent networking capabilities. OsmoticToolkit is based on MaestroNG, a Docker container orchestrator similar to Docker Compose. As OsmoticToolkit has its own matchmaking logic based on cost functions and the Hungarian algorithm [7] covering a certain set of metrics, including CPU and memory, we activate the skipping rule `Context` in RBMM to restrict the matchmaking to contextual factors. OsmoticToolkit allows to model from-scratch infrastructure topologies and applications using graph theory. On an abstract level, the infrastructure topology is modeled as a directed graph $T = (V, E)$, where vertices V is a set of resources, and E is a set of two-sets (sets with two distinct elements) of vertices, whose elements are network links between them. Each graph vertex is annotated with appropriate metadata including computing properties, while different network parameters characterise each link (e.g., latency, bandwidth, packet loss). Each compute resource is emulated leveraging Docker. Similarly, applications deployed in an osmotic ecosystem are structured as a graph $P = (V, E)$, where vertices are represented by MicroELements (MELs) and links (E) by their interconnections for inter-service communication. Thus, RBMM's R/A model is directly mapped to OsmoticToolkit's T/P model.

On a technical level, OsmoticToolkit extends MaestroNG's YAML-based schema by adding support for the decisional factors specified in Table 1. OsmoticToolkit associates the concept of pipeline to an application. Namely, the pipeline's anatomy describes MELs properties and how they are interconnected. Listing 2 illustrates with a code fragment how MEL's constraints can be expressed. The logic is implemented in the Osmotic Orchestrator that uses a two-phase optimisation approach to find the most appropriate deployment plan resources for an application. An application's constraints are classified into hard constraints and soft constraints. Hard constraints refer to must-have requirements which persist and are invariant during execution, such as CPU, memory, or latency; soft constraints refer to desired requirements which can change or be re-prioritised, such as the cost of consuming resources.

Listing 2. MaestroNG YAML format representation of application factors

```
ship_provider : dynamic # static
name : p1
# ships :
    # ship1 :
        #  ip : x.x.x.x
services :
    foo :
        image : ubuntu
        security_opt : [zone==intranet,
            vulnerability==backdoor, consistency
            == true]
        requires: [test]
        labels :
            constraint :
               runtime : python3
               complexity : high
               latency : 5ms
               duration : 900s
       limits :
            memory : 50m
            cpu : 1
       instances :
            foo-1 :
                # ship : ship1
```

The classification of requirements into hard constraints or soft constraints depends on user's need. For example, network latency can be classified as soft constraints if an application is not latency sensitive; however, one could classify latency as hard constraints if the application's response time must not exceed certain threshold limit.

The first phase selects a set of resources by ensuring that all application context-based constraints are satisfied. This is accomplished by executing the RBMM. The second phase balances the complexities of cost and resource-based constraints. This phase involves the Hungarian algorithm. In OsmoticToolkit, optimal deployment is treated as an assignment problem. Generally, each assignment problem is associated with a matrix called the cost or effectiveness matrix. The rows contain the workers or compute resources we wish to assign, and the columns comprise jobs or MELs we want to assign to them.

The cost function c_{ij} used to compute the cost matrix used by the Hungarian algorithm is given in Equation 1.

$$C = \sum_{i=1}^{N} \sum_{j=1}^{M} w_{ij} \times factor_{ij} \times f_j(t) \tag{1}$$

Where i varies from 1 to the number of compute resources N, j varies from 1 to the number of factors to be considered. w_{ij} is a weight between 0 and 1 allowing to prioritise each factor, $factor_i$ is an array containing the values of the factors mentioned above, and $f_j(t)$ represents that

these factors are time-variant. We consider the cost function, which is defined as the weighted sum of the above three parameters. The Hungarian algorithm assigns each compute node satisfying the computation requirements one or more MELs by minimising the previously defined cost function. The output of the algorithm is an optimal deployment plan for the pipeline for describing the MELs contextualization across the Cloud, Edge, Fog, and IoT compute nodes.

4.4. Limitations

Although RBMM is among the most user-controllable matchmakers, we point out in summary the limitations to define future research paths. These are (i) lack of support for dynamic external context factors and redeployment calculation when runtime factors change, (ii) no specific consideration of data flows, network links and microservice interconnects as first-class citizens, that would also help OsmoticToolkit achieving more accurate emulation, (iii) non-optimised recursive algorithm that may limit large deployments with more than a few hundreds of microservices, and (iv) no integration of automatic acquisition of resource characteristics.

5. Previous and Related Work

Matchmaking and mediation has been a traditional topic of research in service-oriented systems design, in particular around semantic web services communities with two decades of history. Early works used heavy semantic modelling, for instance using the Web Service Modelling Ontology, with the advantage of being able to express almost arbitrary details while at the same time requiring a lot of effort to maintain and extend the descriptions [8]. Early systems like ConQo enabled matchmaking between service providers and clients [9], and expressive ontologies like WSMO4IoS modelled cloud providers and their service characteristics, although failed to account for the application-side properties [10] and most only worked on single services, not on sets of services. Ontologies were also proposed in alternative approaches for cloud service composition matchmaking [11], but specifically tailored for virtual machines rather than today's variety beyond cloud platforms, and assuming existing knowledge repositories. The composition objectives were defined as compatibility, total cost, total deployment time or total reliability.

Recent approaches distinguish text-based matchmaking equivalent to full-text searching [12] and attribute-based matchmaking [13]. However, no matchmaker with factor acquisition specifically designed for heterogeneous multi-target deployment of composite software is known, which is a necessity when automating the management of future computing continuums.

6. Conclusions

RBMM performs versatile rule-based matchmaking between composite applications and distributed resources. It advances the state of automation for computing continuums and osmotic processes around IoT, fog, edge and cloud deployments. The RBMM implementation is publicly available as open source library[1]. More challenges have to be resolved in order to make programmable continuums as straightfoward and as widely accepted as centralised programmable infrastructure offered by commercial cloud providers. We point out the open research questions related to the four current system limitations. Additionally, the applications themselves will need to gain more awareness of how they are deployed in order to adapt and for instance offer higher resilience by replicating data if the deployment topology permits, based on an initial matchmaking goal of prioritising resilience over other holistic application characteristics.

References

[1] D. Balouek-Thomert, E. G. Renart, A. R. Zamani, A. Simonet, and M. Parashar, "Towards a computing continuum: Enabling edge-to-cloud integration for data-driven workflows," *Int. J. High Perform. Comput. Appl.*, vol. 33, no. 6, 2019.

[2] N. Seydoux, K. Drira, N. Hernandez, and T. Monteil, "EDR: A generic approach for the distribution of rule-based reasoning in a cloud-fog continuum," *Semantic Web*, vol. 11, no. 4, pp. 623–654, 2020.

[3] P. Gkikopoulos, "Data Distribution and Exploitation in a Global Microservice Artefact Observatory," in *2019 IEEE World Congress on Services, SERVICES 2019, Milan, Italy, July 8-13, 2019*, pp. 319–322, IEEE, 2019.

[4] J. García-Galán, P. Trinidad, O. F. Rana, and A. R. Cortés, "Automated configuration support for infrastructure migration to the cloud," *Future Gener. Comput. Syst.*, vol. 55, pp. 200–212, 2016.

[5] A. V. Dastjerdi, S. K. Garg, O. F. Rana, and R. Buyya, "CloudPick: a framework for QoS-aware and ontology-based service deployment across clouds," *Softw. Pract. Exp.*, vol. 45, no. 2, pp. 197–231, 2015.

[6] A. Buzachis, A. Galletta, A. Celesti, L. Carnevale, and M. Villari, "Towards Osmotic Computing: a Blue-Green Strategy for the Fast Re-Deployment of Microservices," in *2019 IEEE Symposium on Computers and Communications (ISCC)*, pp. 1–6, 2019.

[7] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics*, vol. 2, pp. 83–97, 1955.

[8] T. Vitvar, M. Zaremba, M. Moran, and A. Mocan, "Mediation using wsmo, WSML and WSMX," in *Semantic Web Services Challenge, Results from the First Year*, vol. 8 of *Semantic Web And Beyond*, pp. 31–49, Springer, 2009.

[9] G. Stoyanova, B. Buder, A. Strunk, and I. Braun, "ConQo – A Context- and QoS-Aware Service Discovery," in *Proc. IADIS Intl. Conference WWW/Internet*, October 2008. Freiburg, Germany.

[10] J. Spillner and A. Schill, "A Versatile and Scalable Everything-as-a-Service Registry and Discovery," in *CLOSER 2013 - Proceedings of the 3rd International Conference on Cloud Computing and Services Science, Aachen, Germany*, pp. 175–183, SciTePress, 2013.

[11] A. V. Dastjerdi and R. Buyya, "Compatibility-Aware Cloud Service Composition under Fuzzy Preferences of Users," *IEEE Trans. Cloud Comput.*, vol. 2, no. 1, pp. 1–13, 2014.

[12] Y. Liu, T. Zhu, Y. Jiang, and X. Liu, "Service matchmaking for Internet of Things based on probabilistic topic model," *Future Gener. Comput. Syst.*, vol. 94, pp. 272–281, 2019.

[13] X. Li, J. Yuan, E. Li, W. Yao, and J. Du, "Trust-Aware and Fast Resource Matchmaking for Personalized Collaboration Cloud Service," *IEEE Trans. Netw. Serv. Manag.*, vol. 16, no. 3, pp. 1240–1254, 2019.

1. RBMM code: https://github.com/serviceprototypinglab/rbmm