QoS-aware Genetic Cloud Brokering

Gaetano F. Anastasi[a], Emanuele Carlini[a], Massimo Coppola[a], Patrizio Dazzi[a]

[a]*Information Science and Technologies Institute, National Research Council (ISTI-CNR), Pisa, Italy*

Abstract

The broad diffusion of Cloud Computing has fostered the proliferation of a large number of cloud computing providers. The need of Cloud Brokers arises for helping consumers in discovering, considering and comparing services with different capabilities and offered by different providers. Moreover, consuming services exposed by different providers may alleviate the vendor lock-in issue. While it can be straightforward to choose the best provider when deploying small and homogeneous applications, things get more challenging with large and complex applications. In this paper we propose QBROKAGE, a genetic approach for Cloud Brokering, aiming at finding Infrastructure-as-a-Service (IaaS) resources for satisfying Quality of Service (QoS) requirements of cloud applications. Our approach is capable of evaluating such requirements both for the single application service and for the application as whole. We performed a set of experiments with an implementation of such broker, by considering three-tier applications and scientific application workflows. Results show that our broker can find near-optimal solutions even when dealing with hundreds of providers, providing optimized deployment solutions that includes data transferring cost across multiple clouds.

1. Introduction

Cloud Computing is nowadays one of the most popular computational paradigms. It has been adopted by many companies and considered by many more others for the unquestionable benefits offered, such as potential cost reductions offered by the pay-per-use model, flexibility and scalability, fault-tolerance and increased availability due to the geographic distribution of resources.

Many Cloud providers implicitly force their customers to use proprietary interfaces, virtualization technologies, and communication protocols, so that the cost of switching from that provider's technology to another one would be too high and the migration become materially unfeasible for the customer (i.e. the so called vendor lock-in). For protecting themselves against vendor lock-in, some small to mid-sized businesses (SMBs) may decide to under-invest or simply hesitate to adopt Cloud Computing. Recent surveys [1] also point out that some SMBs are forgoing Cloud Computing because of security and trust reasons, being afraid of losing control on their data, worrying about reliability, integrity and compliance with data privacy laws.

Recently, the adoption of multiple clouds for running cloud-based applications and services has been considered as a mitigation factor towards the vendor lock-in issue. In addition, a multi-cloud environment may be beneficial to cloud-based applications in many other ways. For example, some application services may have special functional and/or non-functional demands that cannot be fulfilled by a single target cloud. In this case, considering a multi-cloud scenario is simply mandatory. Moreover, the multi-cloud scenario can show its advantage in terms of cost-saving for the users: since different services may have different requirements, simply choosing the cheapest provider by considering a single resource may not be cost-effective.

Two orthogonal approaches are commonly exploited for addressing deployments across multiple clouds: Cloud Brokering and Cloud Federation [2]. Cloud Brokers can leverage abstraction APIs, such as Apache Libcloud[1] or Delta Cloud[2] for allowing users to exploit different providers at the same time whereas Cloud Federations provide common platforms providers must be compliant with. Even if Cloud Federation may subsume the Cloud Brokering approach, they can be considered orthogonal from the viewpoint of the goals they pursue. In fact, if on the one hand a Cloud Broker should always consider user profits neglecting provider ones, on the other hands the Cloud Federation must operate a trade-off between these two apparent discording objectives, for example ensuring fairness in exploiting resources belonging to the federated providers.

Additionally, such approaches can help to overcome the trust problem that limits the adoption of Cloud Computing, for instance by selecting time by time providers that are most suitable to fit the security needs of the users. As an example, the user may want to choose a particular provider location when submitting applications for ensuring law compliance in data management. Recent advances

Email addresses: `g.anastasi@isti.cnr.it` (Gaetano F. Anastasi), `emanuele.carlini@isti.cnr.it` (Emanuele Carlini), `m.coppola@isti.cnr.it` (Massimo Coppola), `p.dazzi@isti.cnr.it` (Patrizio Dazzi)

[1]http://libcloud.apache.org/
[2]https://deltacloud.apache.org/

in this research field, designed and developed in the Contrail approach to Cloud Federation [3, 4, 5], treat security needs by explicitly addressing Quality of Protection (QoP) terms as a special case of Quality of Service (QoS).

One of the most relevant research challenges focuses on the problem of scheduling complex applications by respecting user constraints, that have to match the providers' offer. The related aspect to consider is the number of worldwide providers. While it can be considered acceptable to manually search for resources on handful of providers, this task becomes unfeasible when the number of providers grows up to hundreds.

To address this issue we conceived, designed and developed QBROKAGE, a Cloud Brokering approach that provides an optimized deployment solution for a cloud-based application across multiple clouds. QBROKAGE exploits only the information that commercial providers are likely to made available for customers, such as Virtual Machine (VM) costs and their features in term of storage, memory, etc. Let us consider a scenario in which customers submit their applications to QBROKAGE requesting for a deployment configuration that meets QoS requirements, that could be formally expressed by Service Level Agreements (SLAs). Such requirements may involve both non-functional aspects, such as security capabilities of providers, and functional aspects as coming from other specification formats, such as the Open Virtualization Format (OVF [6]). For example, application requirements may specify that VMs require at least a certain amount of memory, and a minimum number of physical CPUs, along with the exact match of geographic location where to place specific parts of the application. Such requirements are used as constraints by QBROKAGE for choosing a set of Cloud providers that can host the services (appliances) and at the same time guaranteeing the respect of the QoS negotiated for the whole application.

In such context, QBROKAGE advocates the exploitation of a Genetic Algorithm (GA) to match services and Cloud resources. GA is a well-known heuristic approach that permits to iteratively find near-optimal solutions for NP-hard problems in large search spaces. Being an heuristic approach, it usually has a computational advantage w.r.t. optimal algorithms and thus it is suitable for being leveraged in an interactive service like our broker. Moreover, our work leverages the GA approach because its model is flexible enough to support multiple constraints at the same time and the injection of additional constraints in the future with minimal interventions on the algorithm. Clearly, this is a crucial aspect for software reuse in the context of Cloud Computing, where QoS models are continuously enriched as providers support QoS guarantees previously not addressed, such as soft real-time guarantees for virtualized services [7] or multi-user virtual environments [8].

1.1. Paper Contributions

The main contribution of this paper is the design and implementation of a generic framework supporting cloud brokering. Such framework embodies a genetic algorithm driving the allocation of applications, which is designed and implemented by considering the following software requirements:

- meeting the heterogeneous QoS requirements of applications;
- finding near-optimal solution according to customers preferences trying at the same time to mitigate vendor lock-in;
- supporting providers with different cost models;
- scaling up with hundreds of providers, while maintaining interactivity.

Several capabilities of QBROKAGE has been already presented in a previous paper [9]. With respect to that paper, we advanced with the implementation of our prototype and we extend our work with the following new contributions:

- we extend our conceptual framework by introducing QoS constraints with *global scope* i.e., constraints that cannot be evaluated by considering VMs in isolation;
- we add the capability of considering network characteristics by implementing two types of QoS constraints with *global scope*, i.e., cost and bandwidth;
- we evaluate our approach in terms of network awareness by setting up some experiments targeting well-known scientific application workflows;
- we further study the scalability of QBROKAGE, presenting an additional experiment to this purpose.

To foster further research in this field and to make our results reproducible, we made the source code and dataset publicly available [10].

1.2. Paper Outline

In Section 2, we present our work with respect to the state of the art. The model proposed in this paper is presented in Section 3. The reference architecture for QBROKAGE and an insight on the algorithm are given respectively in Section 3.3 and Section 4.1. This paper also provides an experimental evaluation of QBROKAGE by means of simulations (Section 5), including a comparison with a state of the art approach, the tuning of the genetic algorithm, scalability performances, the capability of mitigating vendor lock-in and the QoS evaluation of a scientific workflow application when mapping it to multiple clouds.

2. Related Work

2.1. Cloud Brokering

In the research community there is a wide consensus on the importance that brokers can have on Cloud environments for helping consumers in discovering, considering and comparing services with different capabilities as offered by different providers [11]. The need of brokering mechanism particularly arises in Cloud Federation architectures, such as Intercloud [2], the first approach going in the direction of building a unified platform composed by federated providers that can exchange information through super-entities (e.g. the Contrail approach) or as peers (e.g. the Sky [12] approach).

Recently, Cloud Brokering architectures are acquiring importance as well, for dealing with providers that are loosely-coupled or not coupled at all. STRATOS [13] is a cloud broker service that permits to deploy and manage cloud applications on multiple providers, based on requirements specified in higher level objectives. That work shares many similarities with ours and they are compared in Section 5 by considering a scenario that was possible to reproduce. One of the most notable differences between the two works is the formalization of the problem, STRATOS embodies a multi-criteria optimization problem, whilst our work is based on genetic algorithms. Another remarkable difference regards the application description: STRATOS uses a custom XML representation for describing the application as a set of clusters and nodes, whilst our broker accepts OVF [6] description as input and constructs from it a graph representation of the application, in terms of nodes and edges. Since OVF is a promising standard description approach, we believe this capability may help in the adoption of QBROKAGE. Finally, STRATOS supports elasticity allowing for adding VMs to running applications, whilst our work does not address this aspect.

Ngan and Kanagasabai propose a semantic cloud broker [14] based on ontology matching that can cope with semantic interoperability issues caused by different non-standard ways of exposing provider capabilities. Focusing on the same approach, the authors propose a benchmark framework for cloud brokers [15], based on five different degrees of difficulty. Our work does not use ontology matching since we are focusing on QoS parameters that can be quantifiable and uniformed quite easily. Nevertheless, we believe that our work could be complemented with such approach for considering semantic fuzziness in exposing provider capabilities.

Another ontology-based approach that supports multi-criteria optimization has been recently proposed by Zhang et al. [16] for a cloud service recommendation system. This approach shares some similarities with QBROKAGE: for instance it considers QoS parameters for respecting SLAs and it takes into account user preferences for giving priorities to parameters. However, the work by Zhang et al. uses Analytic Hierarchy Process (AHP) as inner mechanism for comparing different criteria and proposing an ordered list of suitable offers. The AHP method uses thresholds only at the end of the process, after operating a pair-wise comparison between criteria. Thus, there is no fine-grain control on each QoS constraint to be satisfied. This is the main difference with our approach: as QBROKAGE uses thresholds for each parameter, each solution can respect all the constraints and at the same time it can take into consideration a QoS parameter more than another for proposing a preference order. Moreover, our work supports QoS specification on a per-VM basis, whilst the work by Zhang et al. seems to support QoS specification on a per-application basis only.

2.2. Genetic Algorithms in Cloud Computing

Several works employs genetic algorithms to place VMs in cloud computing environments. Genetic approach are fairly used in cloud scheduling, in which the problem is to place VMs within a single datacenter. In this context, the work by Pop et al. [17] focuses on the scheduling of independent tasks based on the reputation of resources. Although their model is quite different from ours, their insight into genetic operators could be leveraged in our work for boosting performances in terms of evolutionary steps for population convergence. Another relevant work has been done by Zheng et al. [18] who address the problem of resource scheduling in Infrastructure as a Service (IaaS) Cloud. They focus on parallel genetic algorithm for speeding the resource allocation process and improving the utilization of system resources. By comparison, QBROKAGE is considered as a service that requires user interaction and thus optimizing execution times of the broker is not our primary goal. However, we do not exclude to explore parallel genetic algorithm in the future. The work of Mark et al. [19] considers a genetic algorithm to place VM into a datacenter that has two different cost models: reservation and on-demand. Their work mostly focuses on the prediction of resources usage, while QBROKAGE focuses on the multi cloud environment.

In multi-cloud brokering, genetic algorithms have been exploited by a few of approaches. In particular, Iturriaga et al. [20] and Heilig et al. [21] employ genetic algorithms for the placement of VMs in multi-cloud environments. These proposals mostly focus on the optimization of the execution time of the broker, for instance by leveraging parallel genetic algorithms [20]. Although these works present some similarities with our proposed solution, unlike QBROKAGE they do not consider network dependencies as one of the requirements when placing VMs on different cloud providers. Jrad et al. [22] propose a genetic-based broker targeting services composition in healthcare workflows. Their work share several similarities with QBROKAGE, such as the ability to take into consideration QoS constraints in complex applications. The main difference with our approach is that they explicitly target service composition and thus they evaluate a solution by considering the application as a whole, without caring about the fitness of the placement of a single VM in isolation. In the terminology used in this manuscript for presenting our work, that would mean to consider only the global fitness of a solution, rather than considering both global and local fitness as we do.

Wen et al. [23] focus on partitioning workflows over federated clouds for optimising monetary costs and leverage genetic algorithms for this purpose. Authors design the problem as a bi-objective optimisation problem (security and costs), however they solve it as a single objective optimization by pre-constructing a candidate list composed by those clouds that respect constraints. Thus, the main difference with our approach is that QoS requirements are not part of the genetic model nor they are used in the genetic algorithm itself.

2.3. Workflow computation in Cloud Computing

As part of the evaluation of our Cloud broker, we refer to a scientific workflow application as a case study. Cloud Computing is gaining momentum in both academia and industry and thus there is a growing interest in applying scientific workflow systems over the Cloud [24, 25]. As Zhao et al. identify in their work [24], one of the major benefits of managing and running scientific workflows on top of the Cloud is the opportunity to improve the performance/cost ratio, by trading-off between the workflow requirements and the system resources offered by Cloud providers. A study in this direction has been conducted by Deelman et al. [25], that used the Amazon Cloud as reference for examining the cost of running an astronomy workflow application with different resource provisioning plans.

This paper differs from the described work in studying the performance/cost tradeoff by exploring the possibility of using different providers for the application deployment. There are several works studying multi-cloud workflows [26, 27, 22, 23, 28, 29]. Jrad et al. [27] leveraged WorkflowSim [30] for simulating workflow executions and evaluating cost and performance gains of running workflows on a multi-cloud environment. Their work resembles QBROKAGE for considering QoS values coming from the user in the brokering process. However, their work does not consider network and storage costs, as instead we do in QBROKAGE. This aspect, combined with the fact that they have not described datacenter costs nor published their complete dataset, make that work hardly comparable with ours. To foster future research on this topic, we publish the dataset used in this work along with the source code of QBROKAGE.

Another related work has been done by Coutinho et al. [28] that developed GraspCC-fed, an approach aiming at dimensioning the amount of VMs to allocate for workflow execution in both single cloud provider and federated clouds. That authors also compare their approach with a genetic algorithm-based approach in a single provider scenario. However, their approach has a different goal with respect to our approach. We do not aim at dimensioning the VM number because in QBROKAGE the number of VMs to allocate is chosen by tenants. Instead, QBROKAGE has been conceived for choosing the VMs that best fit the user QoS requirements among the different provider configurations in multi-cloud environments, including cloud federations. In our current implementation, the number of VMs remains fixed during the workflow execution, as GraspCC-fed does. However, we plan to extend our work allowing tenants to dynamically adjust the number of VMs during the execution.

Table 1: Comparison of selected work

	QBrokage	[22]	[23]
User requirements			
Cost	X	X	X
Performance/QoS	X	X	-
Dynamism	-	-	X
Application Requirements			
Workflow size	Large	Very large	Very large
VM Number	Large	Medium	Small
Infrastructure			
Provider Number	Large	Medium	Small
Provider VM Types	Large	Small	Small

2.4. Comparison of selected work

For the reader's convenience, the proposed approach is further compared with selected work [22, 23] that most resembles QBROKAGE, both for leveraging genetic algorithm over federated clouds and for using workflow applications in the experimental system evaluation. In particular, the difference between this work and the related one is emphasized by focusing on the system evaluation and comparing the different experimental setup parameters. Table 1 summarizes such differences by leveraging the scale defined in Table 2. It can be seen that the proposed approach focuses on matching application VMs over Cloud providers when both are characterized by an high level of variability for the specific parameters. Related work [22, 23] focuses on the workflow size in terms of task numbers, however such tasks are mapped into a lower number of VMs w.r.t. the proposed approach. Finally, the work by Wen et al. [23] is the only one that currently deals with the dynamic nature of the Cloud – we left such aspect as future work. However, the presentation of the work by Wen et al. [23] do not focus on performance and cloud configuration but only covers cost minimization. In addition, although the conceptual framework described by Wen et al. [23] seems to support local and global specification, such aspect is not analyzed in that work. Instead, the proposed approach examines in-depth the tradeoff between local and global optimization for allocating VMs over federated clouds.

Table 2: Scale definition

	Value Range
Small	1-9
Medium	10-99
Large	100-999
Very Large	1000

3. Model and Architecture

In this section we initially present our reference model of applications and resources. This model is partially derived from a previous work of the same authors [31]. Subsequently, the reference architecture of our broker is presented.

3.1. Applications

An application α is represented as a Direct Acyclic Graph (DAG) $\langle\ \mathbb{G}_N,\ \mathbb{E}_M\ \rangle$, where $\mathbb{G}$ represents the set of N vertices and $\mathbb{E}$ represents the set of M edges connecting the vertices. Each vertex $g_i \in \mathbb{G}$ embodies an *appliance* each one potentially providing different services, e.g. one appliance provides a firewall and another one provides a back-end database. Each appliance can be composed by multiple correlated Virtual Machines ($vm_{1..K} \in g_i$) but for the sake of simplicity and to ease the presentation we assume that each appliance is composed by only one VM, and thus the terms appliance and VM can be used interchangeably. We have released this assumption in the experiments in Section 5. Each edge $e_{i,j} \in \mathbb{E}$ represents a communication path connecting vertices g_i and g_j and directed from g_i to g_j. For each vertex g_i we define its adjacency list adj_i as the unordered list containing all edges coming out from g_i, i.e. $adj_i = \{e_{i,k} | g_k \in \mathbb{G}\}$.

3.2. Resources

In this work we consider as resources what IaaS cloud providers offer to their customers. Each provider is modeled with a datacenter p_i composed by a set of hosts, that can run one or more VMs depending on their availability of resources. The resources of each datacenter are interconnected by a network characterized by a specific set of features, such as bandwidth, latency and security capabilities. Depending on its performance capabilities each datacenter can run a different number of applications, i.e. set of VMs. Each datacenter is characterized by a set of resources and exposes its limits, i.e. the maximum amount of resources that can be assigned to a VM in order to be run on that provider.

To catch the heterogeneity of current pricing models adopted by cloud providers, we consider both providers adopting a *per-resource* cost model and providers adopting a *per-VM* cost model. In the per-VM case, we model cloud providers as capable of running predefined type of VM provided by customers and charging them for the whole VM used over time. In the per-resource case, we model cloud providers as capable of running each type of VM provided by customers (up to the datacenter limits) and charging them per unit of used resources over time.

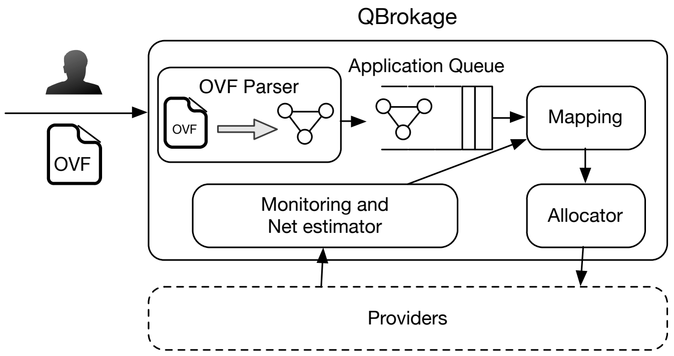


Figure 1: Architecture of QBROKAGE Cloud Broker

3.3. Architectural Model

Figure 1 depicts an high-level representation of the QBROKAGE architecture. In this section we give an insight on it, although an in-depth description of each module is not in the scope of this paper, which instead focuses on the exploitation of a genetic approach to drive the allocation of cloud applications.

An actor submitting its applications to QBROKAGE is represented on the left of Figure 1. Applications are represented by means of the OVF, that permits to describe applications in terms of their building blocks, the appliances. Each appliance is composed by a set of VMs, interconnected by Virtual Networks. The OVF allows to associate requirements both to VMs and Virtual Networks.

The module that receives such OVF is the **OVF Parser**, depicted on Figure 1 inside the box representing QBROKAGE. This is the component responsible of parsing OVFs given in input, for constructing an application graph as defined in Section 3. A sophisticated OVF Parser, supporting a configurable behavior and custom XML tags to drive a proper transformation of OVF-based representations to graph representations of applications, is a key module for our Cloud Broker. The OVF Parser we adopt in our architecture is the one provided by the *OVF Toolkit.* The OVF Toolkit software is a standalone library that we developed and can be used to parse, validate, manage and render OVF files. Essentially, it aims at organizing the information on applications described in OVF files, making it easily available to others software modules through an API and a set of functionalities. A more in-depth description of the OVF Toolkit, including the Parser component, can be found in a dedicated paper recently presented [32].

Once transformed in a graph, a representation of the input application is then enqueued on the **Application Queue**, as depicted in Figure 1. In this way it is possible to host applications that need to be mapped onto one or more cloud providers among the ones managed by QBROKAGE. In principle, many different behaviors and structuring can be applied to this queue, however in our work, for the sake of simplicity, we adopted only a *M/M/1* queue. After their transition through the queue, application mapping requests are delivered to the Mapping component.

Quite straightforwardly, the goal of the **Mapping** module is to compute a set of valid mappings for each application. To this end, it considers the information that characterize both the datacenters and the applications. The information exploited by this component can either be static or dynamic. Static information can be easily stored by QBROKAGE and accessed when needed. For instance the hardware description of the resources, the size of datacenters, etc. Additional information may come to the Mapping component from the **Monitoring and Net estimator** component, which provides a view on (i) the features associated with the network connecting the datacenters as well as (ii) on the state of the providers. The Mapping module computes the associations between resources and applications by exploiting:

- the representation of applications,
- the information related to datacenters,
- QoS properties.

This compound set of data is built according to the model presented in Section 3. This approach allows for achieving an high degree of flexibility and the ability to compute complex plans without dealing with burden of contacting datacenters every time an application needs to be mapped.

Once one or more mapping plans are computed, those are passed in input to the **Allocator** module. The Allocator tries to allocate Virtual Machines onto Cloud providers by considering a single mapping plan at time. Note, that differently to the Mapping component, the Allocator effectively connects to the Cloud providers for allocating VMs. As a consequence, its activity can lead to failures or exceptions, which are managed directly by the module. Indeed, when an allocation defined by the mapping fails, the Allocator can: (i) de-allocate all VMs allocated so far and starts over with another plan, or (ii) try to allocate failing VMs by exploiting the next plan on the list, leaving already allocated VMs in place. The first approach requires longer allocation time than the second one, but it is needed when the application has functional dependencies among its components. In both cases, the user is contacted only when all mapping plans were considered but none of them succeeded.

3.4. Research challenges

The contribution of this paper deals with the design and the implementation of the Mapping module in the above mentioned architecture. The main task of such component is the design of the algorithm to map the application graph to the available providers, according to the following requirements: (i) the module must consider optimal or near-optimal solutions in term of satisfaction of the QoS, while using different providers when the situation allows it; (ii) the module must be agnostic with respect to the submitted application, i.e. it should be able to deal with different kind of QoS modelling when mapping a give application; (iii) it shall be fast, allowing a quasi-interactive response from the submission to the definition of a plan;

We tackled these challenges by providing the design of the mapping module according to a genetic algorithm, able to map different kind of QoS constraints of application over the consider resource modelling. Such design, as well as some details on its implementation, is presented in the next section.

4. Genetic Brokering

GA can be formulated in many flavors, that differ for the selection of new population, the structure of genetic operators, their combination and so on. In Section 4.1 we explain the proposed algorithm, in which the problem solution is the allocation mapping of appliances to Cloud providers.

In employing GA, a key point is the evaluation of different solutions for choosing the better one(s) that is candidate for solving the problem. Our brokering can be defined QoS-oriented and thus we formulate in Section 4.2 the QoS constraints used for the evaluation. Such formulation is leveraged by the fitness function we conceived and present in Section 4.3.1 and Section 4.3.2.

4.1. Algorithm

In the proposed algorithm, each solution (chromosome) c is a vector of length N, representing the allocation mapping of each appliance, i.e. a single VM (see Section 3), to a cloud provider. In other words, if $c(i) = j$ then the appliance g_i will be allocated on provider j. For example, Figure 2 shows a chromosome representation where appliance g_2 is allocated on provider p_1 and thus $c(2) = 1$.

The QBROKAGE algorithm follows the canonical GA, hence we do not introduce any novelty from the algorithmic point-of-view. However, as discussed in the rest of this section, we provide our novel contribution in codifying the problem solution and the search in state space, by modeling a novel fitness function. For the reader's convenience, we detail the canonical GA of QBROKAGE in Algorithm 1. In particular, the initial population of S_P individuals is randomly generated and each individual is evaluated by considering better individuals with those having higher fitness values. The population selected for mating, is randomly chosen with a rate R_{cross} among the total and the crossover strategy is the random one-point crossover. Thus, a number of $S_P * R_{cross}$ crossover operations are performed for each generation and each operation produces 2 individuals. Then, mutation is applied with a probability of P_{mut} to each gene in each individual. After applying mutation, an elitist selector is used to select

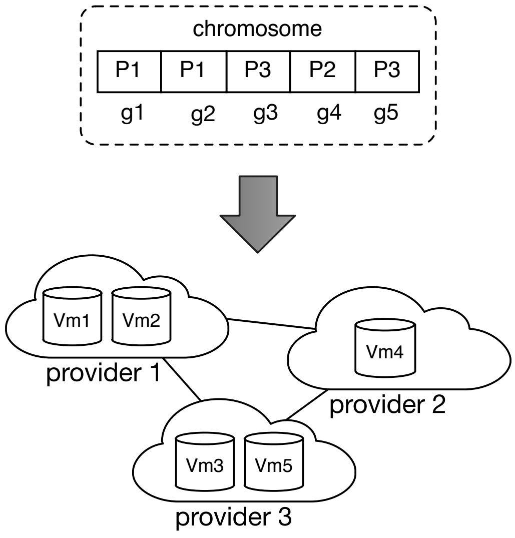


Figure 2: Chromosome representation in QBROKAGE

```
creating_initial_population();
while $termination\_condition = false$ do
    population_evaluation();
    selection_for_mating();
    one-point_crossover();
    random_mutation();
    elitism_selection_new_generation();
    termination_condition_eval();
end
```

Algorithm 1: Canonical GA of QBROKAGE

the top percentage E_{top} of the population size, whilst the remaining is obtained by cloning the best selected up to reach the population size. A discussion on the numeric values to be used for R_{cross}, P_{mut} and E_{top} is deferred to the experimental evaluation described in Section 5.

4.2. QoS Modeling

We consider a set of QoS attributes $\mathcal{Q} = \{q^1, \dots, q^S\}$ that can be classified in three categories, as defined in the work of Ye et al. [33]:

- *ascending* QoS attributes, in which higher values are better than lower ones;
- *descending* QoS attributes, in which lower values are better than higher ones;
- *equal* QoS attributes, in which only equality or inequality is meaningful.

When these attributes are considered in specifying the application, they turn into constraint values to be respected by the infrastructure. Thus, for each attributes we denote as γ_i^j the constraint value related to appliance g_i for the QoS attribute q^j. Instead, we denote as β_i^j the actual value considered for the cloud provider p_i and related to the same attribute q^j. For the sake of simplicity we may omit the index i when it will not be strictly necessary to the comprehension, as happens in the following.

For checking the adherence of constraint values coming from the application to actual values coming from the infrastructure, a set of inequality constraints $\mathcal{QC} = \{QC^1, \dots, QC^S\}$ is built, whose cardinality is the same of set $\mathcal{Q}$. In particular, we have the following cases:

- $QC^j = \gamma^j - \beta^j \leq 0$ in case of ascending attributes
- $QC^j = \beta^j - \gamma^j \leq 0$ in case of descending attributes
- $QC^j = |\gamma^j - \beta^j| - \epsilon \leq 0$ in case of equal attributes (the small constant ϵ is the tolerance range).

When $QC^j \leq 0$ the constraint is respected, otherwise it is not respected. Moreover, the following $-1 \leq QC^j \leq 1$ holds by design, as β^j and γ^j are normalized values with respect to the maximum value for q^j (computed among the set of all providers).

Also, a weight ω^j is associated to each attribute q^j for modeling a search guided by user preferences and/or implementation of particular policies for some attributes. It is out of the scope of this paper to find a user-friendly way for allowing customers to express such preferences. For simplicity we assume that customers can specify an input vector $V = (v^1, \dots, v^S)$ of relative values and each ω^j is calculated by equation $\omega^j = \frac{v^j}{|V|}$ where $|V|$ is the unitary norm of vector V.

4.3. Local and Global Constraints

In the context of the genetic broker defined so far, we consider two different flavors of constrains, local and global. *Local constraints* are considered in isolation, on a per-gene basis. This means that the matching of a particular allocation with a given constraint is evaluated independently from the other constrains. Rather, *global constraints* are consider on a per-chromosome basis, which means that whether or not a particular allocation respect a given constraint depends also on the whole allocation. These concepts lead to the definition of the local and global fitness functions, that are detailed in the next sections.

This diversification is very useful for the purposes of allocating applications in a multi-cloud environment. Some constraints, such as ram, core numbers, etc., have meaning only as a local scope and are not meaningful when considered in global scope. However, some other constraints are meaningful in both local scope and global scope. In particular, the constraints related to the network can be identified as having local and global scope. For instance, if we consider the cost constraint as local, we calculate such a cost of running a VM by only considering the computing requirements and the provider costs for that capabilities. Instead, considering the cost constraint as global, the networking costs of a VM must be also calculated and added to the computing ones. To this end, we need to know the allocation of the connected VMs for an accurate estimation of that cost. In fact, many providers do not charge intra-provider communication but only the inter-provider one. According to these considerations, we compute the networking costs of a VM by summing up the cost for each link specified in its adjacency list.

In general terms, the global fitness function should provide better results in terms of cost, as the allocation as a whole is considered. However, this may result in a longer brokering time, due to the additional complexity of the constraint. This tradeoff is studied experimentally in Section 5.5.

4.3.1. Local fitness function

When evaluating a solution, our algorithm first considers each gene g by defining a column vector $D_g = (d_g^1, \dots, d_g^S)$ as the distance of an allele from constraint satisfaction. For each QoS constraint in the set $\{q^1, \dots q^S\}$, such a distance d_g is computed by using inequalities introduced in Section 4.2. According to the problem formulation, we consider values $-1 \leq QC_g^j < 0$ as those satisfying the constraint j for gene g, with better solutions in minimizing the gradient direction. We recall that each QoS attribute j has associated a weight ω^j.

Since fitness functions are usually modeled as maximization problems, we define the fitness of a gene g as

$$l(\cdot) = AD_g \tag{1}$$

where A is a square matrix

$$A = \begin{pmatrix} a_{1,1} & & 0 \\ & \ddots & \\ 0 & & a_{S,S} \end{pmatrix}$$

with each element in the diagonal defined as

$$a_{jj} = \begin{cases} -\omega^j & \text{if } QC^j < 0; \\ 0 & \text{if } QC^j > 0. \end{cases}$$

Finally, the fitness function of each chromosome c is defined as

$$F(c) = \sum_{i=1}^{N} l(\cdot) * K \tag{2}$$

with N being equal to the number of genes of c and K being equal to a constant defined for awarding those chromosomes with an high number of genes that correspond to allocations respecting constraints.

4.3.2. Global fitness function

With respect to our previous formulation [9], we extended the framework for evaluating genes having mutual relationship among them e.g., a gene may represent a VM that communicates with another VM and there is a QoS constraint on that communication link. To this end, we characterize each constraint q^i with a scope sc that may assume values in the set $\{global, local\}$. If $\mathsf{sc}^i = \mathsf{local}$, the distance d_g^i for a generic gene g will be calculated by considering that gene in isolation i.e., using only information regarding the mapping of g on the provider represented by the allele $c(g)$. Instead, if $\mathsf{sc}^i = \mathsf{global}$ the distance d_g^i for a generic gene g will be calculated by considering not only that gene in isolation but also information regarding all the genes connected with g. We define this information as the adjacency list adj of g, in analogy with the DAG representation of the application α represented by each chromosome. The introduction of the scope notion for a constraint q^i only affects the way its distance d^i is computed and thus values for D_g are likely to change but the formulation given in Eq.1 remains still valid, along with the definition of the matrix A.

Clearly, if $\exists q^i \in \mathcal{Q}, \mathsf{sc}^i = \mathsf{global}$ then the fitness $l(\cdot)$ of that gene g is a function $f(g, \mathsf{adj}_g)$ depending in input by g and all the genes in its adjacency list. Please note that the framework allows for combining different constraints with different scope in the set Q. In fact, the adj_g information is simply discarded when computing the distance d_g^i for a local constraint q^i. The only obvious limitation is that a constraint q^i cannot have both global and local scope at the same time. Finally, the comprehensive formulation of the fitness function when considering the scope of a constraint can be given by refining Eq.2 with the following:

$$F(c) = \sum_{i=1}^{N} l(g_i, \mathsf{adj}_i) * K \tag{3}$$

Clearly, if $\forall q^i \in \mathcal{Q}, \mathsf{sc}^i = \mathsf{local}$ we fall back to the formulation given in our previous work [9], since adj_i is empty for each gene g_i.

4.4. Implementation details

QBROKAGE is implemented in Java and leverages the JGAP framework [34] for performing the described genetic algorithm. The JGAP framework is flexible enough for our purposes, since it allows us to specialize some parts of the framework and at the same time using built-in methods for performing common genetic operations. For instance, by referring to the algorithm of Listing 1, the classic "one-point crossover" operator has not been re-implemented, as it is already present in the JGAP framework and can be used by instantiating an object of the `CrossoverOperator` class. Similarly, the "random mutation" operator can be used by instantiating an object of the `MutationOperator` class and the elitist selector can be used by instantiating an object of the `BestChromosomesSelector` class.

Instead, we need to specialize all the classes related to the fitness function and the genetic representation of the model. For instance, regarding the model we need to extend the `IntegerGene` class of JGAP, which represents a Gene implementation that supports integer values for its allele and provides upper and lower bounds to restrict the range of legal values allowed by that Gene instance. In fact, since our model represents each gene g_i as an appliance and its allele as the index of the provider, we need to carry out for each gene specific information such as the allocation cost of the appliance on that provider and the local fitness.

Regarding the fitness function, we need to extend the abstract class `FitnessFunction` and override the *evaluate* method for implementing the QBROKAGE fitness function as presented in Sec. 4.3.1 and Sec. 4.3.2. The main implementation idea is that we have a class for each QoS constraint q^i and each class expose its methods for calculating the distance d_g according to scopes and attributes. The implemented *evaluate* methods have access to a container of constraints, specified in the configuration phase, for calculating the fitness of each gene and the whole chromosome. Interested readers can further investigate our implementation by referring to the `metascheduler` Java package of our code [10].

5. Experimental Evaluation

This section presents a set of experiments performed for evaluating our broker, both quantitatively and qualitatively. We designed the experiments to focus on the validation, tuning, and scalability of the genetic approach. For simplicity we assume that interconnections between providers have the same network capabilities, i.e. the placement can be conducted by considering each VM in isolation. In future experiments we plan to relax this assumption for reflecting more realistic scenarios. This should be straightforward since our conceptual framework already support the specification of different network requirements/characteristics at both the application and infrastructural level respectively (see Section 3).

We first compare our broker against a state of the art approach. Then, we analyze how the parameters of the genetic approach affect results when dealing with an higher number of providers and VMs. After experimentally tuning our broker, we evaluate it when coping with hundreds of providers and we finally measure the number of providers used in acquiring resources, for giving an estimation of the vendor lock-in degree when using our implementation. The experiments were run in an environment that simulates Cloud Federation and/or Inter-Cloud scenarios. Such environment is provided by SmartFed [31], a cloud simulator we built upon CloudSim [35]. We extended SmartFed functionalities in order to run the presented scenarios described in this section. All simulations have been run on a machine equipped with Java 7, 16GB of RAM, an Intel i5-2550 quad core @3,30 GHz. Unless differently specified, presented results are obtained by averaging the output of 20 independent runs of simulation.

Presented experiments in Section 5.1, Section 5.2, Section 5.3 and Section 5.4 share the same common setup for the QoS attributes of applications. The considered set is the following $\mathcal{Q} = \{costPerVm, ram, storage, location\}$. In this case *costPerVm* is a descending QoS attributes (lower values are better), *ram* and *storage* are ascending attributes and *location* is an equal attributes.

Table 3: STRATOS Application Description

	#VM	CPU	RAM (GB)	Disk (GB)
Load Balancer	1	1	1	160
Web Server	2	1	1	160
Database	1	6	4	160

Table 4: Provider Costs for STRATOS Scenarios ($ per hour)

	EC2	Rackspace	Aruba.it
SmallVertex	0.085	0.240	0.093
LargeVertex	0.680	0.240	0.238

In addition, the experiment described in Section 5.5 exploits two more QoS constraints characterized by a global scope, that are the *CostPerVm_Global* – cost per VM including networking – and *Network_Global* – the network bandwidth required by a VM for communicate with the others. Such constraints are able to discriminate between intra-provider communication and inter-provider communication for adopting a different cost and use model as commonly real cloud providers do. In order to show the potential of such kind of constraints, we simulated the allocation of a computation workflow over different cloud resources. However, it is worth noticing that our broker aims to be a general solution, effectively handling any kind of application formalized in the OVF format (see Section 3).

In all experiments, we are interested to minimize cost for the user and thus we leverage the weight mechanism already presented in Section 4.2 for going in the direction of cost minimization.

5.1. QBROKAGE *Comparison*

QBROKAGE is compared with the state of the art by considering and extending a scenario defined by the authors of the STRATOS broker service [13]. Such scenario, briefly recalled here for the reader's convenience, consider a typical three-tier application composed by a loadbalancer, a web server and a database – 4 VMs in total. In our model, such application can be represented by three vertices, as described in Table 3, where the number of VMs and the desired characteristics of the corresponding VM are indicated for each vertex. As the loadbalancer and the web server have the same requirements, their configuration will be denoted as Small, whilst the database one will be denoted as Large. In the former experiment, the authors of STRATOS consider two providers, Amazon EC2 and Rackspace (RS), which charge customers on a VM basis, at costs indicated in the first two columns of Table 4.

They show that up to 48% are realized when using the broker, reaching the optimum cost value of 0.495 for the whole application. We repeated such experiment with QBROKAGE and we obtained the optimum cost value at the first evolution step with a population $S_P = 20$, as described in the first row of Table 5.

Table 5: QBROKAGE results for STRATOS Scenarios

	Optimal Cost	QBrokage Cost	Alg. Steps
FormerScenario	0.495	0.495	1
ExtendedScenario	0.493	0.493	3

Table 6: Vertex requirements and providers cost and characteristics

	min	max
Vertex requirements and providers characteristics		
memory	512 MB	16 GB
bandwidth	10 KB	10 MB
disk	4 GB	10 TB
cores	1	8
mips per core	1000	25000
Provider cost (in currency per hour)		
memory (per GB)	0.01	0.1
storage (per GB)	0.0002	0.0020

In addition, we extend this scenario with another provider, for leveraging the QBROKAGE capability of considering also providers applying a pricing model that charges customers on resource basis, rather than on VM basis. For this experiment we consider Aruba.it an Italian provider that allows customers to create VMs by specifying the desired quantity for each resource and thus applies a per-resource pricing model. For obtaining prices indicated in last column of Table 4, we use the following prices expressed in dollars per hour (for simplicity we assume that the euro/dollar exchange rate is 1 euro equal to 1.3 dollars – see `http://www.cloud.it/en/cloud-computing/pricing.aspx`): 0.026 for 1 CPU, 0.005 for 1 GB of RAM, 0.0039 for 10 GB of Hard Disk. When considering the three providers in conjunction, QBROKAGE finds the optimal result for cost minimization, which is 0.493 as resumed in the second row of Table 5.

5.2. QBROKAGE *Tuning*

Granted that our broker is able to cope with a small number of VMs and providers, we increased the size of the problem and contemporary studied how different parameters affect the behavior of QBROKAGE, in order to properly tune the genetic allocator. In particular, our purpose is to experimentally find reasonable values for R_{cross} and P_{mut}, which are the main parameters characterizing the genetic algorithm described in Section 4.1. To this end, we consider a case study in which an application is composed by 3 vertex (12 VMs in total) and desired resources must be found among a set of 50 providers, assuming that each provider is capable of accepting an infinite number of VMs.

The requirements for each application vertex and the provider characteristics have been generated by following a uniform distribution, in the ranges defined in Table 6. For

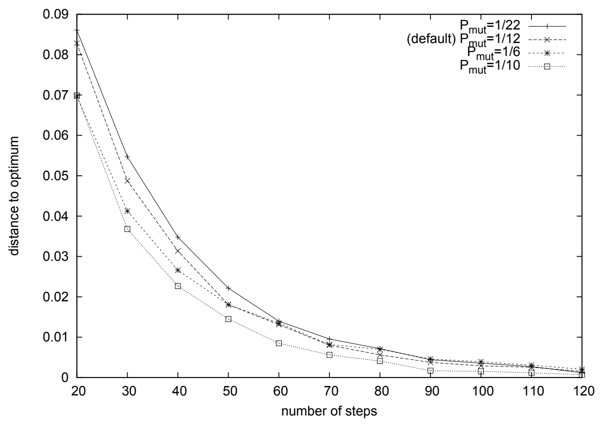


Figure 3: Distance to optimum with $R_{cross} = 0.35$ and different values for P_{mut} by varying number of genetic steps

simplicity, in this case we only use the per-resource cost model for providers, with cost presented in Table 6. Please note that the generation of providers has been performed such to conveniently build a baseline optimal cost allocation. In particular, we generated first the set of providers able to satisfy the QoS with a certain cost, and then we added to the set of available providers either more expensive or unsuitable for QoS requirements. We computed the optimal baselines in this way even for the subsequent experiments, if not explicitly stated otherwise.

In this particular experiment the optimal solution in terms of cost, i.e. the less expensive allocation that contemporary satisfies the QoS requirements, is 58.67. Then, starting from default configuration of JGAP – $R_{cross} = 0.35$, $P_{mut} = 1/12, E_{top} = 0.9$ – we first vary mutation probability and then crossover rate, studying how solutions are distant to optimum for an increasing number of evolution steps.

Figure 3 shows the distance to optimum (where 1 is the maximum distance calculated in the positive space) with $R_{cross} = 0.35$ and $P_{mut} = \{1/6, 1/10, 1/12, 1/22\}$. Although all the configurations reach the optimum within 120 steps, it can be seen that the curve relative to $P_{mut} = 1/10$ is very close to the optimum starting from 90 steps, much earlier than the other configurations. Further increments of P_{mut} does not yield better performances, as seen with 1/6.

Given this result, we try different crossover rates by using configuration with $P_{mut} = 1/10$. Figure 4 shows the distance to optimum with different values for R_{cross}. It can be seen that for $R_{cross} = 0.2$ the distance to optimum is significantly higher than the other configurations, at least up to 60 evolution steps. After that point, all the curves converge to the optimum. The bottom curve, corresponding to $R_{cross} = 0.80$, is the one which yields a measurable advantage with fewer steps. However, increasing R_{cross} leads to longer execution times, for the higher number of operations to be performed. In fact, by measuring the execution times for $R_{cross} = 0.80$ and $R_{cross} = 0.35$, we

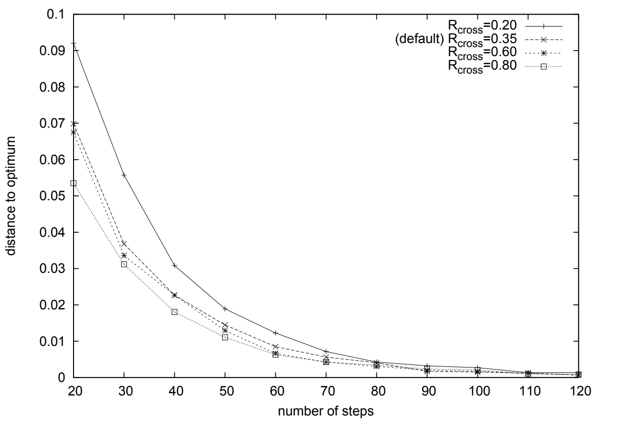


Figure 4: Distance to optimum with $P_{mut} = 1/10$ and different R_{cross} by varying number of genetic steps

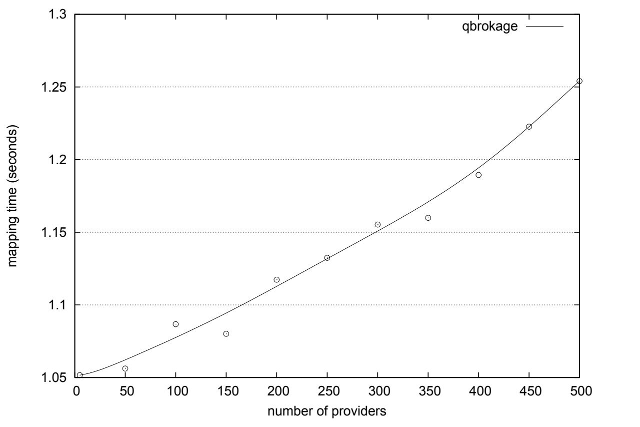


Figure 5: QBROKAGE computation time in seconds, population size 50, up to 500 providers

found that the latter is always completing around $400ms$ early. As an example, in the case of 100 evolution steps, we obtained, on average, an execution time of $903.05ms$ for $R_{cross} = 0.35$ and $1290.35ms$ for $R_{cross} = 0.8$, with a difference on the distance-to-optimum that is quite negligible. For this reason, we decided to operate a trade-off, using the configuration with $R_{cross} = 0.35$.

To resume, as a consequence of the result shown in Figure 3 and 4 and results gathered for execution times (not completely shown for brevity), we choose $R_{cross} = 0.35$, $P_{mut} = 1/10$ as default configuration for QBROKAGE, to be used in the following experiments. For simplicity, when considering the elitism parameter E_{top} in this experiment and in the following, we stick it to 0.9, meaning that we select the top 90% of the population size. However, please consider that lowering such value may further benefit the performance of QBROKAGE in some cases (e.g. a relative low number of evolution steps).

5.3. Scalability

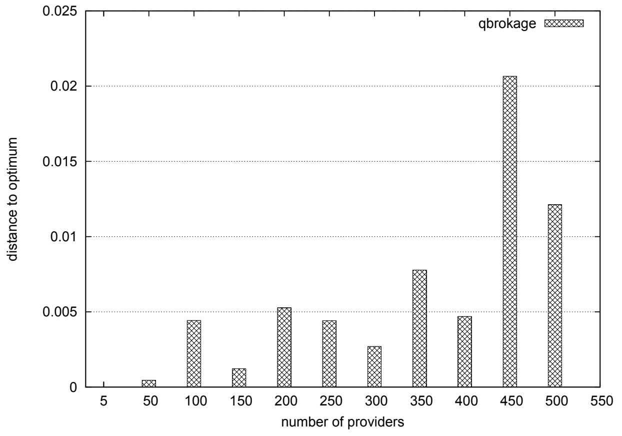
In this experiment we study how QBROKAGE scales in terms of distance-to-optimum and computation time when increasing the number of providers up to 500. This experiment was run with $R_{cross} = 0.35$, $P_{mut} = 1/10, S_P = 50$ and 120 evolution steps. Figure 5 shows the results for the computation time. It can be noticed that the time grows linearly with the number of providers. Considering the gathered results, QBROKAGE takes from 1.05 (50 providers) to 1.25 (500 providers) seconds to compute the mapping. In our opinion, this increment is not due to the genetic algorithm itself but it is due to the sorting performed on providers and thus requiring more time when increasing provider number. Although not extremely fast, it can considered still a tolerable delay for an interactive service. The computation time could be reduced by using less evolution steps and thus sacrificing precision (see Section 5.2) and/or by exploiting a parallel genetic algorithm.

The distance-to-optimum obtained for different number of providers has been plotted in Figure 6a. When changing the number of providers, we regenerate the application and providers by using ranges described in Table 6. Thus, the same generation of provider is used for each of the 20 runs regarding a certain number of providers. We refer to this generation mode as *lazy mode*. The plot seems to suggest that the considered configuration of QBROKAGE scales up to 400 providers. Indeed, until 400 providers the distance is always below 0.78% (worst case of 350 providers).

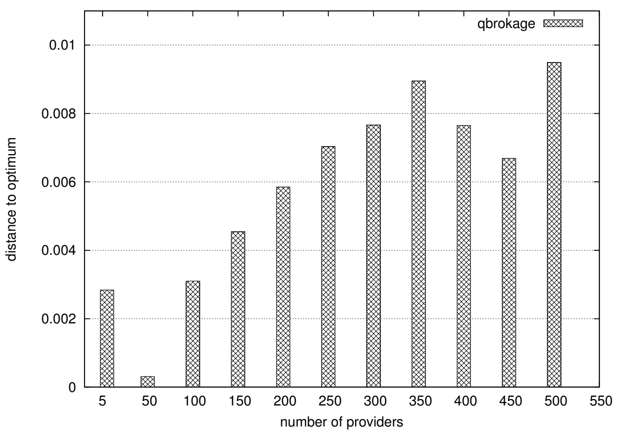
To further investigate this issue, we changed the provider generation mode, by using a different generation of providers for each of the 20 runs regarding a certain number of providers. We refer to this generation mode as *non-lazy mode*. The results, plotted in Figure 6b, show that the distance-to-optimum is always below the 1%, even for the maximum number of providers. When compared to the lazy mode experiment of Figure 6a, this may suggest that those spikes obtained when the number of providers assumes values $\{450, 500\}$ are more likely related to those particular datacenter generations, rather than to a scalability issue of QBROKAGE. Even in the non-lazy mode, the time trend for compute a mapping is very similar to the one depicted in Figure 5. In this case the maximum average time is always obtained with 500 providers and its equal to $1.34s$ with a deviation standard equal to 87.29.

5.4. Vendor Lock-in

In this experiment, we compare QBROKAGE with a naive approach in term of resilience to the vendor lock-in. The naive approach narrows down the provider number by considering those that meet the given QoS and then tries to map the application by only considering cost minimization. We measure the dispersion of the application by averaging the number of used providers in 20 independent runs, for a different number of suitable providers. In this experiment the providers and the costs are generated in the range described by Table 6, whilst the application is always generated by considering min values of such range,



(a) Distance to optimum, population size 50, up to 500 providers



(b) Distance to optimum when generating providers in non-lazy mode

Figure 6: Distance to optimum comparison for different number of providers

Table 7: Lock-in Degree

	QBrokage		Naive	
Suitable providers	Cost in currency	Used providers	Cost in currency	Used providers
50	0.082	1.50	0.081	1.0
100	0.078	2.35	0.077	1.0
150	0.082	3.20	0.080	1.0
200	0.075	3.50	0.073	1.0

in order to have all the providers as suitable for choice. Results can be seen in Table 7.

It can be noticed that QBROKAGE trades cost-effectiveness for the ability to use multiple providers. The increment of cost is limited, as the maximum difference in cost is +2.6% when considering 200 providers. However, this yields to an increment of +75% (still with 200 providers) in the number of used providers.

5.5. Global vs Local Fitness

In this experiment we consider scientific workflows that may benefit in different ways from running in a multi-cloud environment. For example, some tasks of the workflow application may have special demands in terms of QoS and/or QoP, which cannot be fulfilled by a single target Cloud. The same reasoning can be done in terms of performances. For these reasons, considering a multi-cloud scenario can be mandatory for running a workflow application. Moreover, the multi-cloud scenario can show its advantage in terms of cost-saving for the workflow users. In fact, since different tasks may have different requirements, simply choosing the cheapest provider for a certain resource cannot be cost-effective. This is the rationale behind the experiment reported in this section, where we show that placing VMs by only considering the cheapest providers for memory and storage lead to more expensive solutions w.r.t. considering also the network cost for the communication between VMs that can be placed in different providers.

In particular, we focus on the Montage [36] workflow because it is a well-known astronomy application, allowing for constructing custom image mosaics of the sky. Montage is characterized by a certain number of input files with relative large input size and a discrete computational load at each node, thus it is very suitable for studying the performance/cost ratio of a Cloud application, coming out by trading-off between the application requirements and the system resources offered by Cloud providers. The application is constructed starting from the so-called DAX representation produced by the WorkflowGenerator software[3]. A parser for this file type already exists and it is provided by WorkflowSim[4]. Thus, in this case we substitute our OVF parser with the WorkflowSim one, rather then customize it for dealing with such representation.

In any case, QBROKAGE needs a DAG representation of the application for computing network requirements and we create the application graph as depicted in Figure 7. To each node we associated a VM dedicated to execute the computational load – each VM is characterized by 6502.18 MIPS, 1 core, 1.7 GB of RAM and 160GB of storage. In particular, Figure 7 shows a Montage application characterized by 25 nodes and indicates the computation length of each vertex (in millions of instructions) inside each node, whilst the message size (in MB) transmitted on a communication link has been depicted on oriented edges connecting vertices. In order to highlight the benefit of considering the network, we consider three different version of the workflow, characterized by 25, 50 and 100 nodes.

We consider a node of the Montage workflow as the minimum allocation unit, thus in the context of the genetic brokering we represent each workflow node as a gene.

[3]See `https://github.com/pegasus-isi/WorkflowGenerator` for more information

[4]See also `https://github.com/WorkflowSim/WorkflowSim-1.0`

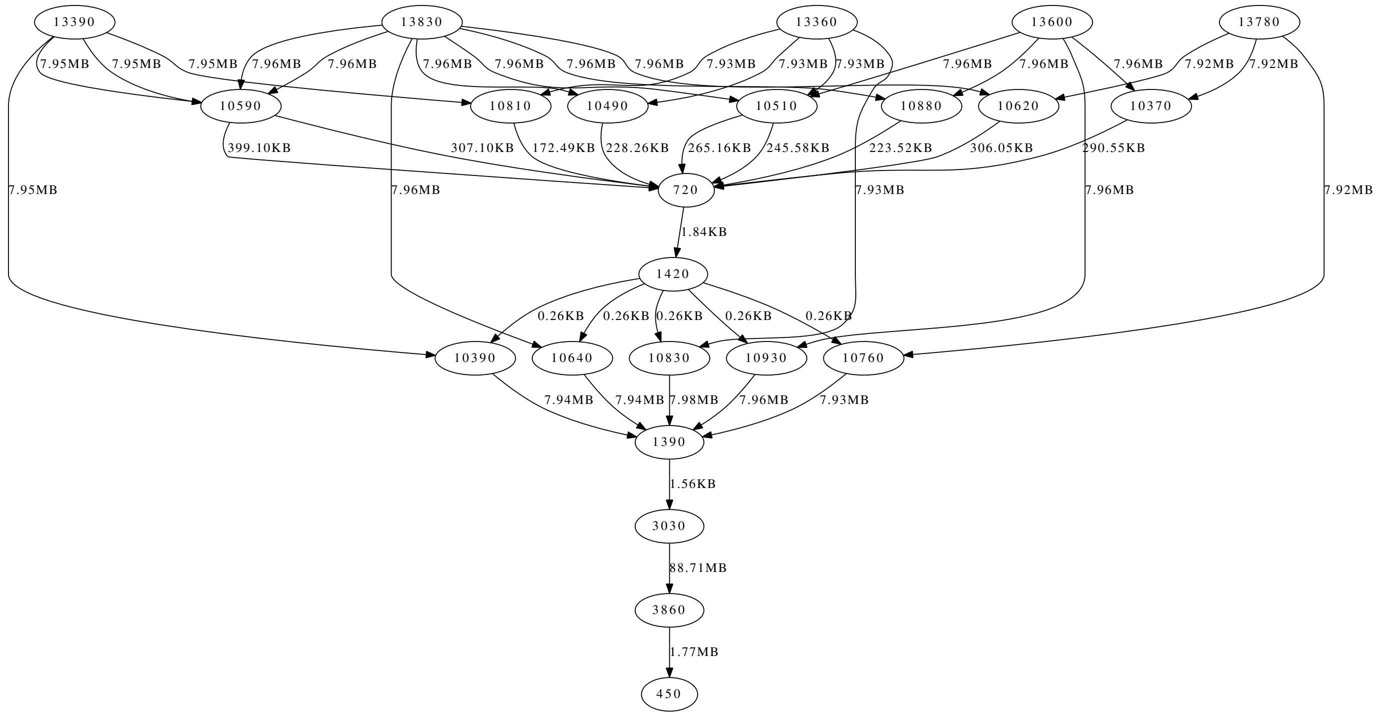


Figure 7: The Montage workflow with 25 nodes. The labels on edges represent volume of communication; The labels on vertexes represent computational time.

Its allele represents the provider used for the mapping. Please consider that we are interested to investigate our broker w.r.t. commercial Cloud providers (public clouds) where each provider has infinity capacity, i.e. we assume that each provider can always provide a worker node with the exposed capability for scheduling the task. Instead, for applying our approach to hybrid or private clouds, proper workflow scheduling and resource allocation algorithms should be used on the bottom level [37, 38]. For the same reason, in our experiment we make the assumption that each task on a particular level will run as soon as the tasks of the previous level complete their jobs, i.e. there is no scheduling delay.

We compared the mapping of the three Montage versions with two flavors of QBROKAGE. The *base* is our baseline version and it includes the following set of QoS attributes $\mathcal{Q} = \{costPerVm, ram, storage, location\}$, thus it does not consider the network, neither in terms of link bandwidth nor cost. The *networked* version instead considers in addition the cost and the bandwidth requirements when VMs are placed in different providers, i.e. they set-up an inter-provider communication. For intra-provider communication we make the realistic assumption that providers charge no cost and offer unlimited network bandwidth.

The *base* version employs only local constraints, which means that the local fitness function (see Section 4.3.1) is exploited during the brokering phase. The *networked* version employs global constraints in addition to the local one, exploiting the global fitness function (see Section 4.3.2) during the brokering phase, in order to consider network links in the evaluation, both in terms of costs and functional requirements. For fairly comparing the two QBROKAGE flavors, regardless of the workflow nature the cost is weighted more than the other parameters during the fitness evaluation, since it is modeled in both local and global versions.

To compare these two flavors of QBROKAGE we considered four metrics:

- the completion time (T_c), which counts the second a workflow needs to be completed. This value is influenced both by the capabilities of the host used for the tasks, and from the links among providers;
- the workflows per hour (w/h), which gives an estimation of the number of a workflows that is possible to run in an hour given its T_c;
- the cost per hour (c/h), which is computed by considering the cost of single run, in terms of host resources and network, multiplied by w/h;
- the brokering mapping time (T_m), as the time in millisecond necessary to QBROKAGE to provide the allocation plan.

We run experiments with 3 Montage DAGs, each composed respectively of 25, 50 and 100 nodes, by keeping the number of providers fixed to 100. The providers characteristics and costs were generated according to Table 6, with the addition of the network cost in the range of $[0.05, 0.15]\$$ per GB.

Table 8: Workflow applications: comparison of the base and networked QBROKAGE

Base	T_c	w/h	c/h	T_m
Montage_25	8.0	452.5	28.1	3642
Montage_50	9.5	378.5	51.1	6839.5
Montage_100	11.9	302.7	83.0	13770.2
Networked	T_c	w/h	c/h	T_m
Montage_25	8.1	446.8	18.4	3675.2
Montage_50	9.6	375.0	38.0	8707.1
Montage_100	12.0	300.3	70.6	18789.0

The results of the comparison are presented in Table 8 (each result is the average of 10 independent runs). Results show that the networked version of QBROKAGE yields advantages in terms of cost per hour. Indeed, in spite of a slightly T_c (and consequently a lower number of w/h) the gain in terms of cost is evident. With 25, 50 and 100 tasks the c/h is reduced respectively by the 35%, 25% and 15%. The reduction decreases with the number of tasks since we keep fixed the number of providers and thus the gaining space for the networked version to find better solution is reduced.

Further, the networked version yields longer mapping time than the base version, respectively by the 1%, 24% and 30%. This is explained by the fact that computing the fitness of a possible allocation (i.e. chromosome) with the networked version takes more time with respect to the base version. This is also confirmed by the fact that the difference in mapping time grows with the size of the application, and hence with the length of the chromosome.

From the analysis of the results, it is visible a tradeoff cost/mapping time between the base and networked versions. According to the data, larger Montage instances (roughly when number of nodes and providers of the same order of magnitude) benefit from the base version. Rather, smaller instances achieve lower costs and negligible increment in mapping time when employing the networked version.

6. Conclusion and Future Work

As Cloud Computing becomes a predominant trend, a growing amount of Cloud providers is joining the market, increasing heterogeneity and assortment of offered resources. Therefore, customers may find it hard to select the most suitable set of providers for acquiring resources needed by complex applications. In this paper we described QBROKAGE, a cloud broker that explores a large number of candidate solutions and choose those that meet the QoS requirements of the application. We proposed a genetic approach to the problem, because in our opinion it provides the adequate level of flexibility for supporting multiple and heterogeneous QoS constraints. Our proposal can deal with hundreds of providers, by preserving at the same the interactivity of the service. Moreover, it is capable of mitigating vendor lock-in risks by design. In addition, our architecture is capable of building graph representation of cloud applications and QBROKAGE is able to fully exploit such representation, by considering costs and capabilities of communication among providers. This enables to evaluate a wider-range of application-level QoS terms, such as throughput and response time.

While QBROKAGE performs well with static applications, some of them may require the ability to adjust the amount of computational resources during their life time, for instance by acquiring more resources at run time. Even if QBROKAGE is able to deal with custom-sized applications, its current version does not support elasticity for applications that have been already deployed. Such support is deferred to future work.

Acknowledgment

The authors acknowledge the support of the Project FP7-257438 Contrail (Open Computing Infrastructures for Elastic Services) and FP7-256980 NESSoS (Network of Excellence on Engineering Secure Future Internet Software Services and Systems). This work has been also partially funded by the following research projects: InGeoCLOUDS (no. 297300), MIDAS (no. 318786), E-CLOUD (no. 325091).

References

[1] Microsoft, Small and midsize businesses cloud trust study: U.S. study results, `http://www.microsoft.com/en-us/news/download/presskits/security/docs/twcjune13us.pdf`.

[2] R. Buyya, R. Ranjan, R. N. Calheiros, Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services, Algorithms and Architectures for Parallel Processing 6081/2010 (LNCS 6081) (2010) 20.

[3] E. Carlini, M. Coppola, P. Dazzi, L. Ricci, G. Righetti, Cloud federations in contrail, in: Euro-Par 2011: Parallel Processing Workshops, Vol. 7155 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2012, pp. 159–168.

[4] G. Anastasi, E. Carlini, M. Coppola, P. Dazzi, A. Lazouski, F. Martinelli, G. Mancini, P. Mori, Usage control in cloud federations, in: Cloud Engineering (IC2E), 2014 IEEE International Conference on, 2014, pp. 141–146. doi:10.1109/IC2E.2014.58.

[5] M. Coppola, P. Dazzi, A. Lazouski, F. Martinelli, P. Mori, J. Jensen, I. Johnson, P. Kershaw, The contrail approach to cloud federations, Proceedings of the International Symposium on Grids and Clouds (ISGC'12).

[6] Open Virtualization Format Specification, Version 1.1, Specification, DMTF (Jan. 2010).

[7] T. Cucinotta, G. Anastasi, L. Abeni, Respecting Temporal Constraints in Virtualised Services, in: Computer Software and Applications Conference, 2009. COMPSAC '09. 33rd Annual IEEE International, Vol. 2, 2009, pp. 73–78.

[8] H. Kavalionak, E. Carlini, L. Ricci, A. Montresor, M. Coppola, Integrating peer-to-peer and cloud computing for massively multiuser online games, Peer-to-Peer Networking and Applications 8 (2) (2015) 301–319.

[9] G. F. Anastasi, E. Carlini, M. Coppola, P. Dazzi, QBROKAGE: A Genetic Approach for QoS Cloud Brokering, in: Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on, IEEE, 2014, pp. 304–311.

[10] Smartfed, https://github.com/ecarlini/smartfed, Accessed 2015-12-8.

[11] I. Petri, M. Punceva, O. Rana, G. Theodorakopoulos, Broker Emergence in Social Clouds, in: Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on, 2013, pp. 669–676. doi:10.1109/CLOUD.2013.38.

[12] A. A. Falasi, M. A. Serhani, S. Elnaffar, The Sky: A Social Approach to Clouds Federation, Procedia Computer Science 19 (0) (2013) 131 – 138. doi:http://dx.doi.org/10.1016/j.procs.2013.06.022.

[13] P. Pawluk, B. Simmons, M. Smit, M. Litoiu, S. Mankovski, Introducing STRATOS: A Cloud Broker Service, in: Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on, 2012, pp. 891–898. doi:10.1109/CLOUD.2012.24.

[14] L. D. Ngan, R. Kanagasabai, OWL-S Based Semantic Cloud Service Broker, in: Web Services (ICWS), 2012 IEEE 19th International Conference on, 2012, pp. 560–567. doi:10.1109/ICWS.2012.103.

[15] L. D. Ngan, S. Tsai Flora, C. C. Keong, R. Kanagasabai, Towards a Common Benchmark Framework for Cloud Brokers, in: Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on, 2012, pp. 750–754. doi:10.1109/ICPADS.2012.121.

[16] M. Zhang, R. Ranjan, M. Menzel, S. Nepal, P. Strazdins, W. Jie, L. Wang, An infrastructure service recommendation system for cloud applications with real-time qos requirement constraints, IEEE Systems Journal PP (99) (2015) 1–11. doi:10.1109/JSYST.2015.2427338.

[17] F. Pop, V. Cristea, N. Bessis, S. Sotiriadis, Reputation guided genetic scheduling algorithm for independent tasks in inter-clouds environments, in: Proceedings of the 2013 27th International Conference on Advanced Information Networking and Applications Workshops, WAINA '13, IEEE Computer Society, Washington, DC, USA, 2013, pp. 772–776. doi:10.1109/WAINA.2013.206.

[18] Z. Zheng, R. Wang, H. Zhong, X. Zhang, An approach for cloud resource scheduling based on Parallel Genetic Algorithm, in: Computer Research and Development (ICCRD), 2011 3rd International Conference on, Vol. 2, 2011, pp. 444–447. doi:10.1109/ICCRD.2011.5764170.

[19] C. C. T. Mark, D. Niyato, T. Chen-Khong, Evolutionary optimal virtual machine placement and demand forecaster for cloud computing, in: Advanced Information Networking and Applications (AINA), 2011 IEEE International Conference on, IEEE, 2011, pp. 348–355.

[20] S. Iturriaga, S. Nesmachnow, B. Dorronsoro, E.-G. Talbi, P. Bouvry, A parallel hybrid evolutionary algorithm for the optimization of broker virtual machines subletting in cloud systems, in: P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2013 Eighth International Conference on, IEEE, 2013, pp. 594–599.

[21] L. Heilig, E. Lalla-Ruiz, S. Voß, A biased random-key genetic algorithm for the cloud resource management problem, in: Evolutionary Computation in Combinatorial Optimization, Springer, 2015, pp. 1–12.

[22] F. Jrad, J. Tao, I. Brandic, A. Streit, Sla enactment for large-scale healthcare workflows on multi-cloud, Future Generation Computer Systems 43 (2015) 135–148.

[23] Z. Wen, R. Qasha, Z. Li, R. Ranjan, P. Watson, A. Romanovsky, Dynamically partitioning workflow over federated clouds for optimising the monetary cost and handling run-time failures, IEEE Transactions on Cloud Computing PP (99) (2016) 1–1. doi:10.1109/TCC.2016.2603477.

[24] Y. Zhao, Y. Li, I. Raicu, S. Lu, C. Lin, Y. Zhang, W. Tian, R. Xue, A service framework for scientific workflow management in the cloud, Services Computing, IEEE Transactions on PP (99) (2014) 1–1. doi:10.1109/TSC.2014.2341235.

[25] E. Deelman, G. Singh, M. Livny, B. Berriman, J. Good, The cost of doing science on the cloud: The montage example, in: High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for, 2008, pp. 1–12. doi:10.1109/SC.2008.5217932.

[26] M. Malawski, G. Juve, E. Deelman, J. Nabrzyski, Algorithms for cost-and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds, Future Generation Computer Systems 48 (2015) 1–18.

[27] F. Jrad, J. Tao, A. Streit, A broker-based framework for multi-cloud workflows, in: Proceedings of the 2013 International Workshop on Multi-cloud Applications and Federated Clouds, MultiCloud '13, ACM, New York, NY, USA, 2013, pp. 61–68. doi:10.1145/2462326.2462339.
URL `http://doi.acm.org/10.1145/2462326.2462339`

[28] R. de C. Coutinho, L. M. Drummond, Y. Frota, D. de Oliveira, Optimizing virtual machine allocation for parallel scientific workflows in federated clouds, Future Generation Computer Systems 46 (0) (2015) 51 – 68. doi:http://dx.doi.org/10.1016/j.future.2014.10.009.
URL `http://www.sciencedirect.com/science/article/pii/S0167739X14002027`

[29] Y. C. Lee, H. Han, A. Y. Zomaya, M. Yousif, Resource-efficient workflow scheduling in clouds, Knowledge-Based Systems 80 (2015) 153–162.

[30] W. Chen, E. Deelman, Workflowsim: A toolkit for simulating scientific workflows in distributed environments, in: E-Science (e-Science), 2012 IEEE 8th International Conference on, 2012, pp. 1–8. doi:10.1109/eScience.2012.6404430.

[31] G. F. Anastasi, E. Carlini, P. Dazzi, Smart cloud federation simulations with cloudsim, in: Proceedings of the First ACM Workshop on Optimization Techniques for Resources Management in Clouds, ORMaCloud '13, ACM, New York, NY, USA, 2013, pp. 9–16. doi:10.1145/2465823.2465828.
URL `http://doi.acm.org/10.1145/2465823.2465828`

[32] G. F. Anastasi, E. Carlini, M. Coppola, P. Dazzi, M. Distefano, An ovf toolkit supporting inter-cloud application splitting, in: Cloud Networking (CLOUDNET), 2014 IEEE 1st International Conference on, 2014.

[33] Z. Ye, X. Zhou, A. Bouguettaya, Genetic Algorithm Based QoS-Aware Service Compositions in Cloud Computing, in: J. Yu, M. Kim, R. Unland (Eds.), Database Systems for Advanced Applications, Vol. 6588 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2011, pp. 321–334.

[34] Jgap: Java genetic algorithms package, http://jgap.sourceforge.net/, Accessed 2015-12-8.

[35] R. Calheiros, R. Ranjan, A. Beloglazov, C. De Rose, R. Buyya, Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Software: Practice and Experience 41 (1) (2011) 23–50.

[36] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, K. Vahi, Characterization of scientific workflows, in: Workflows in Support of Large-Scale Science, 2008. WORKS 2008. Third Workshop on, 2008, pp. 1–10. doi:10.1109/WORKS.2008.4723958.

[37] P. Hoenisch, S. Schulte, S. Dustdar, Workflow scheduling and resource allocation for cloud-based execution of elastic processes, in: Service-Oriented Computing and Applications (SOCA), 2013 IEEE 6th International Conference on, IEEE, 2013, pp. 1–8.

[38] P. Hoenisch, S. Schulte, S. Dustdar, S. Venugopal, Self-adaptive resource allocation for elastic process execution, in: Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on, IEEE, 2013, pp. 220–227.