Towards Knowledge-Based Assisted IaaS Selection

Kyriakos Kritikos and Kostas Magoutis and Dimitris Plexousakis
ICS-FORTH
Heraklion, Crete, Greece
Email: {kritikos,dp}@ics.forth.gr

***Abstract*—current PaaS platforms enable single or hybrid cloud deployments. However, such deployment types cannot best cover the user application requirements as they do not consider the great variety of services offered by different cloud providers and the effects of vendor lock-in. On the other hand, multi-cloud deployment enables selecting the best possible service among equivalent ones providing the best trade-off between performance and cost. In addition, it avoids cases of service level deterioration due to service under-performance as main effects of vendor lock-in. While many multi-cloud application deployment research prototypes have been proposed, such prototypes do not examine the effect that deployment decisions have on application performance. As such, they blindly attempt to satisfy low-level hardware requirements by neglecting the impact of allocation decisions on higher-level requirements at the component or application level. To this end, this paper proposes a new IaaS selection algorithm which, apart from being able to satisfy both low and high level requirements of different types, it also exploits deployment knowledge offered via reasoning over previous application execution histories to take the best possible allocation decisions. The experimental evaluation clearly shows that by considering this extra knowledge, more optimal deployment solutions are derived, able to maintain the service levels requested by users, in less solving time.**

***Keywords*-IaaS, selection, knowledge base, rules, evaluation, requirements, placement, location, security, performance, quality of service, deployment**

I. INTRODUCTION

Cloud computing changes the way applications are deployed by enabling an on-demand allocation and usage of cheap resources. This has revolutionised application deployment as apart from cost savings, it enables a more dynamic workload handling. Indeed, via following the on-demand scheme, applications can control the amount of resources reserved by demanding more resources when workload increases and releasing resources when workload is decreased.

To assist in application deployment and adaptive provisioning, the current platforms offer solutions focusing on a single public or hybrid clouds. Deployment is performed by first selecting those IaaS offerings of a cloud provider that better satisfy the user hardware requirements for each application component and then performing respective deployment actions. Adaptation is mainly offered via specifying and enforcing scalability rules which indicate when to scale-out/in the user application. Such addressing type is not adequate based on the following rationale. First, scalability rules are provider-specific and tend to locally solve the problem by not considering the whole application context. In fact, a single scale-out of a component is not the best possible panacea for all sorts of problems. Second, the application is locked-in in a particular cloud provider. As such, if the selected offerings of this provider change and are not satisfactory any more, it is hard to migrate the user application to a new cloud provider.

Recent research prototypes, mainly originating from European research projects, address the aforementioned issues via multi-cloud application deployment and provisioning. The main rationale is that multi-cloud deployment has better advantages than single-cloud deployment as: (a) the user is not locked-in; (b) the user application can be deployed on the best possible IaaS service combination coming from different cloud providers which leads to a better satisfaction of user requirements; (c) more diverse security requirements at the application component level can be addressed by resorting to different cloud providers able to satisfy them.

Most research prototypes follow a model-driven architecture promising to automate the various tasks involved in multi-cloud application management. However, a similar but inefficient approach to deployment and adaptation is commonly followed. During IaaS selection only hardware and cost requirements are considered while adaptation is mainly performed via scalability rules. The PaaSage European project is one exception to the latter which employs both a global and a local adaptation approach. The main rationale is that scalability rules address the local level and when local adaptation fails, then adaptation at the global level is performed by reconfiguring the user application.

To address this reconfiguration by also considering high-level user requirements, this paper proposes an IaaS selection approach advancing the state-of-the-art by: taking into account all possible user requirements of different types as well as allocation-to-performance mapping knowledge to drive the selection of better allocation decisions rather than following an almost blind approach. The types of requirements covered concern security, placement, location, cost and performance requirements. The mapping knowledge comes mainly from derivations of added-value knowledge drawn via a Knowledge Base operating over the execution history of multi-cloud applications in the form of best deployments of components or whole applications. This information can enable a faster solution time as it leads to a great reduction in the IaaS selection problem solution space. By exploiting this knowledge type, the gap between low and high-level requirements can be bridged while a more informed selection can be performed by relying on the actual performance exhibited for applications when being deployed in certain clouds. As such, the best possible solution can be guaranteed at almost each time point thus also catering for reconfiguration as bad solutions that might have been initially applied for applications are avoided in subsequent reconfigurations of the same or equivalent applications.

The experimental evaluation conducted on a typical IaaS selection algorithm indicates our approach superiority with respect to the optimality of the results obtained as well as to the speed up in solving time.

The rest of the paper is structured as follows. First, the next section provides an overall architecture of the IaaS selection system along with the analysis of the respective components and their interactions. Section 3 analyses the IaaS selection approach proposed. Section 4 explains the way the experimental evaluation was conducted and discusses the results produced. Section 5 reviews related work. The final section concludes the paper and draws directions for further research.

II. System Architecture

Figure 1 shows an overview of the IaaS selection system components (coloured in grey) in the context of the PaaSage prototype platform architecture. This selection system spans the PaaSage's *Upperware* and *Meta-Data DataBase* (*MDDB*) modules. The *Upperware* module transforms a user request into a set of deployment actions to be executed by the *Executionware* module. The *Upperware* and especially its *Adapter* component produce these deployment actions also in case of (global) application reconfiguration by computing the difference between the initial application state and the desired one. The *Executionware*, apart from executing respective deployment actions, monitors the application, informs its execution history, and locally-adapts it based on scalability rules. The *Meta-Data DataBase* is the main communication medium between the components as well as the storage place for different kinds of models, realised via a *Model Repository*. Any kind of model is specified by the CAMEL language, a multi-purpose DSL spanning different aspects in multi-cloud application management, including deployment, monitoring, requirements and security.

During a normal PaaSage flow in the context of a user request session, the user non-functional and deployment application requirements are sent to the *Reasoner* which transforms them into a constraint model that is then solved to produce a concrete multi-cloud application deployment plan. Our system is the actual realisation of such a Reasoner component. The concrete deployment plan produced is then used by the Adapter to compute the set of deployment actions to

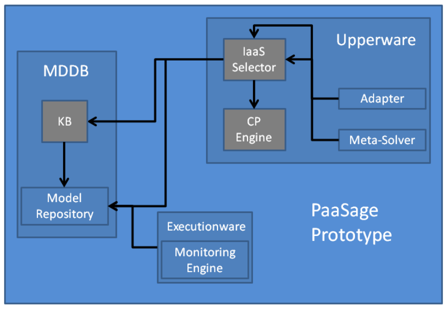

Figure 1. Architecture of PaaSage prototype including the components of the proposed approach

be performed. In case of global adaptation, the *Meta-Solver* component senses the respective need and coordinates the production of a new deployment plan.

Our internal system architecture comprises 4 main components. The *IaaS Selector* is responsible for delivering the main system functionality (IaaS selection). It retrieves information from the *KnowledgeBase* and the *Model Repository* mapping to the kind of extra knowledge needed to support more clever allocation decisions as well as to normal IaaS offerings advertised by different cloud providers. All this information along with user requirements lead to the construction of a constraint programming (CP) model which includes an optimisation formula that is sent for solving to the *CP Engine*. The respective solution is communicated back to the IaaS Selector which relays it to the *Adapter*.

The *KnowledgeBase* encompasses a set of rules operating over the *Model Repository* content and attempting to derive two main types of facts: (a) best deployments for applications and (b) their components. To be able to exploit information from equivalent applications or components, the *KnowledgeBase* also includes application/component equivalence derivation rules. Currently, this derivation mainly relies on name and category matching. As soon as more information is available for each application/component in the *Model Repository*, a more rich and complete way to match application/components will be developed focusing on interface matching. The *KnowledgeBase* enforces the rules according to a repetitive schedule of 1 hour. This scheduling type avoids calling the *KnowledgeBase* for each IaaS selection request, as rule firing usually takes time, especially when operating over long execution histories. Moreover, inside one hour, the changes that might have been performed will not be so critical that could require, e.g., to invalidate the facts obtained from previous *KnowledgeBase* execution. As such, the IaaS Selector will just query the *KnowledgeBase* to obtain already derived facts.

Concerning implementation, the Java programming language has been used. The Drools Engine (www.drools.org) was used to realise the *KnowledgeBase*, while the Choco CP Engine (choco-solver.org) along with Ibex (www.ibex-lib.

org) for real-based constraints handling have been selected as state-of-the-art open-source CP tools. The communication with the *Model Repository* relied on *CDOClient*, offered as an open-source component in the PaaSage prototype. In case of the *KnowledgeBase*, its global java object inclusion mechanism in rule firing has been exploited to enable its communication with this repository by enabling the passing of a CDOSession (i.e., an existing session between the CDOClient and the *Model Repository* encapsulated by a CDOServer – see for more details about CDO technology) object. To also support the suitable modelling of facts, domain code was developed to represent them such that they can be passed into the results of KnowledgeBase queries. The code was carefully designed to represent small pieces of information that do not introduce communication delays in KnowledgeBase query answering. In particular, information already stored in the ModelRepository is not modelled but just referenced via the use of CDOID identifiers. As such, information that may be additionally needed for each information piece can be retrieved from the *Model Repository* via using these CDOIDs. For example, a best deployment is represented by a specific model including references to components and IaaS offerings which need to be fetched by the *IaaS Selector* to properly represent the respective information into the CP model to be generated.

III. Approach

A. Input

The main IaaS selection functionality is delivered by generating and then solving a CP model so as to discover the best possible solution to the current IaaS selection problem. This generation relies on three kinds of information: (a) IaaS offerings, (b) best deployment facts, and (c) user requirements. The user requirements are in the form of a cloud-provider-independent deployment model as well as non-functional constraints imposed on this deployment model. Such non-functional constraints span high-level security requirements in term of security controls, low-level security requirements in terms of security SLOs, normal performance SLOs, as well as location, placement and cost constraints. All such constraints can be posed at the global application level or at the local level of each application component. Moreover, the user provides optimisation objectives indicating those non-functional parameters whose values need to be optimised (e.g., minimise application response time). These objectives are assorted with weights indicating the relative importance that they have over each other. The Analytic Hierarchy Process [1] is followed to product such weights.

B. Pre-Filtering

Before the constraint optimisation problem is generated, a pre-filtering of the solution space is performed to reduce the actual solving time via following three main steps which are shortly analysed in the next three paragraphs.

The first pre-filtering step involves reducing the provider space by considering high-level security requirements in the form of security controls. Each cloud provider is able to support a sub-set of all possible security controls from standardised sets like the Cloud Control Matrix (CCM) one proposed by the Cloud Security Alliance (CSA). This sub-set should then be matched with the one required by the user so as to guarantee that the high-level user security concerns are met. For instance, the user might desire to know whether the cloud provider has a consistent unified framework for business continuity planning. This maps to the BCR-01 security control in CCM required to be supported by the cloud provider. The security control capabilities of cloud providers can be derived via the approach in [] which relies on the answering of a self-assessment questionnaire called CAIQ (developed again by CSA) which includes questions whose answering can lead to deriving whether a specific security control is supported by a cloud provider.

The second filtering step follows a constraint-based approach to filter the IaaS space according to hardware-related, location and cost local requirements for each component. As both hardware, location and cost capabilities and requirements are expressed as constraints, a non-functional matching approach can be employed. This is drawn from the work in [2] exploiting smart structures to speed-up the matching. In that work, the extremely fast and scalable *Unary* algorithm can be exploited. As hardware and cost requirements and capabilities map to numeric constraints, we just need to explain the way location constraints are handled. Location requirements and capabilities take the form of a set which includes as members specific countries and continents (e.g., $\{Ireland, Asia\}$. In this sense, the matching of such requirements and capabilities maps to set matching, i.e., that the requested location set is included in the location set of the IaaS offering, after the respective location capability set is enriched to include the continent per each individual country stated. For instance, if the original set is $\{Irenand\}$, then it can be enriched into $\{Ireland, Europe\}$ enabling it to be matched with location requirements of the following two forms: $\{Ireland\}$ and $\{Europe\}$.

The third filtering step relies on the approach in [3] (see more details there about the exact algorithm used). This approach advocates that there are usually Pareto optimal solutions per component which prevail and are solely selected in an optimisation problem. This can be explained by the fact that if one solution is better than the others in at least one non-functional parameter and equivalent with respect to the rest, then this solution will always be preferred from the others. By considering only Pareto optimal solutions, the solution space is significantly reduced. To support a more meaningful filtering by using this approach, we consider only the user high-level non-functional parameters. As cost is always associated to an IaaS offering, the only missing information is the mapping between the IaaS offering to the respective component performance.

Such information can be provided in the form of a function via component profiling [4]. In particular, the respective resources offered by a IaaS can be considered as input to a set of functions which can be used to derive the value of respective non-functional parameters for the application components. Such functions can actually explicate the exact points where the non-functional capabilities of a component are modified. This means that such points represent resource bounds over which such capabilities are either improved or deteriorated. For instance, consider that the component i, if 2 cores and 1024 GB of RAM are offered, then its execution time will be 10 seconds. However, if 4 cores and 2048 GB of RAM are offered instead, then the execution time will be reduced to 7 seconds. Any offerings with respective characteristics that lie between [2, 4) for the number of cores and [1024, 2048) for the GB of RAM will map to an execution time of 10 seconds.

C. Core IaaS Selection

Depending on the availability of added-value knowledge, different IaaS selection problem forms can be solved. In the case of a totally new application with components not matching any other component (more usual in initial situations), an extended constraint optimisation model called NORMAL is produced. In case the application is not similar to already existing ones but has components equivalent to existing ones, a more limited constraint optimisation model called COMPONENT is solved for which the candidate deployments for matched components are significantly filtered to contain only the best ones derived. Finally, in case that applications matching the current one exist and we have derived best deployments for them, a simple optimisation model named as APPLICATION is generated which attempts to select the most optimal from the best application deployments according to the user requirements posed.

By considering the weights provided by the user, a single objective optimisation model formulation can be followed which maps to maximising the weighted sum of the application of (non-functional) parameter-specific utility functions applied over the respective global parameter values for the whole application. As such, multi-objective optimisation is transformed into single-objective one enabling to both satisfy the user preferences and avoid solving a more complex optimisation problem. The single optimisation objective considered is formulated as follows:

$$\max\left(\sum_{q=1}^{Q} w_q * uf_q\left(v_q\right)\right)$$

where Q is the number of all non-functional parameters, q is the index of the p-th parameter, uf_q is the parameter's utility function and v_q is the global value of the solution for this parameter.

The utility functions format depends on the respective parameter's monotonicity. It has been selected according to the approach in [5] which enables the catering of over-constrained user requirements such that solutions are produced to minimise as much as possible the number of user constraints violated. The utility functions are expressed according to the following compact expression:

$$uf_q\left(x\right)=\begin{cases} a_q+\frac{v_q^{max}-x}{v_q^{max}-v_q^{min}}\cdot\left(1-a_q\right), & v_q^{min}\leq x\leq v_q^{max}\wedge q\downarrow \\ a_q+\frac{x-v_q^{min}}{v_q^{max}-v_q^{min}}\cdot\left(1-a_q\right), & v_q^{min}\leq x\leq v_q^{max}\wedge q\uparrow \\ \left(a_q-\frac{v_q^{min}-x}{v_q^{max}-v_q^{min}}\cdot\left(1-a_q\right),0\right), & x<v_q^{min} \\ \left(a_q-\frac{x-v_q^{max}}{v_q^{max}-v_q^{min}}\cdot\left(1-a_q\right),0\right), & x>v_q^{max} \end{cases}$$

The expression indicates that the utility function returns a utility from a_q to 1.0 when the non-functional parameter value is between the user bounds v_q^{min} and v_q^{max}, while the utility drops from a_q to 0.0 otherwise. a_q represents the elastic violation factor which can take different values depending on the non-functional parameter such that for more significant parameters we allow less violation degrees and for less significant ones increased violation degrees.

The main decision variables take the form of x_{ijk} indicating whether the offering k from provider j can be used to deploy the application component i. This holds for most of the optimisation model forms apart from the APPLICATION one. In the latter case, the decision variables take the form of x_i indicating whether the best deployment i is selected for the user application. Due to page restriction reasons and as the problem format for *APPLICATION* is quite simplified, from now on, the analysis concentrates on the rest optimisation problem forms which can be expressed in the same uniform way.

The core constraint set about the main decision variables indicates that the sum of the values of these variables should equal to 1 for each application component i. This constraint is expressed as follows: $\sum_j \sum k x_{ijk} = 1$. In the case of REDUCED_COMPONENT, the decisions are actually fixed (equal to 0) for those offerings that do not map to best component deployments. This means that less cloud providers are eligible and less IaaS offers from these providers can be selected.

The rest of the constraints in the optimisation problem focus on satisfying all user constraint types and bridging the gap between low and high-level requirements.

User placement requirements can take the following two positive forms (while their negative form is also supported): (a) *IaaS co-location*: two components should be placed in the same VM/IaaS; (b) *Cloud co-location*: two components should be placed in the same cloud.

IaaS co-location constraints can be expressed by the following constraint set: $x_{ijk} = 1 \rightarrow x_{i'jk} = 1$. This indicates that the same decision should be taken for two components i and $i^{'}$, i.e., the same cloud provider and offering should be selected. The negative form of this requirement is expressed by the following constraint set: $x_{ijk} = 1 \rightarrow x_{i^{'}jk}! = 1$.

Cloud co-location constraints can be expressed by the following constraint set: $\sum_j \sum_k x_{ijk} == 1 \rightarrow \sum_j \sum_k x_{i^{'}jk} == 1$. This indicates that if j cloud provider is selected for component i, the same provider should be selected for component $i^{'}$. This requirement's negative form can be expressed similarly as follows: $\sum_j \sum_k x_{ijk} == 1 \rightarrow \sum_j \sum_k x_{i^{'}jk}! = 1$.

The global cost value for the whole application is derived from the following constraint: $v_q = \sum_i \sum_j \sum_k (x_{ijk} * v_qjk)$, where v_{qjk} represents the cost (indexed by parameter q) for the k IaaS offering of provider j. This indicates that the sum of each component cost equals the application cost, while each component's cost maps to the cost of the IaaS selected to host it.

While cost is additive, the remaining parameters, spanning performance and security aspects, may not be. In general, we expect that there is a parameter-specific function enabling us to map respective parameter requirements from the component to the application level. As such, the respective overall parameter value for the whole application can be expressed as a function over this parameter's values across all application components as follows: $v_q = f_q(v_{qi})$, where f_q is the function mapping to parameter q and v_{qi} represents this parameter's value for component i.

As an example of such a function, consider the non-functional parameter of availability. This parameter is multiplicative. This means that the function will map to the product of availability of each application component as follows: $f_q(v_{qi}) = \prod_i v_{qi}$.

Function f_q can always be determined. First, as for many non-functional parameters, the function can be quite obvious. Second, in case it is not, different techniques can be applied. For example, the function can be derived from the application's workflow. In this case, parameter aggregation patterns mapping to different workflow elements can be recursively exploited to define a complex function to be used for producing the overall parameter value. In case the application is not workflow-based, the respective function can be learned or derived by applying statistic-based techniques [4]. The latter, as already stated, can also be used to bridge the gap between the low-level hardware capabilities of IaaS offerings and the component non-functional capabilities.

As such, the respective functions are piece-wise linear and can be expressed via the following generic form:

$$v_{qi} = f_{qi}(R_{jk}) = \begin{cases} v_1, R_{jk} \leq r_1 \\ v_2, r_1 < R_{jk} \leq r_2 \\ \cdots \\ v_n, r_{n-1} < R_{jk} \leq r_n \end{cases}$$

R_{jk} represents the resources set provided by the selected IaaS offering j of provider k for component i, v_t with $1 \leq t \leq n$ are the distinct exact values the non-functional parameter q will take when the selected IaaS resources are bounded between the r_{t-1} and r_t resource bounds / points.

By considering the example of execution time, the respective function could be expressed as:

$$v_{qi} = f_{qi}(R_{jk}) = \begin{cases} 10, R_{jk} \leq [2, 1024] \\ 7, [2, 1024] < R_{jk} \leq [4, 2048] \end{cases}$$

D. Discussion

The proposed IaaS selection approach can be guaranteed to quickly produce optimal results as it significantly reduces the solution space by including: (a) various pre-filtering steps to filter both the IaaS and cloud provider space; (b) additional knowledge about best deployments for components and applications. Fast solving time is also guaranteed as the constraint optimisation model has much more constraints than the number of variables. The proposed approach relies on using CP techniques able to also deal with non-linear functions. The use of such functions is essential to bridge the gap between selected IaaS capabilities and non-functional application component capabilities and between component and application capabilities. This is a characteristic not exhibited by other state-of-the-art approaches which attempt to more or less blindly perform IaaS selection without considering high-level user requirements.

IV. Evaluation

The mail experimental evaluation goal was to assess the proposed algorithms' performance and accuracy. As such, a typical application was considered that can run in the cloud called SugarCRM (www.sugarcrm.com). This application comprises 3 main components: (a) a component realising the main business logic; (b) a container for hosting the business logic component; (c) a relational database (MySQL). For this application, certain hardware requirements are designated for each component to guarantee its proper functioning. We also employed application profiling to infer functions mapping each hardware capabilities profile into different component performance capabilities. Furthermore, we deployed the application in the cloud by relying of a great variety of deployment options that were enforced via the PaaSage prototype system. This allowed us to produce the application's respective execution history.

We have relied on the [1] marketplace from which we have drawn the respective offerings from a multitude of public cloud providers. These offerings specification was restricted to well-known hardware/VM characteristics (number of cores, size of main memory and hard disk) along with the respective cost.

[1]

Based on the gathered offerings, a respective framework was developed which controls both the number of providers and of offerings per provider so as to perform the experimental evaluation in a controlled manner. This framework also randomly produces the application requirements based however on practical ranges that have been derived according to the application execution history. The evaluation metrics supported are: (a) average solving time per algorithm and (b) average accuracy per algorithm. The second metric is calculated by dividing the overall utility of the solution produced by the algorithm with the ideal utility. The latter is produced via an exhaustive approach attempting to assess the application performance under all possibilities.

An experiment comprises executing a series of steps mapping to changing a specific control parameter's value. Each experiment step is performed 20 times to alleviate for any kind of OS interference. This maps to producing 20 raw metric values per algorithm which are then averaged to produce the algorithm's metric value for the current step. The experiments were performed in a laptop with: 1.8 GHz dual core CPU, 500GB of storage and 6 GB of RAM.

The algorithms considered were the 3 ones proposed plus a baseline mapping to a typical IaaS selection algorithm (called *RESOURCE*) that considers only hardware requirements and capabilities by also attempting to minimise cost. Due to the unavailability of respective code, no state-of-the-art algorithms were considered in the evaluation.

A. *Experiment Results*

Due to space limitations, the results of just one experiment are reported. This experiment involves evaluating the algorithms' performance and accuracy by increasing the number of offerings per cloud provider, from 10 to 30 with 5 as step of increase. Figures 2 and 3 depict the respective results.

For each aspect, the corresponding results are quite expected. In particular, by considering solving time, the *NORMAL* algorithm has the worst possible performance which is very distant from the solving times of the other algorithms. The performance between *COMPONENT* and *RESOURCE* is close but *RESOURCE* is slightly worse as it has a bigger candidate set for all application components. Obviously, *APPLICATION* has the best performance as it just needs to evaluate the overall utility of a small set of solutions. In addition, it seems to be the most scalable algorithm. The worst scalability is exhibited by *NORMAL*, as expected, due to the exponential complexity of the respective solving technique exploited.

Concerning accuracy, *APPLICATION* is the best algorithm having average accuracy around 0.99, quite close to the most optimal solution. The latter solution could be produced only in case the application's execution history was rich enough to include all possible deployment options. This cannot be expected to take place for a single user but for a

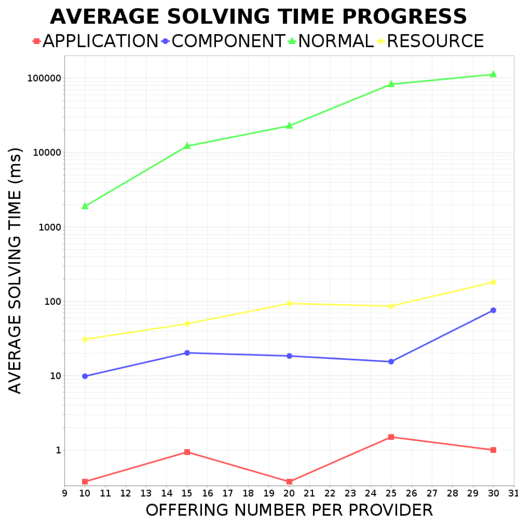

Figure 2. Experiment results on solving time

multitude of users in the same system with different application requirements. By also considering the *APPLICATION*'s solving time, we can deduce that the best possible trade-off between performance and accuracy has been achieved. The second best algorithm is *COMPONENT*. This is natural due to the following reasons: (a) not all possible deployment options are examined, so this algorithm will not have an ideal accuracy; (b) by combining best deployments for components, it is not certain that the respective application deployment is the best. For instance, two components may be derived to be deployed on the same IaaS offering. However, by doing so, there will be interference between them resulting in a degradation of the expected performance leading to a non-optimal deployment. As such, only overall meaningful combinations of component deployments can lead to optimal application deployments.

The third best algorithm is *NORMAL*. This is due to the fact that benchmarking information (or any kind of deployment knowledge) was not taken into account. As such, a set of equivalent IaaS offerings with respect to their hardware capabilities lead always to the same application component performance based on the functions derived from application profiling. However, as argued and proved by many research approaches, equivalent IaaS offerings in different clouds usually lead to variations in a component performance. Thus, by neglecting deployment knowledge, such variations are also neglected which leads to low accuracy results.

The worst algorithm with respect to accuracy is *RESOURCE*. This is expected as this algorithm does not consider the effect that a certain IaaS offering can have on a component performance. On the contrary, it just considers cost in order to perform the optimisation. This leads to also neglecting the other non-functional application requirements, including response time and throughput. Indeed, by inspect-

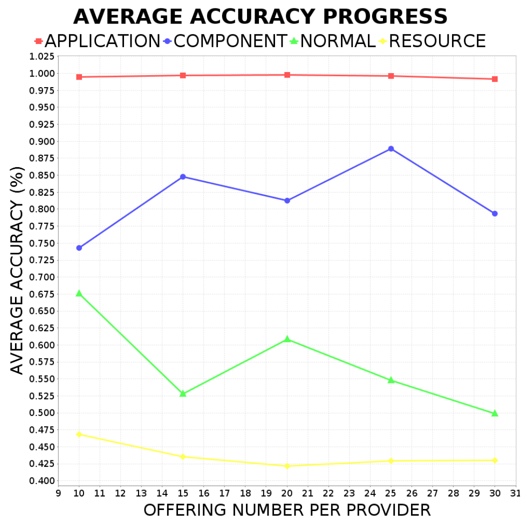

Figure 3. Experiment results on accuracy

ing the solutions produced by this algorithm, the other non-functional application requirements are severely violated.

Concerning the accuracy behaviour, the inclusion of additional deployment alternatives per component does not influence the *APPLICATION* and *COMPONENT* algorithms. On the other hand, we can see an accuracy degradation for the rest of the algorithms. For *RESOURCE*, this is quite natural as more options mean more possibilities to reduce overall cost by also neglecting the rest of the non-functional requirements. This results in adopting quite cheap IaaS offerings leading to quite worse component performance. As *NORMAL* cannot discern between equivalent IaaS offerings from different clouds, the same effect actually applies as the more cheaper alternatives will always be preferred. In addition, the great drop in that algorithm accuracy actually reveals this effect impact, indicating that *NORMAL*'s accuracy tends to become equal to *RESOURCE* one.

B. Discussion

The empirical evaluation results produced reveal quite interesting facts. First, the addition of execution history knowledge does lead to a great increase in accuracy which is also more robust to any kind of interference from the IaaS offering space (e.g., increase in offers with same hardware capabilities). Second, the clever exploitation of such knowledge also leads to a great solving time reduction. Better levels of scalability are also attained, especially with respect to the *APPLICATION* algorithm. Third, any algorithm that does consider hardware requirements and even their cloud-independent mapping to application/component performance cannot reach great accuracy levels. In fact, its accuracy drops with the increase in the number of deployment alternatives. However, we believe that the benchmark-oriented knowledge consideration can lead to increasing the accuracy of such algorithms. This does not necessarily mean, though, that solving time can be greatly decreased for a CP-based algorithm. Based on the above analysis, we definitely recommend designing IaaS selection algorithms that do exploit both execution history and benchmarking knowledge to increase their accuracy as well as reach better performance levels. Such algorithms can be both used at design and possibly runtime in order to (dynamically) produce the best possible IaaS selection solution which is more robust to any kind of contextual modification.

V. Related Work

Many research approaches [6]–[8] focus on placing VMs in a certain cloud with the ultimate goal to increase the cloud provider gains. Some VM placement approaches are even more sophisticated [9] by considering federated clouds where one cloud can sub-contract part of its workload on a different but partnering cloud. The VM placement problem is similar to that of IaaS selection but not actually equivalent. It also caters for satisfying the provider and not the user view and respective requirements. However, some ideas and techniques are certainly similar or equivalent and are certainly re-used in the context of both problems.

In [10], a dynamic approach to application/service placement is followed accounting for the dynamic nature for both the demand and the IaaS service cost. This approach applies control and game-theoretic models to dynamically find the best application placement based on the hosting cost by considering fluctuations in application demand and IaaS price plus the competition between application providers against a specific set of resources. While this approach is dynamic and considers additional information aspects with respect to our work, it does not consider other optimisation criteria apart from cost and does not exploit other added-value knowledge forms that could enhance solution optimality. In addition, it does not focus on reducing the application placement time.

In [11], there is an evaluation of application/service placement approaches in dynamic pricing scenarios. This evaluation highlights that exhaustive-based approaches return always the most optimal results but take quite long to execute. On the other hand, greedy-based approaches deliver surprisingly fast sub-optimal but satisfactory results so they constitute the best compromise between solution quality and execution time. The evaluation results produced by this work are indeed valid. However, we believe that the suitability of the algorithm also lies on the exact phase of the application lifecycle on which it is applied. At design time, optimal solutions are preferred with the main rationale that such solutions are more robust when IaaS services are under-performing. On the other hand, application re-configuration should be fastly detected and performed such that the service levels exhibited by the application are not deteriorated for a long time leading to SLA violations and a decrease in application provider gains. As such, exhaustive approaches should be used at design time and greedy-based approaches at runtime. Our approach can be considered as exhaustive but is carefully applied such that the solution space becomes quite limited. In this way, it can guarantee a faster solving time, enabling its application at runtime.

[12] focuses mainly on the more accurate representation of cloud provider cost models and proposes a cost-model application placement approach over federated clouds comprising one private and many public clouds. This approach relies on a brute-force algorithm examining the cost of all possible application/service placements. As such, while the approach can be more accurate in capturing an IaaS service cost, it is not suitable to be applied even at design time due to its huge solution space. However, the cost model proposed by this work could be adopted by our IaaS selection approach to become even more accurate.

The semantic approach in [13] deals with the optimised selection of both virtual appliances and virtual machines on which these appliances can be placed by considering deployment cost and time, the compatibility between the different cloud services selected and the composition's global reliability in terms of the SLA confidence level metric, evaluating cloud provider reliability with respect to the SLA guarantees promised and the respective execution history exhibited. The optimised selection addresses the multi-objective optimisation problem by using evolutionary techniques on the quest to find Pareto-optimal solutions. This approach produces the formal user preferences indirectly by applying fuzzy logic over linguistic terms expressing such preferences. It also addresses an aspect not considered in our work mapping to the compatibility of virtual appliances to virtual machines. However, our approach can indirectly address this aspect by imposing constraints on additional characteristics of IaaS offerings. Our work also does not consider the overall deployment time. We regard that this is an important parameter that has to be optimised which needs special addressing as: (a) the deployment time in different clouds can vary so there should be the means to derive it for all clouds; (b) this parameter's value depends on the deployment order between the application components – this information must be derived by considering the communication requirements between these components in the overall application deployment plan. Fortunately, CAMEL exhibits the respective modelling capability needed.

Our work can address any kind of non-functional parameter and not just cost and reliability as in the above approach. This enables our work to consider additional high-level requirements that enable to perform IaaS selection by accounting for the actual service level to be exhibited by the application. In addition, it is exhaustive in nature, thus able to provide optimal results in contrast to the sub-optimal ones delivered by this approach due to the use of evolutionary algorithms. Last but not least, the solutions derived are optimal also based on the user preferences while in a Pareto-optimal approach multiple solutions can be considered optimal but only some will actually best satisfy these preferences.

The learning-based algorithm in [14] is another IaaS selection alternative in PaaSage. It relies on a combined stochastic programming and learning approach to solve the IaaS selection problem by considering that initial configuration solutions might be blindly selected by taking into account only hardware and cost requirements but then as the application execution goes on, bad solutions are recorded and avoided in the future. As such, the application configuration will gradually reach an optimum from which no further reconfiguration can be performed unless respective application context is modified (e.g., requirement change or new IaaS offerings). This approach solves the IaaS selection problem by also considering previous execution history knowledge. However, its current problem is that it considers cost as the main optimisation criterion and thus cannot bridge the gap between the IaaS capabilities offered and the high-level application requirements posed. Thus, it would be interesting to see how this approach could evolve to bridge this gap. This could enable evaluating both approaches to see which one is the best possibly in different situations.

VI. Conclusions

This paper has proposed a novel IaaS selection approach which attempts not to blindly select those IaaS services that satisfy local hardware requirements such that the overall cost user optimisation requirement is met. On the contrary, this approach attempts to bridge the gap between the IaaS capabilities selected and the overall non-functional capabilities at the application level. As such, low-level and high-level requirements are bridged allowing the user to express additional optimisation objectives that can better account for the usual trade-off between application performance and cost. Apart from this, the proposed approach can deal with a variety of different user requirements, spanning location, component placement, security, cost and performance requirements. This also makes the approach even more appealing to users that do not desire to be restricted with providing optimisation objectives on a certain subset of all possible non-functional parameters but also place suitable constraints that can impact the actual deployment solution. The proposed approach is also quite fast in execution time and accurate. This is mainly due to: (a) the advanced pre-filtering of the solution space and (b) the exploitation of knowledge assisting in selecting those solution parts that have been deemed as best according to the execution history of the current or equivalent applications.

The experimental evaluation conducted against typical IaaS selection work highlights the main benefits of the proposed approach: (a) it is much faster and (b) produces results which are more optimal.

The following research directions are planned. First, a thorough approach evaluation by also considering the effect that some types of requirements have on solving time. Second, connecting the deployment of an application with its design in terms of SaaS selection. Third, applying the combined SaaS and IaaS approach within the PaaSage prototype and the respective use cases to validate and better highlight the main approach benefits in terms of both application deployment and reconfiguration in real circumstances.

References

[1] T. Saati, *The Analytic Hierarchy Process*. McGraw-Hill, 1980.

[2] K. Kritikos and D. Plexousakis, "Novel Optimal and Scalable Nonfunctional Service Matchmaking Techniques," *IEEE T. Services Computing*, vol. 7, no. 4, pp. 614–627, 2014.

[3] Y. Chen, J. Huang, C. Lin, and J. Hu, "A Partial Selection Methodology for QoS-Aware Web Service Composition," *IEEE Transactions on Services Computing*, vol. 8, no. 3, pp. 384–397, 2015.

[4] P. Xiong, C. Pu, X. Zhu, and R. Griffith, "vPerfGuard: An Automated Model-driven Framework for Application Performance Diagnosis in Consolidated Cloud Environments," in *ICPE*. New York, NY, USA: ACM, 2013, pp. 271–282.

[5] A. M. Ferreira, K. Kritikos, and B. Pernici, "Energy-Aware Design of Service-Based Applications," in *ICSOC*, ser. LNCS. Springer, 2009.

[6] J. Dong, X. Jin, H. Wang, Y. Li, P. Zhang, and S. Cheng, "Energy-Saving Virtual Machine Placement in Cloud Data Centers ," in *CCGrid*. IEEE/ACM, 2013, pp. 618 – 624.

[7] E. Casalicchio, D. A. Menascé, and A. Aldhalaan, "Autonomic Resource Provisioning in Cloud Systems with Availability Goals, booktitle = Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference, series = CAC '13, year = 2013, isbn = 978-1-4503-2172-3, location = Miami, Florida, USA, pages = 1:1–1:10, articleno = 1, numpages = 10, url = http://doi.acm.org/10.1145/2494621.2494623, doi = 10.1145/2494621.2494623, acmid = 2494623, publisher = ACM, address = New York, NY, USA, keywords = VM allocation, autonomic computing, cloud computing,."

[8] D. Jayasinghe, C. Pu, T. Eilam, M. Steinder, I. Whally, and E. Snible, "Improving performance and availability of services hosted on iaas clouds with structural constraint-aware virtual machine placement," in *Proceedings of the 2011 IEEE International Conference on Services Computing*, ser. SCC '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 72–79. [Online]. Available: http://dx.doi.org/10.1109/SCC.2011.28

[9] D. Breitgand, A. Mararschini, and J. Tordsson, "Policy-Driven Service Placement Optimization in Federated Clouds," IBM, Research Report, February 2011.

[10] Q. Zhang, Q. Zhu, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, "Dynamic Service Placement in Geographically Distributed Clouds," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 10, pp. 1–11, 2013.

[11] W. Li, P. Svärd, J. Tordsson, and E. Elmroth, "Cost-optimal cloud service placement under dynamic pricing schemes," in *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, ser. UCC '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 187–194. [Online]. Available: http://dx.doi.org/10.1109/UCC.2013.42

[12] J. Altmann and M. M. Kashef, "Cost Model Based Service Placement in Federated Hybrid Clouds," *Future Gener. Comput. Syst.*, vol. 41, no. C, pp. 79–90, Dec. 2014. [Online]. Available: http://dx.doi.org/10.1016/j.future.2014.08.014

[13] A. V. Dastjerdi and R. Buyya, "Compatibility-aware Cloud Service Composition Under Fuzzy Preferences of Users," *IEEE Transactions on Cloud Computing*, vol. 2, no. 1, pp. 1–13, 2014. [Online]. Available: http://dx.doi.org/10.1109/TCC.2014.2300855

[14] G. Horn, "A vision for a stochastic reasoner for autonomic cloud deployment," in *Second Nordic Symposium on Cloud Computing & Internet Technologies (NordiCloud '13)*. ACM, September 2013, pp. 46–53. [Online]. Available: http://doi.acm.org/10.1145/2513534.2513543